ativ_6

May 30, 2023

```
[ ]: import matplotsoccer as mps
     import numpy as np
     import pandas as pd
     import scipy
     import socceraction.spadl as spd
     from sklearn.decomposition import NMF
     from sklearn.metrics import pairwise_distances
     from tqdm import tqdm
```

1 [CDAF] Atividade 6

1.1 Nome e matrícula

Nome: Igor Lacerda Faria da Silva Matrícula: 2020041973

1.2 Referências

- [1] https://www.ecmlpkdd2019.org/downloads/paper/701.pdf
- [2] https://dtai.cs.kuleuven.be/sports/blog/player-vectors:-characterizing-soccer-players'-playing-style
- [3] https://dtai.cs.kuleuven.be/sports/player_vectors
- [4] https://github.com/TomDecroos/matplotsocce

1.3 Introdução

- Nessa atividade, temos implementado o "Player Vectors", método proposto em [1] para caracterizar o estilo de jogo de jogadores baseado nas localizações que realizam cada tipo de ação.
- [2] apresenta o conteúdo do paper em [1] de forma mais resumida e visual, em formato de blog.
- [3] oferece uma demo interativa com uma aplicação do método no contexto de comparar a similaridade entre jogadores.
- [4] é uma biblioteca para plotar visualizações de partidas de futebol. Além disso, ela tem uma função pronta para criar heatmaps de ações de jogadores, que é util para o nosso contexto.

1.4 Intruções

- Para cada header abaixo do notebook, vocês devem explicar o que foi feito e à qual seção/subseção/equação de [1] ela corresponde. Justifique suas respostas.

- Além disso, vocês devem montar um diagrama do fluxo de funções/tarefas de toda a pipeline do Player Vectors abaixo. Esse diagrama deve ser enviado como arquivo na submissão do Moodle, para além deste notebook.

1.4.1 Carregando os dados

[ ]:
```python
def load_matches(path):
    matches = pd.read_json(path_or_buf=path)
    # as informações dos times de cada partida estão em um dicionário dentro da␣
 ↪coluna 'teamsData', então vamos separar essas informações
    team_matches = []
    for i in range(len(matches)):
        team_match = pd.DataFrame(matches.loc[i, "teamsData"]).T
        team_match["matchId"] = matches.loc[i, "wyId"]
        team_matches.append(team_match)
    team_matches = pd.concat(team_matches).reset_index(drop=True)

    return matches, team_matches
```

[ ]:
```python
def get_position(x):
    return x["name"]


def load_players(path):
    players = pd.read_json(path_or_buf=path)
    players["player_name"] = players["firstName"] + " " + players["lastName"]
    players["role"] = players["role"].apply(get_position)
    players = players[["wyId", "player_name", "role"]].rename(
        columns={"wyId": "player_id"}
    )

    return players
```

[ ]:
```python
def load_events(path):
    events = pd.read_json(path_or_buf=path)
    # pré processamento em colunas da tabela de eventos para facilitar a␣
 ↪conversão p/ SPADL
    events = events.rename(
        columns={
            "id": "event_id",
            "eventId": "type_id",
            "subEventId": "subtype_id",
            "teamId": "team_id",
            "playerId": "player_id",
            "matchId": "game_id",
        }
    )
```

```
    events["milliseconds"] = events["eventSec"] * 1000
    events["period_id"] = events["matchPeriod"].replace({"1H": 1, "2H": 2})

    return events
```

[ ]:
```
def load_minutes_played_per_game(path):
    minutes = pd.read_json(path_or_buf=path)
    minutes = minutes.rename(
        columns={
            "playerId": "player_id",
            "matchId": "game_id",
            "teamId": "team_id",
            "minutesPlayed": "minutes_played",
        }
    )
    minutes = minutes.drop(["shortName", "teamName", "red_card"], axis=1)

    return minutes
```

