Progetto_programmazione_data_intensive

July 1, 2020

Programmazione di Applicazioni Data Intensive
Laurea in Ingegneria e Scienze Informatiche
DISI - Università di Bologna, Cesena

Luca Marini, Simone Mattioli

1 Classificazione: Determinare la posizione di un calciatore in base alle sue caratteristiche

FIFA 19 è un videogioco di calcio sviluppato da EA Sports, pubblicato il 28 settembre 2018 per PlayStation 3 (con supporto PlayStation Move), PlayStation 4, Xbox 360, Xbox One (con Kinect), Microsoft Windows e Nintendo Switch. fifa19_wikipedia

Definizione del problema

Il progetto ha lo scopo di classificare il ruolo di un giocatore in base alle sue caratteristiche fisiche e tecniche.

 # Librerie

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
import matplotlib.patches as mpatches

import sklearn
import pickle
from scipy.stats import norm
import seaborn as sns
import os.path
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, Perceptron
from sklearn.svm import SVC
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
from sklearn.dummy import DummyClassifier
from google.colab import files

pd.set_option('max_rows', 99999)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31:
FutureWarning: The module is deprecated in version 0.21 and will be removed in
version 0.23 since we've dropped support for Python 2.7. Please rely on the
official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144:
FutureWarning: The sklearn.neighbors.base module is  deprecated in version 0.22
and will be removed in version 0.24. The corresponding classes / functions
should instead be imported from sklearn.neighbors. Anything that cannot be
imported from sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
```

1.0.1 Caricamento dei dati

[2]:
```
if not os.path.exists("fifa.csv"):
    from urllib.request import urlretrieve
    urlretrieve("https://raw.githubusercontent.com/amanthedorkknight/
 ↪fifa18-all-player-statistics/master/2019/data.csv", "fifa.csv")
```

[3]:
```
fifa = pd.read_csv("fifa.csv")
```

2 Esplorazione dei dati

Il dataset comprende: * 18207 diverse istanze, ognuna rappresentante un giocatore * ad ogni giocatore è associato un insieme di 89 colonne contengono informazioni anagrafiche, statistiche, caratteristiche fisiche e tecniche e il ruolo

[4]:
```
len(fifa)
```

[4]: 18207

[5]:
```
len(fifa.columns)
```

[5]:
```
89
```

[6]:
```
fifa.head(3)
```

[6]:
```
   Unnamed: 0      ID  ... GKReflexes  Release Clause
0           0  158023  ...        8.0         €226.5M
1           1   20801  ...       11.0         €127.1M
2           2  190871  ...       11.0         €228.1M

[3 rows x 89 columns]
```

[7]:
```
fifa.tail(3)
```

[7]:
```
       Unnamed: 0      ID  ... GKReflexes  Release Clause
18204       18204  241638  ...       13.0           €165K
18205       18205  246268  ...        9.0           €143K
18206       18206  246269  ...        9.0           €165K

[3 rows x 89 columns]
```

Di seguito sono riportate le dimensioni in memoria, il numero di istanze non nulle e il tipo delle feature che compongono i dati raccolti nel dataset

[8]:
```
fifa.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18207 entries, 0 to 18206
Data columns (total 89 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Unnamed: 0                18207 non-null  int64
 1   ID                        18207 non-null  int64
 2   Name                      18207 non-null  object
 3   Age                       18207 non-null  int64
 4   Photo                     18207 non-null  object
 5   Nationality               18207 non-null  object
 6   Flag                      18207 non-null  object
 7   Overall                   18207 non-null  int64
 8   Potential                 18207 non-null  int64
 9   Club                      17966 non-null  object
 10  Club Logo                 18207 non-null  object
 11  Value                     18207 non-null  object
 12  Wage                      18207 non-null  object
 13  Special                   18207 non-null  int64
 14  Preferred Foot            18159 non-null  object
 15  International Reputation  18159 non-null  float64
 16  Weak Foot                 18159 non-null  float64
 17  Skill Moves               18159 non-null  float64
```

```
18  Work Rate                 18159 non-null  object
19  Body Type                 18159 non-null  object
20  Real Face                 18159 non-null  object
21  Position                  18147 non-null  object
22  Jersey Number             18147 non-null  float64
23  Joined                    16654 non-null  object
24  Loaned From               1264 non-null   object
25  Contract Valid Until      17918 non-null  object
26  Height                    18159 non-null  object
27  Weight                    18159 non-null  object
28  LS                        16122 non-null  object
29  ST                        16122 non-null  object
30  RS                        16122 non-null  object
31  LW                        16122 non-null  object
32  LF                        16122 non-null  object
33  CF                        16122 non-null  object
34  RF                        16122 non-null  object
35  RW                        16122 non-null  object
36  LAM                       16122 non-null  object
37  CAM                       16122 non-null  object
38  RAM                       16122 non-null  object
39  LM                        16122 non-null  object
40  LCM                       16122 non-null  object
41  CM                        16122 non-null  object
42  RCM                       16122 non-null  object
43  RM                        16122 non-null  object
44  LWB                       16122 non-null  object
45  LDM                       16122 non-null  object
46  CDM                       16122 non-null  object
47  RDM                       16122 non-null  object
48  RWB                       16122 non-null  object
49  LB                        16122 non-null  object
50  LCB                       16122 non-null  object
51  CB                        16122 non-null  object
52  RCB                       16122 non-null  object
53  RB                        16122 non-null  object
54  Crossing                  18159 non-null  float64
55  Finishing                 18159 non-null  float64
56  HeadingAccuracy           18159 non-null  float64
57  ShortPassing              18159 non-null  float64
58  Volleys                   18159 non-null  float64
59  Dribbling                 18159 non-null  float64
60  Curve                     18159 non-null  float64
61  FKAccuracy                18159 non-null  float64
62  LongPassing               18159 non-null  float64
63  BallControl               18159 non-null  float64
64  Acceleration              18159 non-null  float64
65  SprintSpeed               18159 non-null  float64
```

```
 66  Agility                 18159 non-null  float64
 67  Reactions               18159 non-null  float64
 68  Balance                 18159 non-null  float64
 69  ShotPower               18159 non-null  float64
 70  Jumping                 18159 non-null  float64
 71  Stamina                 18159 non-null  float64
 72  Strength                18159 non-null  float64
 73  LongShots               18159 non-null  float64
 74  Aggression              18159 non-null  float64
 75  Interceptions           18159 non-null  float64
 76  Positioning             18159 non-null  float64
 77  Vision                  18159 non-null  float64
 78  Penalties               18159 non-null  float64
 79  Composure               18159 non-null  float64
 80  Marking                 18159 non-null  float64
 81  StandingTackle          18159 non-null  float64
 82  SlidingTackle           18159 non-null  float64
 83  GKDiving                18159 non-null  float64
 84  GKHandling              18159 non-null  float64
 85  GKKicking               18159 non-null  float64
 86  GKPositioning           18159 non-null  float64
 87  GKReflexes              18159 non-null  float64
 88  Release Clause          16643 non-null  object
dtypes: float64(38), int64(6), object(45)
memory usage: 55.9 MB
```

3 Significato delle feature

Di seguito sono riportate le informazioni sul Data Set utilizzato

Statistiche generali:

- `Name`: nome del giocatore
- `Age`: età giocatore
- `Nationality`: nazionalità giocatore
- `Overall`: potenziale del giocatore calcolato sulla base delle sue statistiche chiave (es. velocità, passaggio, tiro, fisico, dribbling, difesa per i giocatori non portieri, altrimenti viene aggiunta la statistica "riflessi") e la sua reputazione internazionale.
- `Potential`: potenziale Overall di miglioramento
- `Club`: club di appartenenza
- `Value`: valore monetario del giocatore
- `Wage`: stipendio del giocatore
- `Preferred Foot`: piede preferito
- `International Reputation`: fama del giocatore a livello internazionale su scala da 0 a 5
- `Weak Foot`: abilità di utilizzo del piede debole su una scala da 0 a 5
- `Skill Moves`: abilità di utilizzo di finte per il dribbling
- `Work Rate`: profensione che il giocatore ha nel lavorare per la squadra in fase offensiva/difensiva

- `Position`: posizione ricoperta dal giocatore
- `Jersey Number`: numero di maglia del giocatore
- `Height`: altezza giocatore
- `Weight`: peso giocatore
- `Photo`: url contenente l'immagine del giocatore
- `Contract Valid Until`: anno di termine del contratto del giocatore
- `Flag`: url contenente l'immagine della nazione del giocatore
- `Joined`: data in cui il giocatore si è unito al club
- `Club Logo`: url contenente l'immagine del logo del club
- `Body Type`: tipo di corpo
- `Real Face`: specifica se il viso virtuale del calciatore è ottenuto da una scansione di quello del calciatore reale
- `Loaned From`: club precedente di appartenenza
- `ID, Unnamed: 0`: identificatori
- `valutazione da 0 a 100 di quanto un giocatore è adatto a un ruolo specifico`: "LS", "ST", "RS", "LW", "LF", "CF", "RF", "RW", "LAM", "CAM", "RAM", "LM", "LCM", "CM", "RCM", "RM", "LWB", "LDM", "CDM", "RDM", "RWB", "LB", "LCB", "CB", "RCB", "RB", "Special"

Statistiche tecniche:

Le statistiche tecniche sono espresse da un numero che va da 0 a 99 e maggiore è il numero, più un calciatore ha padronanza di quella statistica.

- `Crossing`: abilità nel cross
- `Finishing`: abiltà nel tiro in porta
- `HeadingAccuracy`: abilità nel colpo di testa
- `ShortPassing`: anilità nel passaggio corto
- `Volleys`: abilità nel tiro al volo
- `Dribbling`: abilità nel dribbling
- `Curve`: abilità nel tiro a giro
- `FKAccuracy`: abilità sui calci di punizione
- `LongPassing`: abilità sui passaggi lunghi
- `BallControl`: controllo di palla
- `Acceleration`: accelerazione
- `SprintSpeed`: velocità di scatto
- `Agility`: agilità
- `Reactions`: abilità nel reagire prontamente alle situazioni
- `Balance`: equilibrio
- `ShotPower`: potenza di tiro
- `Jumping`: abilità nel saltare
- `Stamina`: resitenza
- `Strength`: forza fisica
- `LongShots`: abilità nel tiro dalla distanza
- `Aggression`: aggressività
- `Interception`: abilità nell'intercettare il pallone
- `Positioning`: abilità nel posizionarsi in campo
- `Vision`: abilità nella visione di gioco
- `Penalties`: abilità del giocatore sui calci di rigore

- `Composure`: freddezza del giocatore sotto pressione
- `Marking`: abilità nel marcare gli avversari
- `StandingTackle`: abilità nell'effettuare contrasti da in piedi
- `SlidingTackle`: abilità nell'effettuare contrasti in scivolata
- `GKDiving`: abilità del portiere nel tuffo
- `GKHandling`: abilità del portiere nel trattenere il pallone
- `GKKicking`: abilità del portiere nel calciare il pallone
- `GkPositioning`: abilità del portiere nel posizionamento in porta
- `GKReflexes`: riflessi del portiere

#Variabile da predire

Definizione delle classi

La variabile da predire è `Position`, ovvero la posizione del giocatore. Questa colonna contiene 27 diversi valori. Per semplificare il problema si è deciso di diminuire il numero da 27 a 8 ruoli più generali: * `Goalkeeper`: Portiere * `Central Back`: Difensore centrale * `Left Back`: Terzino sinistro * `Right Back`: Terzino destro * `Central Midfielder`: Centrocampista centrale * `Central Defensive Midfielder`: Mediano * `Forward`: Attaccante * `Wing`: Ala

[9]:
```
# Lista delle 27 posizioni
fifa['Position'].unique()
```

[9]:
```
array(['RF', 'ST', 'LW', 'GK', 'RCM', 'LF', 'RS', 'RCB', 'LCM', 'CB',
       'LDM', 'CAM', 'CDM', 'LS', 'LCB', 'RM', 'LAM', 'LM', 'LB', 'RDM',
       'RW', 'CM', 'RB', 'RAM', 'CF', 'RWB', 'LWB', nan], dtype=object)
```

[10]:
```
#portieri
fifa['Position'] = fifa['Position'].replace(dict.fromkeys(['GK'], 'Goalkeeper'))

#difensori
fifa['Position'] = fifa['Position'].replace(dict.fromkeys(['CB', 'RCB', 'LCB'],␣
 ↪'Central Back'))
fifa['Position'] = fifa['Position'].replace(dict.fromkeys(['LB', 'LWB'], 'Left␣
 ↪Back'))
fifa['Position'] = fifa['Position'].replace(dict.fromkeys(['RB', 'RWB'], 'Right␣
 ↪Back'))

#centrocampisti
fifa['Position'] = fifa['Position'].replace(dict.fromkeys(['RCM','LCM','CM',␣
 ↪'CAM'], 'Central Midfilder'))
fifa['Position'] = fifa['Position'].replace(dict.fromkeys(['RDM','LDM','CDM'],␣
 ↪'Central Defensive Midfilder'))

#attaccanti
fifa['Position'] = fifa['Position'].replace(dict.fromkeys(['ST',␣
 ↪'CF','LF','RF'], 'Forward'))
```