[ ]:
```
PATH_DATA = "data/wyscout"
```

[ ]:
```
leagues = ["England", "Spain"]
events = {}
matches = {}
team_matches = {}
minutes = {}
for league in leagues:
    path = f"{PATH_DATA}/matches/matches_{league}.json"
    matches[league], team_matches[league] = load_matches(path)
    path = f"{PATH_DATA}/events/events_{league}.json"
    events[league] = load_events(path)
    path = f"{PATH_DATA}/minutes_played/minutes_played_per_game_{league}.json"
    minutes[league] = load_minutes_played_per_game(path)
```

[ ]:
```
path = f"{PATH_DATA}/players.json"
players = load_players(path)
players["player_name"] = players["player_name"].str.decode("unicode-escape")
```

[ ]:
```
def calculate_minutes_per_season(minutes_per_game):
    minutes_per_season = minutes_per_game.groupby("player_id", as_index=False)[
        "minutes_played"
    ].sum()

    return minutes_per_season
```

Análise: este é apenas um carregamento de dados que não corresponde a nenhuma seção do artigo [1].

1.4.2 SPADL

[ ]:
```
def spadl_transform(events, team_matches):
    spadl = []
    game_ids = events.game_id.unique().tolist()
    for g in tqdm(game_ids):
        match_events = events.loc[events.game_id == g]
        match_home_id = team_matches.loc[
            (team_matches.matchId == g) & (team_matches.side == "home"),␣
 ↪"teamId"
        ].values[0]
        match_actions = spd.wyscout.convert_to_actions(
            events=match_events, home_team_id=match_home_id
        )
        match_actions = spd.play_left_to_right(
            actions=match_actions, home_team_id=match_home_id
        )
        match_actions = spd.add_names(match_actions)
        spadl.append(match_actions)
    spadl = pd.concat(spadl).reset_index(drop=True)

    return spadl
```

[ ]:
```
spadl = {}
for league in leagues:
    spadl[league] = spadl_transform(
        events=events[league], team_matches=team_matches[league]
    )
```

Análise: novamente, nesta seção, os dados são apenas transformados para o formato SPADL. É meio forçado dizer que isso corresponde a alguma seção do artigo [1].

1.4.3 Construção de Heatmaps

[ ]:
```
def construct_heatmaps(spadl, season_minutes, action_type):
    heatmaps = {}
    if action_type == "pass":
        heatmaps["start"] = {}
        heatmaps["end"] = {}
    for player_id in tqdm(season_minutes["player_id"].tolist()):
        mask = (spadl["player_id"] == player_id) & (spadl["type_name"] ==␣
 ↪action_type)
        player_actions = spadl[mask]
        if action_type != "pass":
            heatmaps[player_id] = mps.count(
                x=player_actions["start_x"], y=player_actions["start_y"], n=25,␣
 ↪m=16
```