```
fifa['Position'] = fifa['Position'].replace(dict.fromkeys(['RS', 'RW','RM'␣
 ↪,'LW', 'RAM','LAM', 'LS','LM'],'Wing'))

fifa['Position'].unique()
```

[10]:
```
array(['Forward', 'Wing', 'Goalkeeper', 'Central Midfilder',
       'Central Back', 'Central Defensive Midfilder', 'Left Back',
       'Right Back', nan], dtype=object)
```

4 Feature scarsamente utili

Si è deciso di eliminare alcune colonne che non portano alcuna informazione utile al raggiungimento dell'obiettivo preposto:

- `Photo`: url contenente l'immagine del giocatore
- `Name`: nome del giocatore
- `Club`: club al quale appartiene il giocatore
- `Nationality`: nazionalità del giocatore
- `Contract Valid Until`: anno di termine del contratto del giocatore
- `Flag`: url contenente l'immagine della nazione del giocatore
- `Joined`: data in cui il giocatore si è unito al club
- `Club Logo`: url contenente l'immagine del logo del club
- `Body Type`: tipo di corpo
- `Real Face`: specifica se il viso virtuale del calciatore è ottenuto da una scansione di quello del calciatore reale
- `Loaned From`: club precedente di appartenenza
- `ID, Unnamed: 0`: identificatori
- `Special`

Queste ultime feature sono state eliminate, poichè avrebbero reso banale la classificazione del ruolo.

[11]:
```
fifa = fifa.drop(columns=["Name", "Nationality", "Club","Photo",
                          "Contract Valid Until", "Flag", "Club Logo",
                          "Body Type", "Real Face", "Loaned From",
                          "Release Clause", "Joined", "ID", "Unnamed: 0",
                          "Special"])
```

Ora il dataset è composto da 74 colonne

[12]:
```
len(fifa.columns)
```

[12]:
```
74
```

[13]:
```
fifa.head(3)
```

[13]:
```
    Age  Overall  Potential  ... GKKicking GKPositioning GKReflexes
0   31       94         94  ...      15.0          14.0        8.0
```

```
1   33       94         94  ...      15.0          14.0       11.0
2   26       92         93  ...      15.0          15.0       11.0

[3 rows x 74 columns]
```

[14]:
```
fifa.describe()
```

[14]:
```
                Age       Overall  ...  GKPositioning    GKReflexes
count  18207.000000  18207.000000  ...   18159.000000  18159.000000
mean      25.122206     66.238699  ...      16.388898     16.710887
std        4.669943      6.908930  ...      17.034669     17.955119
min       16.000000     46.000000  ...       1.000000      1.000000
25%       21.000000     62.000000  ...       8.000000      8.000000
50%       25.000000     66.000000  ...      11.000000     11.000000
75%       28.000000     71.000000  ...      14.000000     14.000000
max       45.000000     94.000000  ...      90.000000     94.000000

[8 rows x 41 columns]
```

4.1 Gestione dei valori mancanti

Elenco delle colonne del Dataframe che contengono dei valori nulli:

[15]:
```
nullseries = fifa.isnull().sum()
print(nullseries[nullseries > 0])
```

```
Preferred Foot               48
International Reputation     48
Weak Foot                    48
Skill Moves                  48
Work Rate                    48
Position                     60
Jersey Number                60
Height                       48
Weight                       48
LS                         2085
ST                         2085
RS                         2085
LW                         2085
LF                         2085
CF                         2085
RF                         2085
RW                         2085
LAM                        2085
CAM                        2085
RAM                        2085
LM                         2085
```

LCM	2085
CM	2085
RCM	2085
RM	2085
LWB	2085
LDM	2085
CDM	2085
RDM	2085
RWB	2085
LB	2085
LCB	2085
CB	2085
RCB	2085
RB	2085
Crossing	48
Finishing	48
HeadingAccuracy	48
ShortPassing	48
Volleys	48
Dribbling	48
Curve	48
FKAccuracy	48
LongPassing	48
BallControl	48
Acceleration	48
SprintSpeed	48
Agility	48
Reactions	48
Balance	48
ShotPower	48
Jumping	48
Stamina	48
Strength	48
LongShots	48
Aggression	48
Interceptions	48
Positioning	48
Vision	48
Penalties	48
Composure	48
Marking	48
StandingTackle	48
SlidingTackle	48
GKDiving	48
GKHandling	48
GKKicking	48
GKPositioning	48
GKReflexes	48

```
dtype: int64
```

- Si può notare che molte colonne del Dataframe hanno un numero di valori mancanti uguale a 48, e visto che la maggior parte di questi valori nulli sono essenziali (rappresentano le abilità di un giocatore), allora si è deciso di eliminare queste 48 righe presenti nel dataframe
- Sono state rimosse tutte le righe che non avevano un valore nella colonna 'Position'
- Dopo aver applicato queste modifiche si può vedere che nel DataFrame non sia più presente alcun valore nullo

[16]:
```
fifa = fifa.dropna(subset=['Position'])
nullseries = fifa.isnull().sum()
print(nullseries[nullseries > 0])
```

```
LS     2025
ST     2025
RS     2025
LW     2025
LF     2025
CF     2025
RF     2025
RW     2025
LAM    2025
CAM    2025
RAM    2025
LM     2025
LCM    2025
CM     2025
RCM    2025
RM     2025
LWB    2025
LDM    2025
CDM    2025
RDM    2025
RWB    2025
LB     2025
LCB    2025
CB     2025
RCB    2025
RB     2025
dtype: int64
```

Rimangono 2025 istanze che contengono valori nulli nelle colonne di valutazioni da 0 a 100 di quanto un giocatore è adatto a un ruolo specifico. Si scopre che tutte queste tuple sono i portieri.

[17]:
```
for role in fifa.loc[:, 'LS':'RB'].columns:
  zero_players = fifa[fifa[role].isnull()]
zero_players['Position'].unique()
```

[17]:
```
array(['Goalkeeper'], dtype=object)
```

[18]:
```
fifa.loc[fifa['Position'] == 'Goalkeeper'].head(3)
```

[18]:
```
    Age  Overall  Potential  ... GKKicking GKPositioning GKReflexes
3    27       91         93  ...      87.0          88.0       94.0
9    25       90         93  ...      78.0          88.0       89.0
18   26       89         92  ...      88.0          85.0       90.0

[3 rows x 74 columns]
```

Quindi si sostituiscono tutti questi valori assenti con degli zeri.

[19]:
```
fifa = fifa.fillna(0)
```

[20]:
```
nullseries = fifa.isnull().sum()
print(nullseries[nullseries > 0])
```

```
Series([], dtype: int64)
```

[21]:
```
fifa.head()
```

[21]:
```
   Age  Overall  Potential  ... GKKicking GKPositioning GKReflexes
0   31       94         94  ...      15.0          14.0        8.0
1   33       94         94  ...      15.0          14.0       11.0
2   26       92         93  ...      15.0          15.0       11.0
3   27       91         93  ...      87.0          88.0       94.0
4   27       91         92  ...       5.0          10.0       13.0

[5 rows x 74 columns]
```

Come si può notare, ora non esiste più alcun dato nullo.

4.2 Conversione di dati

In seguito si è passati alla conversione dei valori contenuti nelle seguenti colonne: * `Weight`: il peso dei giocatori è stato convertito da una stringa espressa in libbre ad un float che rappresenta i chilogrammi * `Height`: è stata modificata l'unità di misura dell'altezza, da foot (piedi) a cm (centimetri) * `Value`: il valore di un giocatore è espresso da una stringa, tramite un numero e una lettera che indica quanti zeri devono essere aggiunti al numero stesso (ad es. K = * 1000), quindi lo si è trasformato in un unico numero * `Wage`: stesso discorso di Value, ma applicato allo stipendio del calciatore * `WorkRate`: è stata divisa in due colonne separando il lavoro in fase offensiva da quello in fase difensiva, ed è stato convertito il valore da stringa (Low, Medium, High) a intero (1, 2, 3) * `Preferred Feet`: la colonna di stringhe è stata convertita in una colonna di booleani: True -> Right, False -> Left, e per chiarezza la feature è stata rinominata in `Prefers Right Foot` * `Valutazioni da 0 a 100 di quanto un giocatore è adatto a un ruolo specifico`: queste colonne, rappresentate da una stringa formata da 2 numeri separati da un '+', sono state trasformate in un intero

[22]:
```
# funzione che converte il peso dei giocatori da libbre a chilogrammi

def from_lbs_to_kgs(value):
  out = value.replace('lbs', '')
  return float(out) * 0.45

fifa['Weight'] = fifa['Weight'].apply(lambda x : from_lbs_to_kgs(x))
fifa['Weight'].head()
```

[22]:
```
0    71.55
1    82.35
2    67.50
3    75.60
4    69.30
Name: Weight, dtype: float64
```

[23]:
```
# funzione che converte l'altezza dei giocatori in centimetri partendo dalla
# stessa espressa in piedi

def from_foot_to_cm(value):
  h = value.split("'")
  h_ft = int(h[0])
  h_inch = int(h[1])
  h_inch += h_ft * 12
  h_cm = round(h_inch * 2.54, 1)
  return h_cm

fifa['Height'] = fifa['Height'].apply(lambda x : from_foot_to_cm(x))
fifa['Height'].head()
```

[23]:
```
0    170.2
1    188.0
2    175.3
3    193.0
4    180.3
Name: Height, dtype: float64
```

[24]:
```
# funzione con la quale si ottengono i valori monetari come numeri (in euro)
# partendo da una stringa

def get_accurate_money_number(Value):
    out = Value.replace('€', '')
    if 'M' in out:
        out = float(out.replace('M', ''))*1000000
    elif 'K' in Value:
        out = float(out.replace('K', ''))*1000
    return float(out)
```

```
fifa['Value'] = fifa['Value'].apply(lambda x: get_accurate_money_number(x))
fifa['Wage'] = fifa['Wage'].apply(lambda x: get_accurate_money_number(x))

fifa['Wage'].head()
fifa['Value'].head()
```

[24]:
```
0    110500000.0
1     77000000.0
2    118500000.0
3     72000000.0
4    102000000.0
Name: Value, dtype: float64
```

[25]:
```
# funzione che associa a ciascuna classe di rate un intero positivo
def from_string_to_int_rate(value):
  switcher = {
      'Low':1,
      'Medium':2,
      'High':3
  }
  return switcher.get(value,0)

WorkRate_separation = fifa['Work Rate'].str.split('/ ',1,expand = True)
WorkRate_separation[0] = WorkRate_separation[0].apply(lambda x :␣
 ↪from_string_to_int_rate(x))
WorkRate_separation[1] = WorkRate_separation[1].apply(lambda x :␣
 ↪from_string_to_int_rate(x))
fifa['WROffensive'] = WorkRate_separation[0]
fifa['WRDefensive'] = WorkRate_separation[1]
fifa.drop(columns = ['Work Rate'], inplace = True)
```

[26]:
```
#binariziamo la colonna 'Preferred Foot'
fifa = fifa.rename(columns={'Preferred Foot':'Prefers Right Foot'})
foot = {'Right':True,'Left':False}
fifa['Prefers Right Foot'] = fifa['Prefers Right Foot'].map(foot)
```

[27]:
```
for role in fifa.loc[:, 'LS':'RB'].columns:
    value = fifa[role].str.split('+',1,expand = True)
    fifa[role] = value[0]

fifa = fifa.fillna(0)

for role in fifa.loc[:, 'LS':'RB'].columns:
    fifa[role] = fifa[role].astype(int)
```

[28]:
```
fifa = fifa[[ 'Overall', 'Potential', 'LS', 'ST', 'RS', 'LW', 'LF', 'CF', 'RF',
              'RW', 'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM', 'RM', 'LWB',
              'LDM', 'CDM', 'RDM', 'RWB', 'LB', 'LCB', 'CB', 'RCB', 'RB','Value',
              'Wage','Age', 'WROffensive','WRDefensive',
              'International Reputation', 'Weak Foot', 'Skill Moves', 'Height',
              'Weight','Prefers Right Foot', 'Position', 'Crossing', 'Finishing',
              'HeadingAccuracy', 'ShortPassing', 'Volleys', 'Dribbling', 'Curve',
              'FKAccuracy', 'LongPassing', 'BallControl', 'Acceleration',
              'SprintSpeed', 'Agility', 'Reactions', 'Balance', 'ShotPower',
              'Jumping', 'Stamina', 'Strength', 'LongShots', 'Aggression',
              'Interceptions', 'Positioning', 'Vision', 'Penalties', 'Composure',
              'Marking', 'StandingTackle', 'SlidingTackle', 'GKDiving',
              'GKHandling', 'GKKicking', 'GKPositioning', 'GKReflexes',
              'Jersey Number']]
```

[29]:
```
fifa.head(2)
```

[29]:

	Overall	Potential	LS	...	GKPositioning	GKReflexes	Jersey Number
0	94	94	88	...	14.0	8.0	10.0
1	94	94	91	...	14.0	11.0	7.0

[2 rows x 75 columns]

5 Analisi dei dati

Di seguito viene visualizzato il numero di giocatori per ogni posizione ricoperta

[30]:
```
fifa["Position"].value_counts()
```

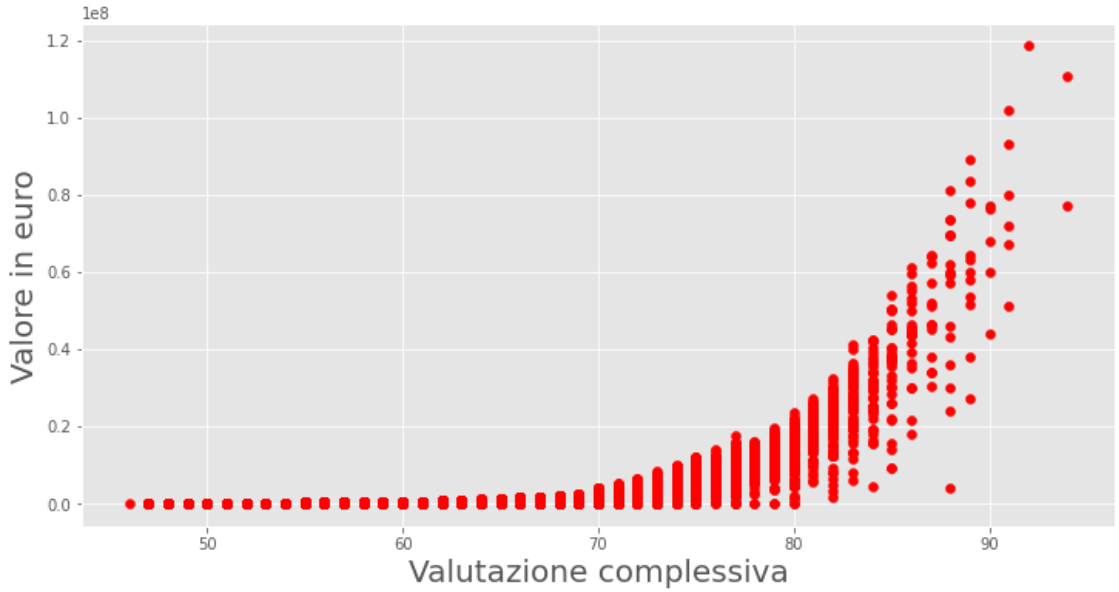
[30]:
```
Wing                          3422
Central Midfilder             3138
Central Back                  3088
Forward                       2257
Goalkeeper                    2025
Central Defensive Midfilder   1439
Left Back                     1400
Right Back                    1378
Name: Position, dtype: int64
```

[31]:
```
plt.style.use('ggplot')
cs=cm.Set1(np.arange(8)/8.)
plt.figure(figsize=(8, 8))
fifa["Position"].value_counts().plot.pie(fontsize=14, colors=cs)
```

[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f74a091b748>
```