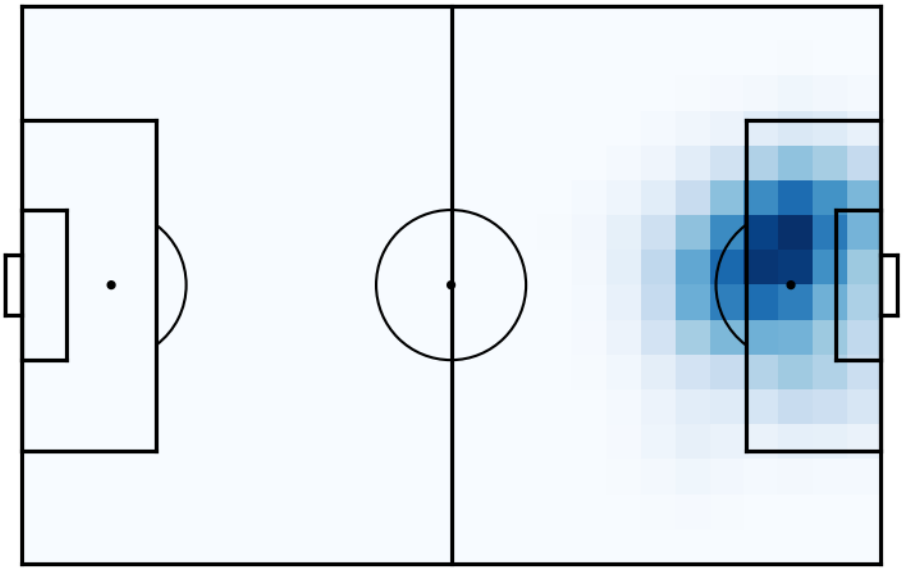
```
            )
            heatmaps[player_id] *= (
                90
                / season_minutes[season_minutes["player_id"] == player_id][
                    "minutes_played"
                ].values[0]
            )
            heatmaps[player_id] = scipy.ndimage.
↪gaussian_filter(heatmaps[player_id], 1)
        else:
            heatmaps["start"][player_id] = mps.count(
                x=player_actions["start_x"], y=player_actions["start_y"], n=25,␣
↪m=16
            )
            heatmaps["start"][player_id] *= (
                90
                / season_minutes[season_minutes["player_id"] == player_id][
                    "minutes_played"
                ].values[0]
            )
            heatmaps["start"][player_id] = scipy.ndimage.gaussian_filter(
                heatmaps["start"][player_id], 1
            )
            heatmaps["end"][player_id] = mps.count(
                x=player_actions["end_x"], y=player_actions["end_y"], n=25, m=16
            )
            heatmaps["end"][player_id] *= (
                90
                / season_minutes[season_minutes["player_id"] == player_id][
                    "minutes_played"
                ].values[0]
            )
            heatmaps["end"][player_id] = scipy.ndimage.gaussian_filter(
                heatmaps["end"][player_id], 1
            )

    return heatmaps
```

[ ]:
```
season_minutes = {}
for league in leagues:
    season_minutes[league] = calculate_minutes_per_season(
        minutes_per_game=minutes[league]
    )
    mask = (season_minutes[league]["minutes_played"] >= 900) & (
        season_minutes[league]["player_id"].isin(
            players.loc[players["role"] != "Goalkeeper", "player_id"]
        )
```

```
    )
    season_minutes[league] = season_minutes[league][mask]
```

[ ]:
```
action_types = ["shot", "pass", "cross", "dribble"]
heatmaps = {}
for league in leagues:
    heatmaps[league] = {}
    for at in action_types:
        heatmaps[league][at] = construct_heatmaps(
            spadl=spadl[league], season_minutes=season_minutes[league],␣
 ↪action_type=at
        )
```

[1]:
```
mps.heatmap(heatmaps["Spain"]["shot"][3359])
```



[1]: <Axes: >

Análise: este *header* corresponde à seção 4.2 do artigo, pois é feita a construção dos mapas de calor.

1.4.4 Comprimindo heatmaps para vetores

[ ]:
```
def heatmaps_to_vectors(heatmaps, action_type):
    if action_type != "pass":
        vectorized_heatmaps = np.array(
            [heatmaps[player_id].reshape(-1) for player_id in heatmaps.keys()]
        )
    else:
        vectorized_heatmaps = np.array(
            [
                np.concatenate(
                    [
                        heatmaps["start"][player_id].reshape(-1),
                        heatmaps["end"][player_id].reshape(-1),
                    ]
                )
                for player_id in heatmaps["start"].keys()
            ]
        )

    return vectorized_heatmaps
```

[ ]:
```
vectorized_heatmaps = {}
for league in leagues:
    vectorized_heatmaps[league] = {}
    for act_type in action_types:
        vectorized_heatmaps[league][act_type] = heatmaps_to_vectors(
            heatmaps=heatmaps[league][act_type], action_type=act_type
        )
```

Análise: já este *header* corresponde ao começo da seção 4.3, que transforma os mapas de calor em vetores.

1.4.5 NMF

[ ]:
```
def nmf_decomposition(vectorized_heatmaps, n_components):
    nmf = NMF(n_components=n_components, init="nndsvda", random_state=0)
    nmf.fit(vectorized_heatmaps)

    return nmf
```

[ ]:
```
n_components = {"shot": 4, "pass": 5, "cross": 4, "dribble": 5}
concat_vectors = {}
nmfs = {}
for act_type in action_types:
    concat_vectors[act_type] = np.concatenate(
        [
```

```
            vectorized_heatmaps["England"][act_type],
            vectorized_heatmaps["Spain"][act_type],
        ]
    )
    nmfs[act_type] = nmf_decomposition(
        vectorized_heatmaps=concat_vectors[act_type],
        n_components=n_components[act_type],
    )
```

Análise: este *header* também corresponde à seção 4.3. Aqui é feita a transformação dos vetores em matriz e a redução da dimensionalidade dessa matriz, usando o NMF.

1.4.6 Reconstruction Evaluation

[2]:
```
for act_type in action_types:
    print(f"{act_type} Reconstruction evaluation\n")
    print(f"Reconstruction error from NMF object: {nmfs[act_type].
 ↪reconstruction_err_}")
    reconst_vectors = np.dot(
        nmfs[act_type].components_.T,
        nmfs[act_type].transform(concat_vectors[act_type]).T,
    ).T
    reconst_error = np.sqrt(np.sum((concat_vectors[act_type] - reconst_vectors)␣
 ↪** 2))
    print(f"Manual reconstruction error: {reconst_error}")
    print(f"Mean reconstruction error: {reconst_error/concat_vectors[act_type].
 ↪shape[1]}")
    print("---------------------------------\n")
```

```
shot Reconstruction evaluation

Reconstruction error from NMF object: 1.322487922632152
Manual reconstruction error: 1.3224879184800893
Mean reconstruction error: 0.0033062197962002234
---------------------------------

pass Reconstruction evaluation

Reconstruction error from NMF object: 28.19626189362776
Manual reconstruction error: 28.196261647153566
Mean reconstruction error: 0.035245327058941955
---------------------------------

cross Reconstruction evaluation

Reconstruction error from NMF object: 2.3365384800195814
Manual reconstruction error: 2.336538445866538
```

```
Mean reconstruction error: 0.005841346114666345
--------------------------------

dribble Reconstruction evaluation

Reconstruction error from NMF object: 4.008096593968764
Manual reconstruction error: 4.008096580590021
Mean reconstruction error: 0.010020241451475051
--------------------------------
```

Análise: este cabeçalho não faz referência ao texto em si, é apenas uma comparação entre algumas maneiras de se fazer a reconstrução da matriz (além do NMF).

1.4.7 Deanonymization Evaluation

[ ]:
```
matches_1st = {}
matches_2nd = {}
spadl_1st = {}
spadl_2nd = {}
season_minutes_1st = {}
season_minutes_2nd = {}
for league in leagues:
    matches[league] = matches[league].sort_values(by="dateutc").
 ↪reset_index(drop=True)
    matches_1st[league] = (
        matches[league].loc[: int(len(matches[league]) / 2) - 1, "wyId"].values.
 ↪tolist()
    )
    matches_2nd[league] = (
        matches[league].loc[int(len(matches[league]) / 2) :, "wyId"].values.
 ↪tolist()
    )
    season_minutes_1st[league] = calculate_minutes_per_season(
        minutes[league][minutes[league]["game_id"].isin(matches_1st[league])]
    )
    season_minutes_2nd[league] = calculate_minutes_per_season(
        minutes[league][minutes[league]["game_id"].isin(matches_2nd[league])]
    )
    season_minutes_1st[league] = season_minutes_1st[league][
        season_minutes_1st[league]["minutes_played"] >= 900
    ]
    season_minutes_2nd[league] = season_minutes_2nd[league][
        season_minutes_2nd[league]["minutes_played"] >= 900
    ]
    season_minutes_1st[league] = season_minutes_1st[league][
        season_minutes_1st[league]["player_id"].isin(
```

```
            season_minutes_2nd[league]["player_id"]
        )
    ]
    season_minutes_2nd[league] = season_minutes_2nd[league][
        season_minutes_2nd[league]["player_id"].isin(
            season_minutes_1st[league]["player_id"]
        )
    ]
    mask_1st = (spadl[league]["game_id"].isin(matches_1st[league])) & (
        spadl[league]["player_id"].isin(season_minutes_1st[league]["player_id"])
    )
    spadl_1st[league] = spadl[league][mask_1st]
    mask_2nd = (spadl[league]["game_id"].isin(matches_2nd[league])) & (
        spadl[league]["player_id"].isin(season_minutes_2nd[league]["player_id"])
    )
    spadl_2nd[league] = spadl[league][mask_2nd]
```