[32]:
```
plt.figure(figsize=(12, 6))
plt.scatter(fifa['Overall'], fifa['Value'], c = 'red');
plt.xlabel('Valutazione complessiva', fontsize = 20)
plt.ylabel('Valore in euro', fontsize = 20)
```

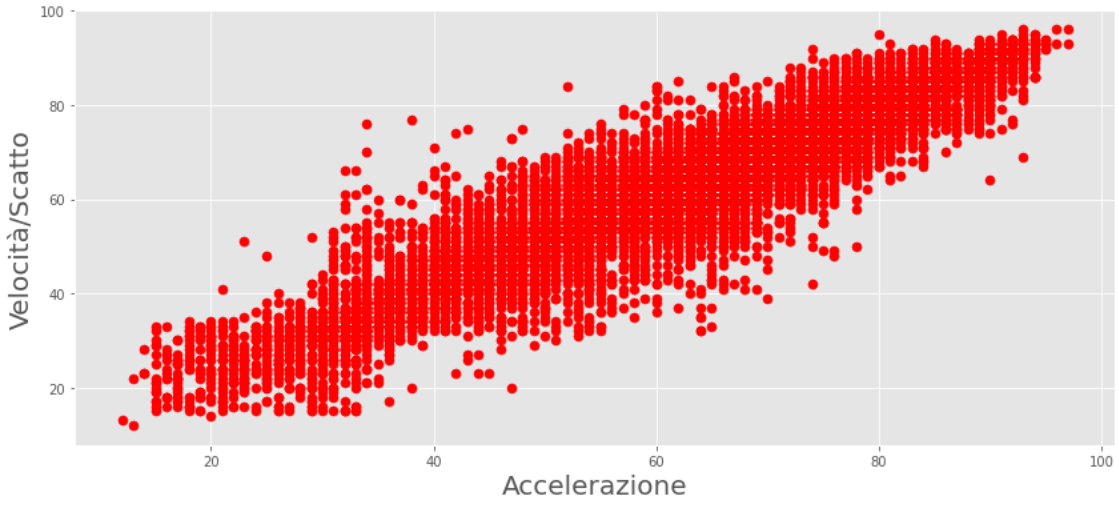
[32]:
```
Text(0, 0.5, 'Valore in euro')
```



Questo grafico evidenzia la correlazione, che ha un andamento simile a una funzione esponenziale, tra la valutazione complessiva di un giocatore e il suo valore espresso in euro.

[33]:
```
plt.figure(figsize=(14, 6))
plt.scatter(fifa['Acceleration'], fifa['SprintSpeed'], c = 'red', s = 50);
plt.xlabel('Accelerazione', fontsize = 20)
plt.ylabel('Velocità/Scatto', fontsize = 20)
```

[33]:
```
Text(0, 0.5, 'Velocità/Scatto')
```

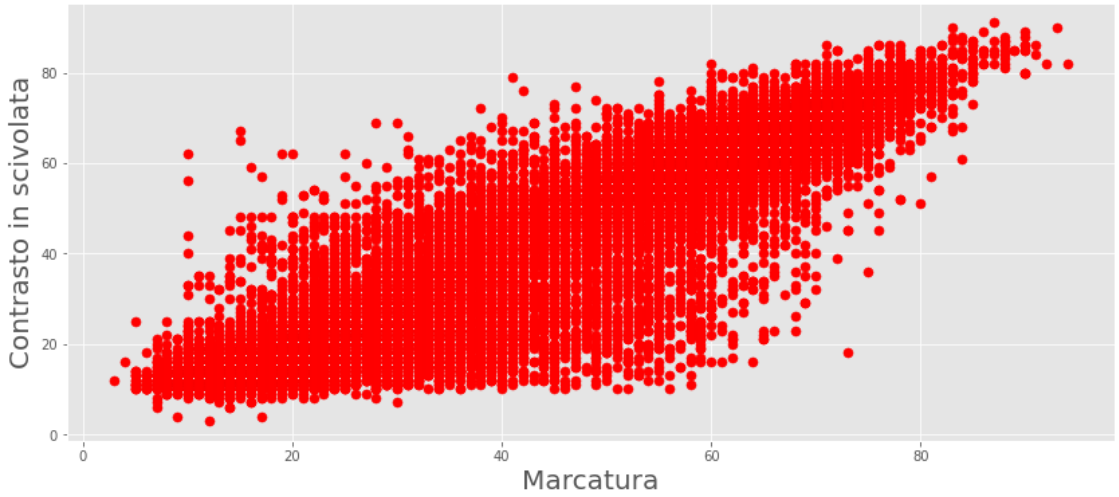


Il grafico riportato sopra mostra come al crescere dell'accelerazione aumenti anche la velocità di

un giocatore, e quindi si ha una proporzionalità diretta tra le due variabili.

[34]:
```
labels = fifa["Position"].unique()
plt.figure(figsize=(14, 6))
plt.scatter(fifa['Marking'], fifa['SlidingTackle'], c='red', s = 50);
plt.xlabel('Marcatura', fontsize = 20)
plt.ylabel('Contrasto in scivolata', fontsize = 20)
```

[34]:
```
Text(0, 0.5, 'Contrasto in scivolata')
```

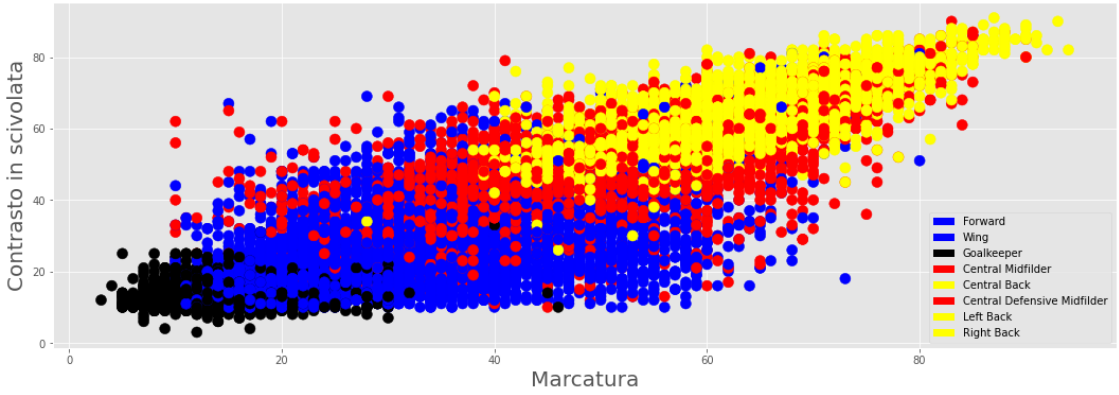


In ambito difensivo, in generale, l'abilità nel marcare è direttamente proporzionale all'abilità che un calciatore ha nel contrasto in scivolata.

[35]:
```
defending_skills_color_map = {"Wing": "blue", "Forward": "blue",
                              "Goalkeeper":"black",
                              "Central Midfilder":"red",
                              "Central Back":"yellow",
                              "Central Defensive Midfilder":"red",
                              "Left Back":"yellow", "Right Back":"yellow"}
defending_skills_colors = fifa["Position"].map(defending_skills_color_map)
plt.figure(figsize=(18, 6))
plt.scatter(fifa['Marking'], fifa['SlidingTackle'], c = defending_skills_colors,
            s = 100);
plt.xlabel('Marcatura', fontsize = 20)
plt.ylabel('Contrasto in scivolata', fontsize = 20)

classes = fifa['Position'].unique()
class_colours = defending_skills_color_map.values()
recs = []

for i in range(0,len(class_colours)):
```

```
    recs.append(mpatches.Rectangle((0,0),1,1,fc=list(class_colours)[i]))
plt.legend(recs,classes,loc=4)
```

[35]: <matplotlib.legend.Legend at 0x7f7486f7c6a0>

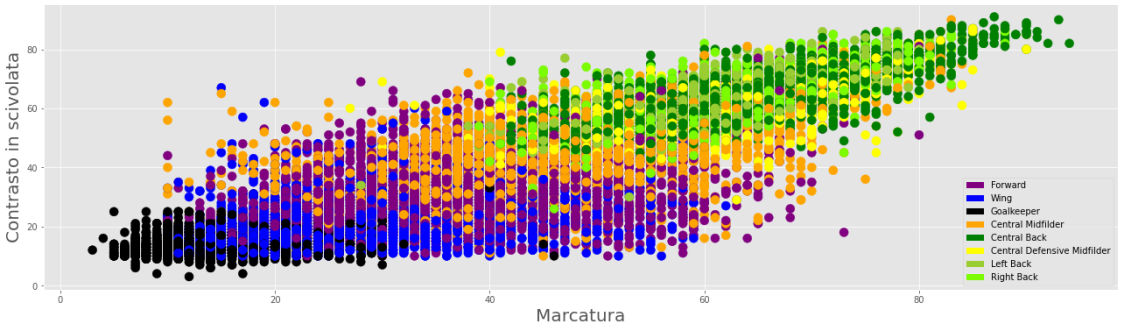


E raggruppando le 10 posizioni da esaminare nei 4 ruoli principali, ovvero: * Portiere (in nero) * Difensore (in blu) * Centrocampista (in rosso) * Attaccante (in blu)

quindi i giocatori più bravi nel difendere sono appunto i difensori, seguiti dai centrocampisti e dagli attaccanti, e infine ci sono i portieri.