[ ]:
```
heatmaps_1st = {}
heatmaps_2nd = {}
for league in leagues:
    heatmaps_1st[league] = {}
    heatmaps_2nd[league] = {}
    for at in action_types:
        heatmaps_1st[league][at] = construct_heatmaps(
            spadl=spadl_1st[league],
            season_minutes=season_minutes_1st[league],
            action_type=at,
        )
        heatmaps_2nd[league][at] = construct_heatmaps(
            spadl=spadl_2nd[league],
            season_minutes=season_minutes_2nd[league],
            action_type=at,
        )
```

[ ]:
```
vectorized_heatmaps_1st = {}
vectorized_heatmaps_2nd = {}
for league in leagues:
    vectorized_heatmaps_1st[league] = {}
    vectorized_heatmaps_2nd[league] = {}
    for act_type in action_types:
        vectorized_heatmaps_1st[league][act_type] = heatmaps_to_vectors(
            heatmaps=heatmaps_1st[league][act_type], action_type=act_type
        )
        vectorized_heatmaps_2nd[league][act_type] = heatmaps_to_vectors(
            heatmaps=heatmaps_2nd[league][act_type], action_type=act_type
        )
```

[ ]:
```
def coefficients_transform(vectorized_heatmaps, nmf):
    return nmf.transform(vectorized_heatmaps)
```

[ ]:
```
vectorized_coefs_1st = {}
vectorized_coefs_2nd = {}
for league in leagues:
    vectorized_coefs_1st[league] = {}
    vectorized_coefs_2nd[league] = {}
    for act_type in action_types:
        vectorized_coefs_1st[league][act_type] = coefficients_transform(
            vectorized_heatmaps=vectorized_heatmaps_1st[league][act_type],
            nmf=nmfs[act_type],
        )
        vectorized_coefs_2nd[league][act_type] = coefficients_transform(
            vectorized_heatmaps=vectorized_heatmaps_2nd[league][act_type],
            nmf=nmfs[act_type],
        )
```

[ ]:
```
player_vectors_1st = {}
player_vectors_2nd = {}
for league in leagues:
    player_vectors_1st[league] = np.concatenate(
        [vectorized_coefs_1st[league][act_type] for act_type in action_types],␣
 ↪axis=1
    )
    player_vectors_2nd[league] = np.concatenate(
        [vectorized_coefs_2nd[league][act_type] for act_type in action_types],␣
 ↪axis=1
    )
```

[ ]:
```
player_vectors_1st = np.concatenate([player_vectors_1st[league] for league in␣
 ↪leagues])
player_vectors_2nd = np.concatenate([player_vectors_2nd[league] for league in␣
 ↪leagues])
```

[ ]:
```
player_ids = []
for league in leagues:
    player_ids += list(heatmaps_1st[league]["shot"].keys())
```