[36]:
```
defending_skills_color_map = {"Wing": "purple", "Forward": "blue",
                              "Goalkeeper":"black",
                              "Central Midfilder":"orange",
                              "Central Back":"green",
                              "Central Defensive Midfilder":"yellow",
                              "Left Back":"yellowgreen",
                              "Right Back":"lawngreen"}
defending_skills_colors = fifa["Position"].map(defending_skills_color_map)
plt.figure(figsize=(22, 6))
plt.scatter(fifa['Marking'], fifa['SlidingTackle'], c = defending_skills_colors,
            s = 100);
plt.xlabel('Marcatura', fontsize = 20)
plt.ylabel('Contrasto in scivolata', fontsize = 20)

classes = fifa['Position'].unique()
class_colours = defending_skills_color_map.values()
recs = []

for i in range(0,len(class_colours)):
    recs.append(mpatches.Rectangle((0,0),1,1,fc=list(class_colours)[i]))
plt.legend(recs,classes,loc=4)
```

[36]: <matplotlib.legend.Legend at 0x7f7486fa40b8>

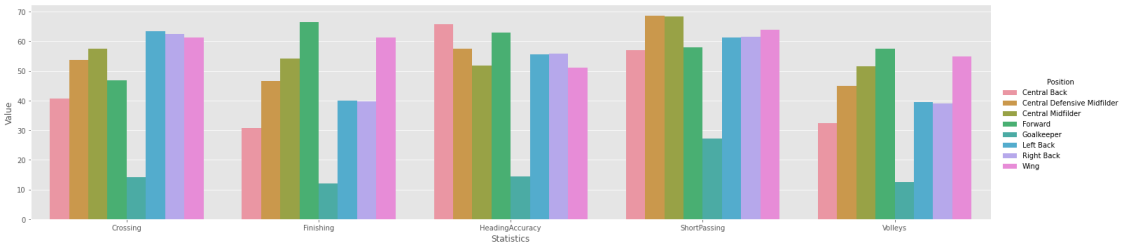


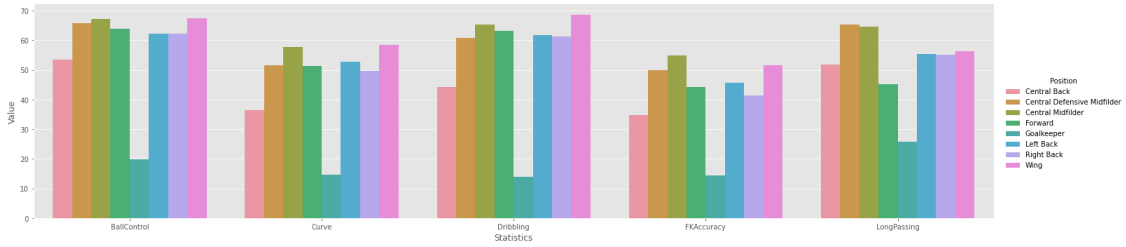
Suddividendo il grafico precedente nelle 8 posizioni si nota che: * i ruoli più difensivi sono Central Back, Right Back, e Left Back, ovvero l'insieme dei ruoli difensivi * seguiti poi dalle posizioni di centrocampo, ovvero Central Defensive Midfilder, Central Midfilder * infine, l'abilità difensiva decresce seguendo il seguente ordine: Wing, Forward e Goalkeeper

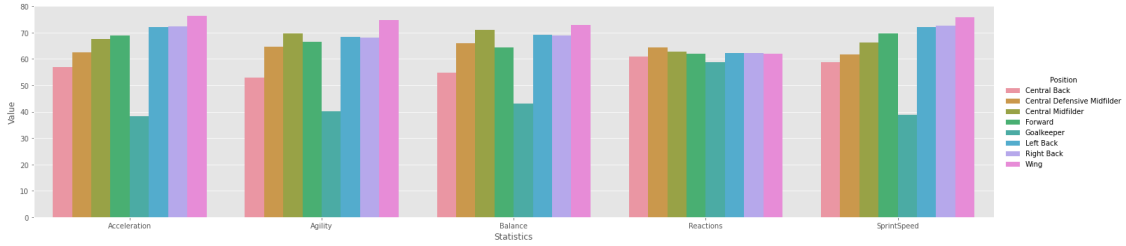
In seguito è stata eseguita un'analisi sui dati per ottenere e visualizzare i valori medi di tutte statistiche per ognuno dei 8 ruoli presi in considerazione.

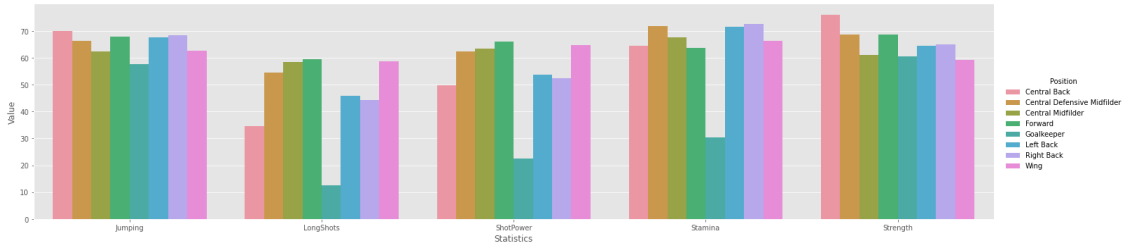
[37]:
```
technical_statistics = fifa.loc[:,'Position':"GKReflexes"]

technical_1 = technical_statistics.loc[:,'Crossing':'Volleys']
technical_2 = technical_statistics.loc[:,'Dribbling':'BallControl']
technical_3 = technical_statistics.loc[:,'Acceleration':'Balance']
technical_4 = technical_statistics.loc[:,'ShotPower':'LongShots']
technical_5 = technical_statistics.loc[:,'Aggression':'Penalties']
technical_6 = technical_statistics.loc[:,'Composure':'SlidingTackle']
technical_7 = technical_statistics.loc[:,'GKDiving':'GKReflexes']


technical_1['Position'] = technical_statistics['Position']
technical_2['Position'] = technical_statistics['Position']
technical_3['Position'] = technical_statistics['Position']
technical_4['Position'] = technical_statistics['Position']
technical_5['Position'] = technical_statistics['Position']
technical_6['Position'] = technical_statistics['Position']
technical_7['Position'] = technical_statistics['Position']


x1 = technical_1.
 ↪melt(id_vars='Position',var_name='Statistics',value_name='Value').
 ↪groupby(['Position','Statistics'], as_index = False).mean()
x2 = technical_2.
 ↪melt(id_vars='Position',var_name='Statistics',value_name='Value').
 ↪groupby(['Position','Statistics'], as_index = False).mean()
```

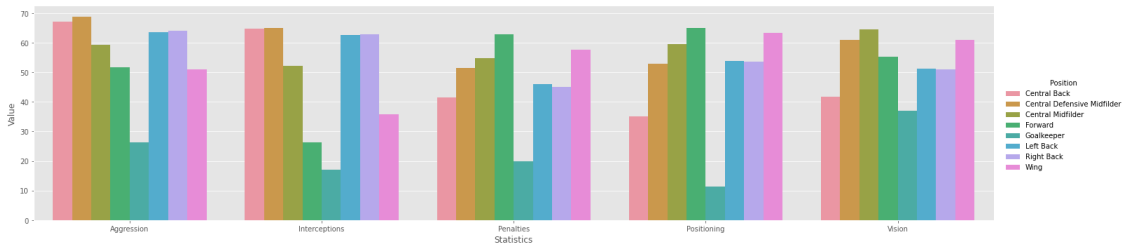
```
x3 = technical_3.
 ↪melt(id_vars='Position',var_name='Statistics',value_name='Value').
 ↪groupby(['Position','Statistics'], as_index = False).mean()
x4 = technical_4.
 ↪melt(id_vars='Position',var_name='Statistics',value_name='Value').
 ↪groupby(['Position','Statistics'], as_index = False).mean()
x5 = technical_5.
 ↪melt(id_vars='Position',var_name='Statistics',value_name='Value').
 ↪groupby(['Position','Statistics'], as_index = False).mean()
x6 = technical_6.
 ↪melt(id_vars='Position',var_name='Statistics',value_name='Value').
 ↪groupby(['Position','Statistics'], as_index = False).mean()
x7 = technical_7.
 ↪melt(id_vars='Position',var_name='Statistics',value_name='Value').
 ↪groupby(['Position','Statistics'], as_index = False).mean()


sns.catplot(x="Statistics", y="Value", hue="Position", kind="bar", data=x1,␣
 ↪height=5,aspect=4 );
sns.catplot(x="Statistics", y="Value", hue="Position", kind="bar", data=x2,␣
 ↪height=5,aspect=4 );
sns.catplot(x="Statistics", y="Value", hue="Position", kind="bar", data=x3,␣
 ↪height=5,aspect=4 );
sns.catplot(x="Statistics", y="Value", hue="Position", kind="bar", data=x4,␣
 ↪height=5,aspect=4 );
sns.catplot(x="Statistics", y="Value", hue="Position", kind="bar", data=x5,␣
 ↪height=5,aspect=4 );
sns.catplot(x="Statistics", y="Value", hue="Position", kind="bar", data=x6,␣
 ↪height=5,aspect=4 );
sns.catplot(x="Statistics", y="Value", hue="Position", kind="bar", data=x7,␣
 ↪height=5,aspect=4 );
```

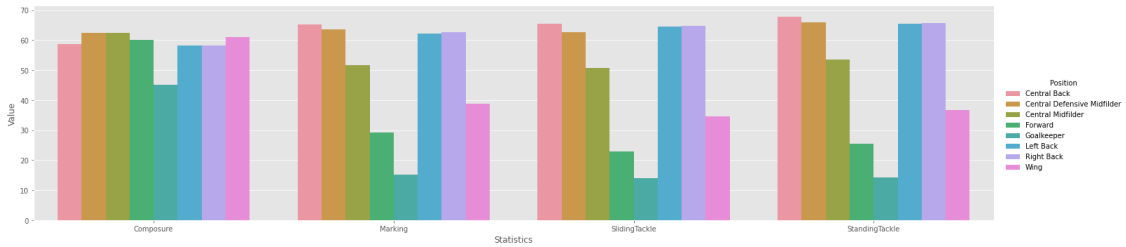


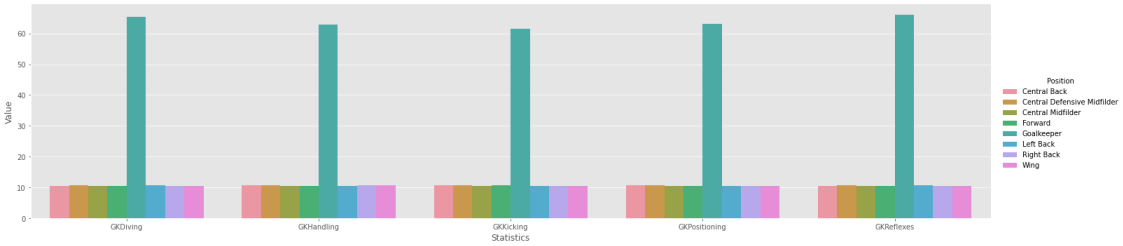

Value
Statistics
BallControl
Curve
Dribbling
FKAccuracy
LongPassing
Position
Central Back
Central Defensive Midfilder
Central Midfilder
Forward
Goalkeeper
Left Back
Right Back
Wing


Value
Statistics
Acceleration
Agility
Balance
Reactions
SprintSpeed
Position
Central Back
Central Defensive Midfilder
Central Midfilder
Forward
Goalkeeper
Left Back
Right Back
Wing


Value
Statistics
Jumping
LongShots
ShotPower
Stamina
Strength
Position
Central Back
Central Defensive Midfilder
Central Midfilder
Forward
Goalkeeper
Left Back
Right Back
Wing

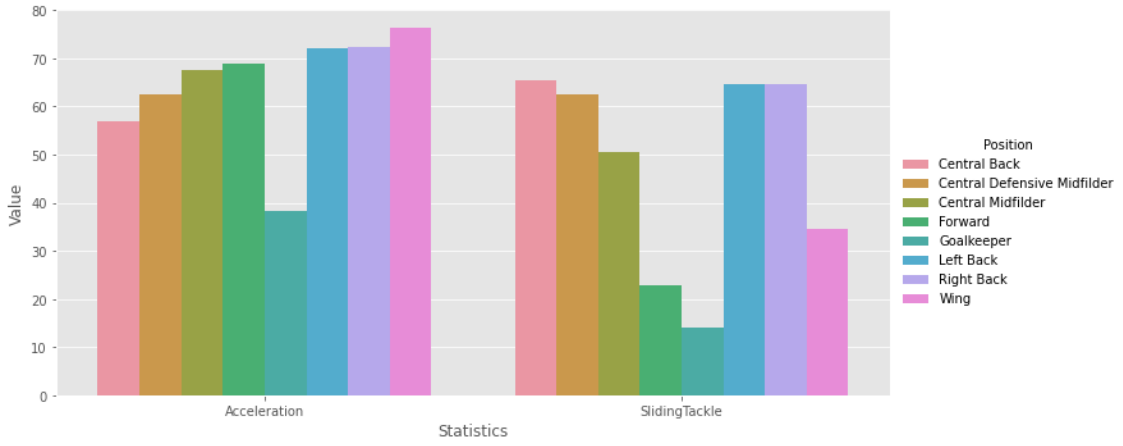

Value
Statistics
Aggression
Interceptions
Penalties
Positioning
Vision
Position
Central Back
Central Defensive Midfilder
Central Midfilder
Forward
Goalkeeper
Left Back
Right Back
Wing


Value
Statistics
Composure
Marking
SlidingTackle
StandingTackle
Position
Central Back
Central Defensive Midfilder
Central Midfilder
Forward
Goalkeeper
Left Back
Right Back
Wing



I grafici sovrastanti riportano il valore medio di ogni singola statistica, raggruppando per posizione di gioco, in particolare si può osservare che: * i calciatori che hanno una maggiore abilità nel tiro in porta (`Finishing`) sono gli attaccanti, seguiti dalle ali e dai centrocampisti centrali * come visto precedentemente, i 4 ruoli più difensivi (`Central Back`, `Left Back`, `Right back` e `Central Defensive Midfielder`) sono quelli più predisposti ad avere una media maggiore nelle statistiche di difesa come la scivolata, l'abilità nell'effettuare contrasti da in piedi, la marcatura el'abilità nell'intercettare il pallone * per quanto riguarda le abilità legate ai portieri, come il tuffo, la parata, il posizionamento in porta, ovviamente i portieri stessi hanno una media nettamente superiore agli altri

[38]:
```
technical_statistics = fifa.loc[:,['Position','Acceleration','SlidingTackle']]
df =technical_statistics.melt(id_vars='Position',var_name='Statistics',
                              value_name='Value').
 ↪groupby(['Position','Statistics'], as_index = False).mean()
sns.catplot(x="Statistics", y="Value", hue="Position", kind="bar", data=df,␣
 ↪height=5,aspect=2 );
```



Si può notare che i giocatori con una buona accelerazione e scarsa abilità nell'eseguire una scivolata sono gli attaccanti (`Forward`).

[39]:
```
fifa.head(5)
```

[39]:

	Overall	Potential	LS	...	GKPositioning	GKReflexes	Jersey Number
0	94	94	88	...	14.0	8.0	10.0
1	94	94	91	...	14.0	11.0	7.0
2	92	93	84	...	15.0	11.0	10.0
3	91	93	0	...	88.0	94.0	1.0
4	91	92	82	...	10.0	13.0	7.0

[5 rows x 75 columns]

6 Standardizzazione dei dati

Prima di procedere, dato che nel nostro dataset troviamo dei valori con scale diverse, si standardizzano i valori in modo tale da averne una serie in una distribuzione normale standard con media uguale a 0 e deviazione standard uguale a 1.

[40]:
```
X = fifa.drop(columns=["Position"])
y = fifa["Position"]

X_train, X_val, y_train, y_val = \
    train_test_split(X, y, test_size=1/4, random_state=42)

scaler = StandardScaler()

Xs_train = scaler.fit_transform(X_train)
Xs_val = scaler.fit_transform(X_val)
```

7 Scelta del modello

Iniziamo lo studio dei modelli di classificazione partendo dal più semplice: il Perceptron.

Appunto sulla GridSearch : nelle grid search sono mostrati solo gli iperparametri che hanno ottenuto i risultati migliori, per permettere un'esecuzione complessiva più rapida. Gli iperparametri 'commentati' sono stati testati in notebook separati.

7.1 Perceptron

A seguito di diversi testi ci siamo accorti che per questo modello la cross fold validation con 5 fold dava il risultato migliore quindi in questa dimostrazione verrà utilizzzato 5 come numero di fold.

[41]:
```
skf = StratifiedKFold(5, shuffle=True)

perceptron = Pipeline([
              ("perc", Perceptron(random_state=42))
])
```

[42]:
```
grid_perc = {
    'perc__penalty' : ['l1','l2',None],
    'perc__class_weight' : ['balanced', None],
    'perc__alpha' : np.logspace(-7, -1, 7)
}

gs_perc = GridSearchCV(perceptron, scoring='accuracy', param_grid = grid_perc,␣
 ↪cv=skf)
gs_perc.fit(Xs_train, y_train);
```

[43]:
```
gs_perc.best_params_
```

[43]:
```
{'perc__alpha': 1e-05, 'perc__class_weight': None, 'perc__penalty': 'l1'}
```

[44]:
```
pd.DataFrame(gs_perc.cv_results_).sort_values("rank_test_score").head(5)
```

[44]:

	mean_fit_time	std_fit_time	...	std_test_score	rank_test_score
15	1.161738	0.213466	...	0.025369	1
12	1.108043	0.089812	...	0.014699	2
10	0.286656	0.023051	...	0.013233	3
1	0.277853	0.023242	...	0.009779	4
7	0.271572	0.027225	...	0.018584	5

[5 rows x 16 columns]

[45]:
```
gs_perc.score(Xs_val,y_val)
```

[45]:
```
0.7333039453383293
```

[46]:
```
with open("model_perceptron.bin", "wb") as f:
    pickle.dump(gs_perc.best_estimator_, f)
```

[47]:
```
# files.download('model_perceptron.bin')
```

7.2 KNeighborsClassifier

[48]:
```
model_KN = Pipeline([
    ("kn", KNeighborsClassifier())
])

grid_KN = {
    'kn__n_neighbors' : [5,10,20,30,40],
    'kn__weights' : ['distance', 'uniform'],
    'kn__p' : [1,2]
}
```

```
gs_KN = GridSearchCV(model_KN, scoring='accuracy', param_grid = grid_KN, cv=skf)
gs_KN.fit(Xs_train, y_train);
```

[49]:
```
gs_KN.best_params_
```

[49]:
```
{'kn__n_neighbors': 30, 'kn__p': 1, 'kn__weights': 'distance'}
```

[50]:
```
pd.DataFrame(gs_KN.cv_results_).sort_values("rank_test_score").head(5)
```

[50]:

	mean_fit_time	std_fit_time	...	std_test_score	rank_test_score
12	0.117752	0.002465	...	0.010055	1
8	0.118047	0.001598	...	0.009170	2
13	0.117035	0.002807	...	0.012128	3
16	0.120491	0.002319	...	0.010421	4
9	0.118837	0.002271	...	0.008791	5

[5 rows x 16 columns]

[51]:
```
gs_KN.score(Xs_val,y_val)
```

[51]:
```
0.7674674895305268
```

[52]:
```
with open("model_KN.bin", "wb") as f:
    pickle.dump(gs_KN.best_estimator_, f)
```

[53]:
```
# files.download('model_KN.bin')
```

7.3 Regressione Logistica

[54]:
```
skfLR = StratifiedKFold(3, shuffle=True)
model_LR = Pipeline([
    ("logreg", LogisticRegression())
])
grid_LR = {
    "logreg__max_iter": [3000],
    "logreg__n_jobs": [-1],
    "logreg__solver": ["saga"],
    "logreg__penalty": ["l2", "l1"],# [elasticnet, None]
    "logreg__C": [ 1, 10], #[0.01, 0.1]
    "logreg__multi_class": ["ovr", "multinomial"]
}
gs_LR = GridSearchCV(model_LR, grid_LR, cv=skfLR)
gs_LR.fit(Xs_train, y_train);
```

[55]:
```
gs_LR.best_params_
```

[55]:
```
{'logreg__C': 10,
 'logreg__max_iter': 3000,
 'logreg__multi_class': 'multinomial',
 'logreg__n_jobs': -1,
 'logreg__penalty': 'l1',
 'logreg__solver': 'saga'}
```

[56]:
```
pd.DataFrame(gs_LR.cv_results_).sort_values("rank_test_score").head(5)
```

[56]:

	mean_fit_time	std_fit_time	...	std_test_score	rank_test_score
7	145.526828	11.780593	...	0.003949	1
3	89.654598	12.428570	...	0.003799	2
6	82.458862	8.124619	...	0.004014	3
2	23.125242	0.966411	...	0.002710	4
4	100.999913	14.221498	...	0.003470	5

[5 rows x 17 columns]

[57]:
```
gs_LR.score(Xs_val, y_val)
```

[57]:
```
0.7974432444346484
```

[58]:
```
with open("model_LR.bin", "wb") as f:
    pickle.dump(gs_LR.best_estimator_, f)
```

[59]:
```
# files.download('model_LR.bin')
```

7.4 Multi Layer Perceptron Classifier

[60]:
```
model_MLP = Pipeline([
    ("mlp", MLPClassifier())
])
grid_MLP = {
    "mlp__activation": ["relu"], #, "identity"], "logistic", "tanh"],
    "mlp__hidden_layer_sizes": [128], #[64, 128, 256, 1024],
    "mlp__batch_size": [100], #, 200],
    "mlp__early_stopping": [True, False],
    "mlp__max_iter": [1000],
    "mlp__alpha": [0.1], # [0.0001, 0.001, 1],
    "mlp__tol": [1], # [0.0001, 0.001, 0.01, 0.1],
}
skfMLP = StratifiedKFold(3, shuffle=True)
gs_MLP = GridSearchCV(model_MLP, grid_MLP, cv=skfMLP)
gs_MLP.fit(Xs_train, y_train);
```

[61]:
```
gs_MLP.best_params_
```

[61]:
```
{'mlp__activation': 'relu',
 'mlp__alpha': 0.1,
 'mlp__batch_size': 100,
 'mlp__early_stopping': False,
 'mlp__hidden_layer_sizes': 128,
 'mlp__max_iter': 1000,
 'mlp__tol': 1}
```

[62]:
```
pd.DataFrame(gs_MLP.cv_results_).sort_values("rank_test_score").head(5)
```

[62]:
```
   mean_fit_time  std_fit_time  ...  std_test_score  rank_test_score
1       1.544839      0.010256  ...        0.012534                1
0       1.490479      0.019846  ...        0.012966                2

[2 rows x 18 columns]
```

[63]:
```
gs_MLP.score(Xs_val, y_val)
```

[63]:
```
0.7981044743222394
```

[64]:
```
with open("model_MLP.bin", "wb") as f:
    pickle.dump(gs_MLP.best_estimator_, f)
```

[65]:
```
# files.download('model_MLP.bin')
```

7.5 XGBoostClassifier Classifier

[66]:
```
model_XGB = Pipeline([
    ("xgb", XGBClassifier())
])
grid_XGB = {
    "xgb__max_depth": [6], # [3, 5, 6, 9],
    "xgb__colsample_bytree": [1], # [0.1, 0.3, 0.9],
    "xgb__gamma": [0], # [0, 0.25, 0.5],
    "xgb__learning_rate": [0.2], # [0.001, 0.01, 0.1, 0.15, 0.2],
    "xgb__subsample": [1], #[0.8, 0.9, 1],
}

skfxgb = StratifiedKFold(3, shuffle=True)
gs_XGB = GridSearchCV(model_XGB, grid_XGB, cv=skfxgb)
gs_XGB.fit(Xs_train, y_train);
```

[67]:
```
gs_XGB.best_params_
```

[67]:
```
{'xgb__colsample_bytree': 1,
 'xgb__gamma': 0,
 'xgb__learning_rate': 0.2,
```

```
 'xgb__max_depth': 6,
 'xgb__subsample': 1}
```

[68]:
```
pd.DataFrame(gs_XGB.cv_results_).sort_values("rank_test_score").head(5)
```

[68]:
```
   mean_fit_time  std_fit_time  ...  std_test_score  rank_test_score
0       24.77505      0.279244  ...        0.006371                1

[1 rows x 16 columns]
```

[69]:
```
gs_XGB.score(Xs_val,y_val)
```

[69]:
```
0.793475865109103
```

[70]:
```
with open("model_XGB.bin", "wb") as f:
    pickle.dump(gs_XGB.best_estimator_, f)
```

[71]:
```
# files.download('model_XGB.bin')
```

7.6 Support Vector Machines

[72]:
```
model_SVC = Pipeline([
                ('svc', SVC(probability=True))
])

grid_SVC = {
    'svc__gamma': ['auto','scale'],
    'svc__C': [1,5],#[0.1,3,7]
    'svc__kernel':['poly','rbf'],#[linear, sigmoid]
    'svc__class_weight':['balanced', None]
}

gs_SVC = GridSearchCV(model_SVC,param_grid=grid_SVC,cv=skf)
gs_SVC.fit(Xs_train,y_train)
```

[72]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=True),
             error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('svc',
                                        SVC(C=1.0, break_ties=False,
                                            cache_size=200, class_weight=None,
                                            coef0=0.0,
                                            decision_function_shape='ovr',
                                            degree=3, gamma='scale',
                                            kernel='rbf', max_iter=-1,
                                            probability=True, random_state=None,
```

```
                                        shrinking=True, tol=0.001,
                                        verbose=False))],
                          verbose=False),
                iid='deprecated', n_jobs=None,
                param_grid={'svc__C': [1, 5],
                            'svc__class_weight': ['balanced', None],
                            'svc__gamma': ['auto', 'scale'],
                            'svc__kernel': ['poly', 'rbf']},
                pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                scoring=None, verbose=0)
```

[73]:
```
gs_SVC.best_params_
```

[73]:
```
{'svc__C': 5,
 'svc__class_weight': None,
 'svc__gamma': 'scale',
 'svc__kernel': 'rbf'}
```

[74]:
```
pd.DataFrame(gs_SVC.cv_results_).sort_values("rank_test_score").head(5)
```

[74]:

	mean_fit_time	std_fit_time	...	std_test_score	rank_test_score
15	29.839470	0.243767	...	0.006591	1
13	29.865755	0.175191	...	0.006485	2
5	31.693917	0.185167	...	0.009834	3
7	31.697186	0.206570	...	0.009837	4
9	30.852389	0.206212	...	0.006206	5

[5 rows x 17 columns]

[75]:
```
gs_SVC.score(Xs_val,y_val)
```

[75]:
```
0.8000881639850121
```

[76]:
```
with open("model_SVM.bin", "wb") as f:
    pickle.dump(gs_SVC.best_estimator_, f)
```

[77]:
```
# files.download('model_SVM.bin')
```

7.7 Classificatore Casuale - DummyClassifier

[78]:
```
random_model = DummyClassifier(strategy="uniform", random_state=42)
random_model.fit(Xs_train, y_train)

random_score = random_model.score(Xs_val, y_val)
```

[79]:
```
random_score
```

[79]:
```
0.1207846594666079
```

Si ottiene una precisione del 12%, visto che il problema di classificazione contiene 8 classi, e quindi 100/8 è uguale circa a 12.

8 Valutazione dei modelli

Osservando i diversi modelli ottenuti precedentemente, tramite una serie di metriche andiamo a identificare il modello migliore.

Si andranno a calcolare per ciascun modello:

- matrici di confusione
- recall, f1-score e precision
- intervallo di confidenza con confidenza fissata al 95%

Andiamo prima a caricare i modelli...

[80]:
```
with open("model_perceptron.bin", "rb") as f:
    model_perc = pickle.load(f)
```

[81]:
```
with open("model_KN.bin", "rb") as f:
    model_KN = pickle.load(f)
```

[82]:
```
with open("model_LR.bin", "rb") as f:
    model_LR=pickle.load(f)
```

[83]:
```
with open("model_MLP.bin", "rb") as f:
    model_MLP = pickle.load(f)
```

[84]:
```
with open("model_XGB.bin", "rb") as f:
    model_XGB = pickle.load(f)
```

[85]:
```
with open("model_SVM.bin", "rb") as f:
    model_SVM = pickle.load(f)
```

Matrice di Confusione del Perceptron

[86]:
```
predictionP = model_perc.predict(Xs_val)
print('Accuracy Score: ', accuracy_score(y_val, predictionP))
```