[ ]:
```
D = pairwise_distances(player_vectors_1st, player_vectors_2nd,␣
 ↪metric="manhattan")

# sort each row
# k_d = np.sort(D, axis=1)
# sort each row and replace distances by index
k_i = np.argsort(D, axis=1)
# replace indices by player ids
```

```
p_i = np.take(player_ids, k_i, axis=0)

rs = np.argmax(
    np.array([p_i[i, :] == player_ids[i] for i in range(p_i.shape[0])]), axis=1
)


def mean_reciprocal_rank(rs):
    return np.mean(1.0 / (rs + 1))


def top_k(rs, k):
    return (rs < k).sum() / len(rs)


mrr = mean_reciprocal_rank(rs)
top1 = top_k(rs, 1)
top3 = top_k(rs, 3)
top5 = top_k(rs, 5)
top10 = top_k(rs, 10)
```

[3]:
```
print(f"Top 1 = {round(top1 * 100, 1)}%")
print(f"Top 3 = {round(top3 * 100, 1)}%")
print(f"Top 5 = {round(top5 * 100, 1)}%")
print(f"Top 10 = {round(top10 * 100, 1)}%")
print(f"MRR = {round(mrr, 3)}")
```

```
Top 1 = 38.3%
Top 3 = 61.4%
Top 5 = 70.5%
Top 10 = 81.9%
MRR = 0.532
```

Análise: este cabeçalho é referente à seção 5.4, em que se tenta prever os jogadores com base em seu estilo de jogo.

1.4.8 Explore Similar Players

[ ]:
```
vectorized_coefs = {}
for league in leagues:
    vectorized_coefs[league] = {}
    for act_type in action_types:
        vectorized_coefs[league][act_type] = coefficients_transform(
            vectorized_heatmaps=vectorized_heatmaps[league][act_type],
            nmf=nmfs[act_type],
        )
```

[ ]:
```
player_vectors = {}
for league in leagues:
    player_vectors[league] = np.concatenate(
        [vectorized_coefs[league][act_type] for act_type in action_types],␣
 ↪axis=1
    )
```

[ ]:
```
player_vectors = np.concatenate([player_vectors[league] for league in leagues])
```

[ ]:
```
player_ids = []
for league in leagues:
    player_ids += list(heatmaps[league]["shot"].keys())
```

[ ]:
```
D = pairwise_distances(player_vectors, player_vectors, metric="manhattan")
# sort each row
# k_d = np.sort(D, axis=1)
# sort each row and replace distances by index
k_i = np.argsort(D, axis=1)
# replace indices by player ids
p_i = np.take(player_ids, k_i, axis=0)
```

[ ]:
```
similar_players = pd.DataFrame(
    data=p_i[:, :11],
    columns=["player_id"] + [f"{i}th_similar" for i in range(1, 11)],
)
```

[ ]:
```
players = players[players["player_id"].isin(similar_players["player_id"])].
 ↪reset_index(
    drop=True
)
```

[ ]:
```
id_to_name = {}
for i in range(len(players)):
    id_to_name[players.loc[i, "player_id"]] = players.loc[i, "player_name"]
```

[4]:
```
similar_players = similar_players.replace(id_to_name)
similar_players
```

[4]:

	player_id	…	10th_similar
0	Toby Alderweireld	…	Daniele Bonera
1	Jan Vertonghen	…	Antonio Rüdiger
2	Christian Dannemann Eriksen	…	Paul Pogba
3	Ragnar Klavan	…	Aymeric Laporte
4	Johann Berg Guðmundsson	…	Sergio Gontán Gallardo
..	…	…	…
628	Roberto Suárez Pier	…	Claudio Ariel Yacob

```
629             Daniel Raba Antolí  …          Andros Townsend
630               Oghenekaro Etebo  …       Idrissa Gana Gueye
631             Youssef En-Nesyri   …              Jamie Vardy
632  Martín Aguirregabiria Padilla  …  Martín Montoya Torralbo

[633 rows x 11 columns]
```

Análise: este cabeçalho faz referêcia à seção 5.2, que compara jogadores similares. Lá o objetivo é o scouting, tentando validar o modelo, fazendo comparações de jogadores com base na opinião popular.