```
Accuracy Score:  0.7333039453383293
```

[87]:
```
cmP = confusion_matrix(y_val, predictionP)
pd.DataFrame(cmP, index=model_perc.classes_, columns=model_perc.classes_)
```

[87]:

	Central Back	...	Wing
Central Back	697	...	0
Central Defensive Midfilder	76	...	4

```
Central Midfilder                      22  ...    75
Forward                                29  ...   176
Goalkeeper                              0  ...     0
Left Back                              23  ...    16
Right Back                             24  ...     5
Wing                                    1  ...   622

[8 rows x 8 columns]
```

Matrice di Confusione di KNeighborsClassifier

[88]:
```
predictionKN = model_KN.predict(Xs_val)
print('Accuracy Score: ', accuracy_score(y_val, predictionKN))
```

```
Accuracy Score:  0.7674674895305268
```

[89]:
```
cmKN = confusion_matrix(y_val, predictionKN)
pd.DataFrame(cmKN, index=model_KN.classes_, columns=model_KN.classes_)
```

[89]:
```
                           Central Back  ...  Wing
Central Back                        684  ...     0
Central Defensive Midfilder          30  ...     5
Central Midfilder                     3  ...   118
Forward                               0  ...    92
Goalkeeper                            0  ...     0
Left Back                            21  ...     3
Right Back                           33  ...     0
Wing                                  1  ...   579

[8 rows x 8 columns]
```

Matrice di Confusione della Regressione Logistica

[90]:
```
predictionLR = model_LR.predict(Xs_val)
print('Accuracy Score: ', accuracy_score(y_val, predictionLR))
```

```
Accuracy Score:  0.7974432444346484
```

[91]:
```
cmLR = confusion_matrix(y_val, predictionLR)
pd.DataFrame(cmLR, index=model_LR.classes_, columns=model_LR.classes_)
```

[91]:
```
                           Central Back  ...  Wing
Central Back                        694  ...     0
Central Defensive Midfilder          27  ...     3
Central Midfilder                     2  ...    83
Forward                               0  ...    81
Goalkeeper                            0  ...     0
Left Back                            10  ...     7
```

```
Right Back                          23  ...     2
Wing                                 0  ...   601

[8 rows x 8 columns]
```

Matrice di Confusione di MLPClassifier

[92]:
```
predictionMLP = model_MLP.predict(Xs_val)
print('Accuracy Score: ', accuracy_score(y_val, predictionMLP))
```

```
Accuracy Score:  0.7981044743222394
```

[93]:
```
cmMLP = confusion_matrix(y_val, predictionMLP)
pd.DataFrame(cmMLP, index=model_MLP.classes_, columns=model_MLP.classes_)
```

[93]:
```
                           Central Back  ...  Wing
Central Back                        695  ...     0
Central Defensive Midfilder          28  ...     2
Central Midfilder                     4  ...    65
Forward                               0  ...    47
Goalkeeper                            0  ...     0
Left Back                            13  ...     7
Right Back                           28  ...     2
Wing                                  1  ...   558

[8 rows x 8 columns]
```

Matrice di Confusione di XGBoostClassifier

[94]:
```
predictionXGB = model_XGB.predict(Xs_val)
print('Accuracy Score: ', accuracy_score(y_val, predictionXGB))
```

```
Accuracy Score:  0.793475865109103
```

[95]:
```
cmXGB = confusion_matrix(y_val, predictionXGB)
pd.DataFrame(cmXGB, index=model_XGB.classes_, columns=model_XGB.classes_)
```

[95]:
```
                           Central Back  ...  Wing
Central Back                        701  ...     0
Central Defensive Midfilder          29  ...     1
Central Midfilder                     2  ...   103
Forward                               0  ...    76
Goalkeeper                            0  ...     0
Left Back                            21  ...     7
Right Back                           34  ...     5
Wing                                  1  ...   605

[8 rows x 8 columns]
```

Matrice di Confusione di Support Vector Machines

[96]:
```
predictionSVM = model_SVM.predict(Xs_val)
print('Accuracy Score: ', accuracy_score(y_val, predictionSVM))
```

```
Accuracy Score:  0.8000881639850121
```

[97]:
```
cmSVM = confusion_matrix(y_val, predictionSVM)
pd.DataFrame(cmSVM, index=model_SVM.classes_, columns=model_SVM.classes_)
```

[97]:

	Central Back	...	Wing
Central Back	699	...	0
Central Defensive Midfilder	18	...	3
Central Midfilder	3	...	86
Forward	0	...	75
Goalkeeper	0	...	0
Left Back	14	...	3
Right Back	26	...	3
Wing	0	...	598

[8 rows x 8 columns]

[98]:
```
reportP = classification_report(y_val, predictionP, output_dict=True)
reportP = pd.DataFrame(reportP).transpose()

reportKN = classification_report(y_val, predictionKN, output_dict=True)
reportKN = pd.DataFrame(reportKN).transpose()

reportLR = classification_report(y_val, predictionLR, output_dict=True)
reportLR = pd.DataFrame(reportLR).transpose()

reportMLP = classification_report(y_val, predictionMLP, output_dict=True)
reportMLP = pd.DataFrame(reportMLP).transpose()

reportXGB = classification_report(y_val, predictionXGB, output_dict=True)
reportXGB = pd.DataFrame(reportXGB).transpose()

reportSVM = classification_report(y_val, predictionSVM, output_dict=True)
reportSVM = pd.DataFrame(reportSVM).transpose()
```

[99]:
```
pd.DataFrame([reportP.loc['weighted avg',:],reportKN.loc['weighted avg',:],
              reportLR.loc['weighted avg',:], reportMLP.loc['weighted avg',:],
              reportXGB.loc['weighted avg',:],reportSVM.loc['weighted avg',:]],
              index=['Perceptron','KNearestNeighbord','LinearRegression',
                     'MultyLayerPerceptron','XGBoost','SupportVectorMachine'])
```

[99]:

	precision	recall	f1-score	support
Perceptron	0.738770	0.733304	0.729303	4537.0
KNearestNeighbord	0.767980	0.767467	0.765018	4537.0
LinearRegression	0.798201	0.797443	0.795732	4537.0
MultyLayerPerceptron	0.801182	0.798104	0.794391	4537.0
XGBoost	0.792605	0.793476	0.791226	4537.0
SupportVectorMachine	0.801373	0.800088	0.798260	4537.0

[100]:
```
def confidence(acc, N, Z):
    den = (2*(N+Z**2))
    var = (Z*np.sqrt(Z**2+4*N*acc-4*N*acc**2)) / den
    a = (2*N*acc+Z**2) / den
    inf = a - var
    sup = a + var
    return (inf, sup)

def calculate_accuracy(conf_matrix):
    return np.diag(conf_matrix).sum() / conf_matrix.sum().sum()
```

[101]:
```
#con confidenza del 95% si ha Z=1.96
pd.DataFrame([confidence(calculate_accuracy(cmP), len(Xs_val), 1.96),
              confidence(calculate_accuracy(cmKN), len(Xs_val), 1.96),
              confidence(calculate_accuracy(cmMLP), len(Xs_val), 1.96),
              confidence(calculate_accuracy(cmLR), len(Xs_val), 1.96),
              confidence(calculate_accuracy(cmXGB), len(Xs_val), 1.96),
              confidence(calculate_accuracy(cmSVM), len(Xs_val), 1.96)],
              index=["Perceptron", "KNeighborsClassifier",
                     "Multi Layer Perceptron",
                     "Logistic Regression",
                     "XGBoostClassifier",
                     "Support Vector Machines"
                     ], columns=["inf", "sup"])
```

[101]:

	inf	sup
Perceptron	0.720242	0.745971
KNeighborsClassifier	0.754952	0.779531
Multi Layer Perceptron	0.786174	0.809531
Logistic Regression	0.785499	0.808884
XGBoostClassifier	0.781451	0.805005
Support Vector Machines	0.788199	0.811470

8.1 Confronto tra modelli

- In questa sezione verrà valutato se l'accuratezza a_1 misurata su un modello sia significativamente migliore della a_2 misurata sull'altro
- Se i numeri di osservazioni N_1 e N_2 nei validation set utilizzati è abbastanza grande (più di 30), l'accuratezza è approssimabile con una distribuzione normale

  - spesso il validation set è lo stesso, per cui $N_1 = N_2$
- La varianza della differenza di accuratezza si stima con

$$\hat{\sigma}^2 = \frac{a_1(1-a_1)}{N_1} + \frac{a_2(1-a_2)}{N_2}$$

- Gli estremi dell'intervallo di confidenza della differenza sono dati da

$$|a_1 - a_2| \pm Z\hat{\sigma}$$

- Se l'intervallo ottenuto non include lo zero (l'estremo inferiore è positivo), abbiamo la certezza al 95% (o altro livello di confidenza) che il modello con accuratezza stimata maggiore sia effettivamente migliore

(testo preso dall'esercitazione sulla Classificazione)

[102]:
```
def diff_interval(a1, a2, N1, N2, Z):
    d = abs(a1 - a2)
    sd = np.sqrt(a1 * (1-a1) / N1 + a2 * (1-a2) / N2)
    return d - Z * sd, d + Z * sd
```

[103]:
```
# confidenza fissata al 95%
def model_diff_interval(m1, m2, X, y, level=0.95):
    a1 = m1.score(X, y)
    a2 = m2.score(X, y)
    N = len(X)
    Z = norm.ppf((1 + level) / 2)
    return diff_interval(a1, a2, N, N, Z)
```

8.1.1 Intervalli che contengono lo zero

[104]:
```
model_diff_interval(model_XGB, model_SVM, Xs_val, y_val)
```

[104]:
```
(-0.009945967772169415, 0.023170565523987805)
```

[105]:
```
model_diff_interval(model_MLP, model_SVM, Xs_val, y_val)
```

[105]:
```
(-0.014504420503847787, 0.01847179982939326)
```

[106]:
```
model_diff_interval(model_LR, model_SVM, Xs_val, y_val)
```

[106]:
```
(-0.013853298551874657, 0.019143137652602102)
```

8.1.2 Intervalli che non contengono lo zero

[107]:
```
model_diff_interval(random_model, model_SVM, Xs_val, y_val)
```

[107]:
```
(0.6642921086494914, 0.6943149003873171)
```

[108]:
```
model_diff_interval(model_perc, model_SVM, Xs_val, y_val)
```

[108]:
```
(0.049434457569158, 0.08413397972420765)
```

[109]:
```
model_diff_interval(model_KN, model_SVM, Xs_val, y_val)
```

[109]:
```
(0.015693511956825255, 0.04954783695214546)
```

8.1.3 Modelli migliori

Da questi confronti emergono i modelli generati dai seguenti algoritmi: * **Logistic regression** * **Multi Layer Perceptron Classifier** * **XGBoostClassifier** * **Support Vector Machines**

Mentre i modelli che hanno un'accuratezza effettivamente minore sono quelli creati a partire dai rimanenti algoritmi: * Perceptron * KNeighbors Classifier * e ovviamente il Dummy Classifier

9 Bilanciamento delle classi

Testiamo ora un modello bilanciando le classi. Per farlo utiliziamo SMOTE classe che ci permette l'oversampling delle classi, ovvero di aggiungere dati alle classi che hanno un numero inferiore di istanze rispetto alla classe col numero maggiore di dati.

[110]:
```
X_resampled, y_resampled = SMOTE(random_state=42).fit_resample(X, y)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
```

```
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

Si può vedere che ora tutte le classi hanno lo stesso numero di istanze.

[111]:
```
pd.Series(y_resampled).value_counts()
```

[111]:
```
Central Back                  3422
Central Defensive Midfilder   3422
Right Back                    3422
Wing                          3422
Goalkeeper                    3422
Left Back                     3422
Central Midfilder             3422
Forward                       3422
dtype: int64
```

Quindi si procede andando a sepoarare il dataset in subset di training e validation e viene effettuata fin da subito la standardizzazione dei dati.

[112]:
```
X_train_resampled, X_value_resampled, y_train_resampled, y_value_resampled =␣
 ↪train_test_split(X_resampled, y_resampled, random_state=42, test_size=1/4)

scaler_b = StandardScaler()

#standardizzazione dei dati
Xs_train_resampled = scaler_b.fit_transform(X_train_resampled)
Xs_val_resampled = scaler_b.fit_transform(X_value_resampled)
```

Si è deciso di fare una prima prova con l'algoritmo di regressione logistica, poichè nella gridsearch non è stato usato alcun parametro per effettuare un bilanciamento delle classi.

RLogistic Regression Balanced

[113]:
```
model_LR_balanced =␣
 ↪LogisticRegression(C=1,max_iter=3000,multi_class='multinomial',penalty='l1',solver='saga')
model_LR_balanced.fit(Xs_train_resampled, y_train_resampled)
model_LR_balanced.score(Xs_val_resampled, y_value_resampled)
```

[113]:
```
0.8265634132086499
```

[114]:
```
predictionLR_balanced = model_LR_balanced.predict(Xs_val_resampled)
cmLR_balanced = confusion_matrix(y_value_resampled, predictionLR_balanced)
pd.DataFrame(cmLR_balanced, index=model_LR_balanced.classes_,␣
 ↪columns=model_LR_balanced.classes_)
```

[114]:
```
                            Central Back  ...  Wing
Central Back                         757  ...     1
Central Defensive Midfilder           19  ...     3
Central Midfilder                      2  ...    92
Forward                                0  ...    62
Goalkeeper                             0  ...     0
Left Back                              6  ...    11
Right Back                            15  ...     8
Wing                                   0  ...   521

[8 rows x 8 columns]
```

[115]:
```
reportLR_balanced = classification_report(y_value_resampled,␣
 ↪predictionLR_balanced, output_dict=True)
pd.DataFrame(reportLR_balanced).transpose()
```

[115]:

	precision	recall	f1-score	support
Central Back	0.947434	0.879210	0.912048	861.000000
Central Defensive Midfilder	0.735462	0.757042	0.746096	852.000000
Central Midfilder	0.656286	0.674171	0.665108	844.000000
Forward	0.811902	0.904142	0.855543	845.000000
Goalkeeper	1.000000	1.000000	1.000000	868.000000
Left Back	0.895417	0.858108	0.876366	888.000000
Right Back	0.817603	0.911348	0.861934	846.000000
Wing	0.746418	0.620238	0.677503	840.000000
accuracy	0.826563	0.826563	0.826563	0.826563
macro avg	0.826315	0.825532	0.824325	6844.000000
weighted avg	0.827605	0.826563	0.825503	6844.000000

Quindi svolgendo a priori un bilanciamento delle classi di tipo oversampling con l'algoritmo Logistic Regression, e sfruttando i parametri migliori ottenuti dal modello precedente, osserviamo un miglioramento del modello di circa 3% nell'accuratezza.

Quindi si procede col testare anche tutti gli altri modelli utilizzando il bilanciamento delle classi, per vedere se in generale si ottengono risultati .

Perceptron Balanced

[116]:
```
model_perc_balanced = Perceptron(penalty='l1',alpha=1e-7)
model_perc_balanced.fit(Xs_train_resampled, y_train_resampled)
model_perc_balanced.score(Xs_val_resampled, y_value_resampled)
```

[116]:
```
0.7599357101110462
```

KNearestNeighbors Classifier Balanced

[117]:
```
model_KN_balanced = KNeighborsClassifier(n_neighbors = 20, p = 1, weights =␣
 ↪'distance')
```

```
model_KN_balanced.fit(Xs_train_resampled, y_train_resampled)
model_KN_balanced.score(Xs_val_resampled, y_value_resampled)
```

[117]: 0.8253945061367621

Multi Layer Perceptron Classifier Balanced

[118]:
```
model_MLP_balanced =␣
↪MLPClassifier(activation='relu',batch_size=100,early_stopping=True,hidden_layer_sizes=128,max
↪1)
model_MLP_balanced.fit(Xs_train_resampled, y_train_resampled)
model_MLP_balanced.score(Xs_val_resampled, y_value_resampled)
```

[118]: 0.8267095265926359

XGBoostClassifier Balanced

[119]:
```
model_XGB_balanced = XGBClassifier(colsample_bytree=1,gamma=0,learning_rate=0.
↪2,max_depth=6,subsample=1)
model_XGB_balanced.fit(Xs_train_resampled, y_train_resampled)
model_XGB_balanced.score(Xs_val_resampled, y_value_resampled)
```

[119]: 0.8464348334307422

SVM Balanced

[120]:
```
model_SVM_balanced = SVC(C=5,gamma = 'auto',kernel='rbf')
model_SVM_balanced.fit(Xs_train_resampled, y_train_resampled)
model_SVM_balanced.score(Xs_val_resampled, y_value_resampled)
```

[120]: 0.8404441846873174

Miglioramenti ottenuti

Dai primi risultati ottenuti si notano miglioramenti sostanziali nelle affidabilità di tutti algoritmi utilizzati per questo problema di classificazione:

- `Perceptron`: **+3%**
- `KNearestNeighbors`: **+6%**
- `MultiLayerPerceptronClassifier`: **+3%**
- `XGBoostClassifier`: **+5%**
- `SupportVectorMachines`: **+4%**

Matrici di Confusione degli algoritmi bilanciati

[121]:
```
predictionP_balanced = model_perc_balanced.predict(Xs_val_resampled)
cmP_balanced = confusion_matrix(y_value_resampled, predictionP_balanced)
```

[122]:
```
predictionKN_balanced = model_KN_balanced.predict(Xs_val_resampled)
cmKN_balanced = confusion_matrix(y_value_resampled, predictionKN_balanced)
```

[123]:
```
predictionMLP_balanced = model_MLP_balanced.predict(Xs_val_resampled)
cmMLP_balanced = confusion_matrix(y_value_resampled, predictionMLP_balanced)
```

[124]:
```
predictionXGB_balanced = model_XGB_balanced.predict(Xs_val_resampled)
cmXGB_balanced = confusion_matrix(y_value_resampled, predictionXGB_balanced)
```

[125]:
```
predictionSVM_balanced = model_SVM_balanced.predict(Xs_val_resampled)
cmSVM_balanced = confusion_matrix(y_value_resampled, predictionSVM_balanced)
```

Intervalli di Confidenza

[126]:
```
#con confidenza del 95% si ha Z=1.96
pd.DataFrame([confidence(calculate_accuracy(cmP_balanced),␣
 ↪len(Xs_val_resampled), 1.96),
              confidence(calculate_accuracy(cmKN_balanced),␣
 ↪len(Xs_val_resampled), 1.96),
              confidence(calculate_accuracy(cmMLP_balanced),␣
 ↪len(Xs_val_resampled), 1.96),
              confidence(calculate_accuracy(cmLR_balanced),␣
 ↪len(Xs_val_resampled), 1.96),
              confidence(calculate_accuracy(cmXGB_balanced),␣
 ↪len(Xs_val_resampled), 1.96),
              confidence(calculate_accuracy(cmSVM_balanced),␣
 ↪len(Xs_val_resampled), 1.96)],
              index=["Perceptron Balanced", "KNeighborsClassifier Balanced",
                     "Multi Layer Perceptron Balanced",
                     "Logistic Regression Balanced",
                     "XGBoostClassifier Balanced",
                     "Support Vector Machines Balanced"
                     ], columns=["inf", "sup"])
```

[126]:

	inf	sup
Perceptron Balanced	0.749672	0.769907
KNeighborsClassifier Balanced	0.816218	0.834205
Multi Layer Perceptron Balanced	0.817560	0.835493
Logistic Regression Balanced	0.817411	0.835350
XGBoostClassifier Balanced	0.837699	0.854782
Support Vector Machines Balanced	0.831578	0.848929

Confronto dei modelli bilanciati

Intervalli che contengono lo zero

[127]:
```
model_diff_interval(model_SVM_balanced, model_XGB_balanced, Xs_val_resampled,␣
↪y_value_resampled)
```

[127]:
```
(-0.006184138810676195, 0.018165436297525813)
```

Intervalli che non contengono lo zero

[128]:
```
model_diff_interval(model_SVM_balanced, model_KN_balanced, Xs_val_resampled,␣
↪y_value_resampled)
```

[128]:
```
(0.002553297017793667, 0.027546060083316892)
```

[129]:
```
model_diff_interval(model_SVM_balanced, model_perc_balanced, Xs_val_resampled,␣
↪y_value_resampled)
```

[129]:
```
(0.06717935876074267, 0.09383759039179972)
```

[130]:
```
model_diff_interval(model_MLP_balanced, model_XGB_balanced, Xs_val_resampled,␣
↪y_value_resampled)
```

[130]:
```
(0.00734109871391656, 0.03210951496229616)
```

[131]:
```
model_diff_interval(model_LR_balanced, model_XGB_balanced, Xs_val_resampled,␣
↪y_value_resampled)
```

[131]:
```
(0.007485049206627447, 0.03225779123755719)
```

Modelli migliori Da questi confronti, utilizzando il bilanciamento delle classi, emergono i modelli generati dai seguenti algoritmi: * **XGBoostClassifier** * **Support Vector Machines**

Mentre i modelli che hanno un'accuratezza effettivamente minore sono quelli creati a partire dai rimanenti algoritmi: * Perceptron * KNeighbors Classifier * Logistic regression * Multi Layer Perceptron Classifier

Quindi bilanciando a priori le istanze contenute in ogni classe altri due algoritmi (Logistic regression e Multi Layer Perceptron Classifier) perdono di efficacia rispetto a XGBoostClassifier e Support Vector Machines

10 Reti Neurali

Dopo aver ottenuto diversi buoni risultati con algoritmi diversi, proviamo l'approccio con Reti Neurali.

10.1 Librerie necessarie

[132]:
```
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from sklearn.preprocessing import OneHotEncoder
from keras.regularizers import l2,l1
```

```
Using TensorFlow backend.
```

10.2 Test

Eseguiamo una copia del dataset principale fifa ed eseguiamo le operazioni su di esso.

[133]:
```
fifa_copy = fifa.copy()
```

[134]:
```
#insieme di colonne che contengono un valore numerico non categorizzabile
numeric_cols = ['Overall', 'Potential', 'LS', 'ST', 'RS', 'LW', 'LF', 'CF',␣
 ↪'RF', 'RW',
       'LAM', 'CAM', 'RAM', 'LM', 'LCM', 'CM', 'RCM', 'RM', 'LWB', 'LDM',
       'CDM', 'RDM', 'RWB', 'LB', 'LCB', 'CB', 'RCB', 'RB', 'Value', 'Wage',
       'Age', 'WROffensive', 'WRDefensive', 'International Reputation',
       'Weak Foot', 'Skill Moves', 'Height', 'Weight'
       , 'Crossing', 'Finishing', 'HeadingAccuracy', 'ShortPassing',
       'Volleys', 'Dribbling', 'Curve', 'FKAccuracy', 'LongPassing',
       'BallControl', 'Acceleration', 'SprintSpeed', 'Agility', 'Reactions',
       'Balance', 'ShotPower', 'Jumping', 'Stamina', 'Strength', 'LongShots',
       'Aggression', 'Interceptions', 'Positioning', 'Vision', 'Penalties',
       'Composure', 'Marking', 'StandingTackle', 'SlidingTackle', 'GKDiving',
       'GKHandling', 'GKKicking', 'GKPositioning', 'GKReflexes',
       'Jersey Number']

#insieme di colonne che contengono un valore numerico categorizzabile
categorical_cols = ['Prefers Right Foot']

#colonna sulla quale vogliamo fare le rpedizioni
classes_col = 'Position'
```

[135]:
```
#funzione di utility che permette la categorizzazione delle colonne (che ne␣
 ↪necessitano) e il bilanciamento delle classi
#ritornando in output il train set e il validation set

def crete_train_val_sets(dataset, classes_col,␣
 ↪categorical_cols,numeric_cols,balanced = False):

  #essendo le nostre classi identificate da delle stringhe dobbiamo convertirle␣
 ↪in numeri interi per poterle categorizzare
```

```
 setattr(dataset,classes_col, pd.Categorical(getattr(dataset,classes_col) ))
 dataset[classes_col] = getattr(dataset,classes_col).cat.codes
 #divido il dataset in trin e validation e successivamente in label e dati
 data_train, data_val = train_test_split(dataset, test_size=1/4,␣
↪random_state=42)
 y_train = data_train[classes_col]
 y_val = data_val[classes_col]
 X_train = data_train[numeric_cols].copy()
 X_val = data_val[numeric_cols].copy()


 #categoriziamo tutte le colonne passate come parametro
 ohe = OneHotEncoder(categories="auto", sparse=False)

 X_train_cat = ohe.fit_transform(data_train[categorical_cols])
 X_val_cat = ohe.transform(data_val[categorical_cols])

 X_train = np.hstack([X_train.values, X_train_cat])
 X_val = np.hstack([X_val.values, X_val_cat])

 #bilanciamento dei dati dopo la categorizzazione
 if balanced:
   X =  np.vstack((X_train, X_val))
   y =  y_train.append(y_val)
   X,y = SMOTE(random_state=42).fit_resample(X,y)
   X_train,X_val,y_train,y_val = train_test_split(X,y, test_size=1/4,␣
↪random_state=42)


 #categoriziamo i label
 y_train = to_categorical(y_train)
 y_val = to_categorical(y_val)

 #standardiziamo i dati
 scaler_X = StandardScaler()
 X_train = scaler_X.fit_transform(X_train)
 X_val = scaler_X.transform(X_val)




 return X_train, X_val, y_train, y_val
```

Come primo approccio tentiamo un modello semplice composto da un solo di nodo input e un nodo output

[136]:
```
X_train,X_val,y_train,y_val = crete_train_val_sets(fifa_copy, classes_col,␣
↪categorical_cols, numeric_cols)
```

[137]:
```
model = Sequential([
    Dense(128, activation="relu", input_dim=X_train.shape[1]),
    Dense(8, activation="softmax")
])
```

[138]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               9728
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 1032
=================================================================
Total params: 10,760
Trainable params: 10,760
Non-trainable params: 0
_________________________________________________________________
```

[139]:
```
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",#un loss function che combina la loss cross␣
↪entropy con la softmax
    metrics=["accuracy"]
)
```

[140]:
```
fit_history = model.fit(X_train, y_train,validation_data=(X_val,y_val),␣
↪batch_size=100, epochs=20)
```

```
Train on 13610 samples, validate on 4537 samples
Epoch 1/20
13610/13610 [==============================] - 1s 45us/step - loss: 0.8871 -
accuracy: 0.6856 - val_loss: 0.6446 - val_accuracy: 0.7602
Epoch 2/20
13610/13610 [==============================] - 0s 18us/step - loss: 0.6217 -
accuracy: 0.7669 - val_loss: 0.5857 - val_accuracy: 0.7820
Epoch 3/20
13610/13610 [==============================] - 0s 18us/step - loss: 0.5831 -
accuracy: 0.7822 - val_loss: 0.5747 - val_accuracy: 0.7888
Epoch 4/20
13610/13610 [==============================] - 0s 19us/step - loss: 0.5666 -
accuracy: 0.7858 - val_loss: 0.5605 - val_accuracy: 0.7897
Epoch 5/20
```

```
13610/13610 [==============================] - 0s 20us/step - loss: 0.5538 -
accuracy: 0.7915 - val_loss: 0.5610 - val_accuracy: 0.7913
Epoch 6/20
13610/13610 [==============================] - 0s 18us/step - loss: 0.5455 -
accuracy: 0.7940 - val_loss: 0.5562 - val_accuracy: 0.7979
Epoch 7/20
13610/13610 [==============================] - 0s 19us/step - loss: 0.5406 -
accuracy: 0.7972 - val_loss: 0.5541 - val_accuracy: 0.7972
Epoch 8/20
13610/13610 [==============================] - 0s 18us/step - loss: 0.5365 -
accuracy: 0.7979 - val_loss: 0.5521 - val_accuracy: 0.7966
Epoch 9/20
13610/13610 [==============================] - 0s 20us/step - loss: 0.5292 -
accuracy: 0.8003 - val_loss: 0.5501 - val_accuracy: 0.7950
Epoch 10/20
13610/13610 [==============================] - 0s 19us/step - loss: 0.5261 -
accuracy: 0.8039 - val_loss: 0.5460 - val_accuracy: 0.8007
Epoch 11/20
13610/13610 [==============================] - 0s 19us/step - loss: 0.5230 -
accuracy: 0.8040 - val_loss: 0.5447 - val_accuracy: 0.7990
Epoch 12/20
13610/13610 [==============================] - 0s 18us/step - loss: 0.5192 -
accuracy: 0.8043 - val_loss: 0.5481 - val_accuracy: 0.7939
Epoch 13/20
13610/13610 [==============================] - 0s 19us/step - loss: 0.5143 -
accuracy: 0.8068 - val_loss: 0.5459 - val_accuracy: 0.7928
Epoch 14/20
13610/13610 [==============================] - 0s 20us/step - loss: 0.5109 -
accuracy: 0.8073 - val_loss: 0.5471 - val_accuracy: 0.7952
Epoch 15/20
13610/13610 [==============================] - 0s 20us/step - loss: 0.5090 -
accuracy: 0.8074 - val_loss: 0.5497 - val_accuracy: 0.7970
Epoch 16/20
13610/13610 [==============================] - 0s 19us/step - loss: 0.5056 -
accuracy: 0.8093 - val_loss: 0.5446 - val_accuracy: 0.7972
Epoch 17/20
13610/13610 [==============================] - 0s 20us/step - loss: 0.5023 -
accuracy: 0.8115 - val_loss: 0.5466 - val_accuracy: 0.7924
Epoch 18/20
13610/13610 [==============================] - 0s 18us/step - loss: 0.5003 -
accuracy: 0.8127 - val_loss: 0.5454 - val_accuracy: 0.7972
Epoch 19/20
13610/13610 [==============================] - 0s 18us/step - loss: 0.4953 -
accuracy: 0.8141 - val_loss: 0.5494 - val_accuracy: 0.7935
Epoch 20/20
13610/13610 [==============================] - 0s 19us/step - loss: 0.4946 -
accuracy: 0.8148 - val_loss: 0.5461 - val_accuracy: 0.7917
```

Senza bilanciamento otteniamo un buon risultato per quanto riguarda l'accuratezza sia sul train set sia sul validation, mentre la loss rimane comunque abbastanza alta.

Proviamo quindi a bilanciare i nostri dati e riutilizzare lo stesso modello.

[141]:
```
X_train_bal,X_val_bal,y_train_bal,y_val_bal = crete_train_val_sets(fifa_copy,␣
↪classes_col, categorical_cols, numeric_cols,balanced=True)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

[142]:
```
model_balanced = Sequential([
    Dense(128, activation="relu", input_dim=X_train_bal.shape[1]),
    Dense(8, activation="softmax")
])
```

[143]:
```
model_balanced.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
dense_3 (Dense)              (None, 128)               9728
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 1032
=================================================================
Total params: 10,760
Trainable params: 10,760
Non-trainable params: 0
_________________________________________________________________
```

[144]:
```
model_balanced.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

[145]:
```
fit_history = model_balanced.fit(X_train_bal,␣
 ↪y_train_bal,validation_data=(X_val_bal, y_val_bal), batch_size=100, epochs=20)
```

```
Train on 20532 samples, validate on 6844 samples
Epoch 1/20
20532/20532 [==============================] - 0s 22us/step - loss: 0.7757 -
accuracy: 0.7331 - val_loss: 0.5636 - val_accuracy: 0.7966
Epoch 2/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.5254 -
accuracy: 0.8122 - val_loss: 0.5217 - val_accuracy: 0.8101
Epoch 3/20
20532/20532 [==============================] - 0s 19us/step - loss: 0.4982 -
accuracy: 0.8220 - val_loss: 0.5072 - val_accuracy: 0.8203
Epoch 4/20
20532/20532 [==============================] - 0s 17us/step - loss: 0.4848 -
accuracy: 0.8252 - val_loss: 0.5120 - val_accuracy: 0.8213
Epoch 5/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4759 -
accuracy: 0.8281 - val_loss: 0.5023 - val_accuracy: 0.8233
Epoch 6/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4687 -
accuracy: 0.8326 - val_loss: 0.5102 - val_accuracy: 0.8188
Epoch 7/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4655 -
accuracy: 0.8319 - val_loss: 0.5008 - val_accuracy: 0.8223
Epoch 8/20
20532/20532 [==============================] - 0s 19us/step - loss: 0.4586 -
accuracy: 0.8322 - val_loss: 0.4903 - val_accuracy: 0.8277
Epoch 9/20
20532/20532 [==============================] - 0s 17us/step - loss: 0.4521 -
accuracy: 0.8363 - val_loss: 0.4909 - val_accuracy: 0.8255
Epoch 10/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4492 -
```

```
accuracy: 0.8372 - val_loss: 0.4884 - val_accuracy: 0.8260
Epoch 11/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4435 -
accuracy: 0.8391 - val_loss: 0.4904 - val_accuracy: 0.8280
Epoch 12/20
20532/20532 [==============================] - 0s 17us/step - loss: 0.4395 -
accuracy: 0.8407 - val_loss: 0.4892 - val_accuracy: 0.8274
Epoch 13/20
20532/20532 [==============================] - 0s 17us/step - loss: 0.4342 -
accuracy: 0.8444 - val_loss: 0.4900 - val_accuracy: 0.8276
Epoch 14/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4320 -
accuracy: 0.8449 - val_loss: 0.4864 - val_accuracy: 0.8280
Epoch 15/20
20532/20532 [==============================] - 0s 17us/step - loss: 0.4259 -
accuracy: 0.8488 - val_loss: 0.4874 - val_accuracy: 0.8295
Epoch 16/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4230 -
accuracy: 0.8478 - val_loss: 0.4870 - val_accuracy: 0.8292
Epoch 17/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4188 -
accuracy: 0.8495 - val_loss: 0.4864 - val_accuracy: 0.8311
Epoch 18/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4153 -
accuracy: 0.8522 - val_loss: 0.4811 - val_accuracy: 0.8331
Epoch 19/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4121 -
accuracy: 0.8530 - val_loss: 0.4808 - val_accuracy: 0.8324
Epoch 20/20
20532/20532 [==============================] - 0s 18us/step - loss: 0.4085 -
accuracy: 0.8539 - val_loss: 0.4835 - val_accuracy: 0.8302
```

Si può osservare un piccolo miglioramento sia nell'accuratezza sia nella loss in entrambi i set. Rispetto al modello addestrato con dati non bilanciati c'è stata una diminuzione di 0.1 nella loss e un aumento di circa 0.05 nell'accuratezza.

Si è deciso di proseguire testando un modello un po' più complesso.

Sono stati effettuati diversi test di modelli con diversi layer e numero di nodi. Quello mostrato di seguito ha ottenuto un risultato migliore rispetto agli altri.

[146]:
```
#questo modello è composto da 6 layer densi, di cui 2 di input/output, e 2 layer␣
 ↪di Dropout.
model_balanced = Sequential([
    Dense(516, activation="relu", input_dim=X_train_bal.shape[1]),
    Dropout(0.1),
    Dense(312,activation='relu'),
    Dense(128,activation='relu'),
    Dropout(0.1),
```

```
    Dense(68,activation='relu'),
    Dense(32,activation='relu'),
    Dense(8, activation="softmax")
])
```

[147]:
```
model_balanced.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

[148]:
```
fit_history = model_balanced.fit(X_train_bal, y_train_bal,␣
 ↪validation_data=(X_val_bal, y_val_bal), batch_size=150, epochs=20)
```

```
Train on 20532 samples, validate on 6844 samples
Epoch 1/20
20532/20532 [==============================] - 2s 96us/step - loss: 0.7032 -
accuracy: 0.7459 - val_loss: 0.5329 - val_accuracy: 0.8074
Epoch 2/20
20532/20532 [==============================] - 2s 87us/step - loss: 0.5186 -
accuracy: 0.8134 - val_loss: 0.5030 - val_accuracy: 0.8252
Epoch 3/20
20532/20532 [==============================] - 2s 86us/step - loss: 0.4898 -
accuracy: 0.8231 - val_loss: 0.5070 - val_accuracy: 0.8220
Epoch 4/20
20532/20532 [==============================] - 2s 86us/step - loss: 0.4776 -
accuracy: 0.8277 - val_loss: 0.5148 - val_accuracy: 0.8220
Epoch 5/20
20532/20532 [==============================] - 2s 87us/step - loss: 0.4545 -
accuracy: 0.8378 - val_loss: 0.4945 - val_accuracy: 0.8235
Epoch 6/20
20532/20532 [==============================] - 2s 86us/step - loss: 0.4463 -
accuracy: 0.8355 - val_loss: 0.4887 - val_accuracy: 0.8269
Epoch 7/20
20532/20532 [==============================] - 2s 86us/step - loss: 0.4379 -
accuracy: 0.8411 - val_loss: 0.4875 - val_accuracy: 0.8258
Epoch 8/20
20532/20532 [==============================] - 2s 88us/step - loss: 0.4271 -
accuracy: 0.8447 - val_loss: 0.4914 - val_accuracy: 0.8274
Epoch 9/20
20532/20532 [==============================] - 2s 89us/step - loss: 0.4087 -
accuracy: 0.8520 - val_loss: 0.5000 - val_accuracy: 0.8254
Epoch 10/20
20532/20532 [==============================] - 2s 89us/step - loss: 0.3961 -
accuracy: 0.8548 - val_loss: 0.4946 - val_accuracy: 0.8285
Epoch 11/20
20532/20532 [==============================] - 2s 85us/step - loss: 0.3798 -
```

```
accuracy: 0.8619 - val_loss: 0.4718 - val_accuracy: 0.8378
Epoch 12/20
20532/20532 [==============================] - 2s 87us/step - loss: 0.3618 -
accuracy: 0.8664 - val_loss: 0.4927 - val_accuracy: 0.8326
Epoch 13/20
20532/20532 [==============================] - 2s 86us/step - loss: 0.3481 -
accuracy: 0.8721 - val_loss: 0.5029 - val_accuracy: 0.8327
Epoch 14/20
20532/20532 [==============================] - 2s 85us/step - loss: 0.3369 -
accuracy: 0.8762 - val_loss: 0.4972 - val_accuracy: 0.8276
Epoch 15/20
20532/20532 [==============================] - 2s 86us/step - loss: 0.3175 -
accuracy: 0.8811 - val_loss: 0.4716 - val_accuracy: 0.8473
Epoch 16/20
20532/20532 [==============================] - 2s 87us/step - loss: 0.3025 -
accuracy: 0.8862 - val_loss: 0.5001 - val_accuracy: 0.8366
Epoch 17/20
20532/20532 [==============================] - 2s 86us/step - loss: 0.2863 -
accuracy: 0.8920 - val_loss: 0.4977 - val_accuracy: 0.8377
Epoch 18/20
20532/20532 [==============================] - 2s 85us/step - loss: 0.2699 -
accuracy: 0.8980 - val_loss: 0.5036 - val_accuracy: 0.8305
Epoch 19/20
20532/20532 [==============================] - 2s 85us/step - loss: 0.2517 -
accuracy: 0.9047 - val_loss: 0.5107 - val_accuracy: 0.8435
Epoch 20/20
20532/20532 [==============================] - 2s 86us/step - loss: 0.2412 -
accuracy: 0.9092 - val_loss: 0.5144 - val_accuracy: 0.8416
```

I risultati sono sicuramente migliori per quel che riguarda il train set che raggiunge un'accuratezza quasi del 90% e una loss molto bassa, quasi la metà dei modelli precedenti. Osservando però il validation set riscontriamo un aumento della loss e un leggero aumento dell'accuratezza. Nonostante le varie prove non siamo riusciti a ottenere dei valori ottimali della loss, ma ci pare complessivamente un risultato buono che abbiamo voluto riportare insieme a tutti gli altri.