ativ_5

May 24, 2023

[ ]:
```
import numpy as np
import pandas as pd
import sklearn.metrics as mt
import socceraction.spadl as spd
import socceraction.vaep.formula as fm
import socceraction.vaep.labels as lab
import xgboost as xgb
from socceraction.vaep import features as ft
from tqdm import tqdm
```

1 [CDAF] Atividade 5

1.1 Nome e matrícula

Nome: Igor Lacerda Faria da Silva Matrícula: 2020041973

1.2 Referências

- [1] https://tomdecroos.github.io/reports/kdd19_tomd.pdf
- [2] https://socceraction.readthedocs.io/en/latest/api/vaep.html
- [3] https://socceraction.readthedocs.io/en/latest/documentation/valuing_actions/vaep.html
- [4] https://github.com/ML-KULeuven/socceraction/tree/master/public-notebooks

1.3 Introdução

- Nessa atividade, temos implementada a pipeline inteira do VAEP [1] para os dados do Wyscout das Top 5 ligas.
- [2] é a documentação das funções do VAEP na API do socceraction.
- [3] apresenta uma explicação do framework com uma mistura de intuição, matemática e código.
- [4] são notebooks públicos que implementam o VAEP para outro conjunto de dados.

1.4 Instruções

- Para cada header do notebook abaixo, vocês devem explicar o que foi feito e à qual seção/subseção/equação do paper "Actions Speak Louder than Goals: Valuing Actions by Estimating Probabilities" ela corresponde. Justifique suas respostas.
- Além disso, após algumas partes do código haverão perguntas que vocês devem responder, possivelmente explorando minimamente o que já está pronto.

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

1.4.1 Carregando os dados

[ ]:
```
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)):
        match = pd.DataFrame(matches.loc[i, "teamsData"]).T
        match["matchId"] = matches.loc[i, "wyId"]
        team_matches.append(match)
    team_matches = pd.concat(team_matches).reset_index(drop=True)

    return team_matches
```

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

    return players
```

[ ]:
```
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
```

[ ]:
```python
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
```

[ ]:
```python
BASE_DIR = "data"
```

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

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

Análise Eu diria que esse trecho faz referência à subseção 2.1 do artigo, pois consiste em um pré-processamento para carregar os dados, que depois vão ser convertidos no formato SPADL. Esse trecho explora como os formatos de dados de diferentes provedoras *não são* uniformes, e mostra como é o pré-processamento de dados do Wyscout.

1.4.2 SPADL

[ ]:
```python
def spadl_transform(events, 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 = matches.loc[
            (matches.matchId == g) & (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],␣
 ↪matches=matches[league])
```

Análise Esse trecho faz referência à seção 2.2, pois os dados são transformados no formato SPADL.

1.4.3 Features

[ ]:
```
def features_transform(spadl):
    spadl.loc[spadl.result_id.isin([2, 3]), ["result_id"]] = 0
    spadl.loc[spadl.result_name.isin(["offside", "owngoal"]), ["result_name"]]␣
 ↪= "fail"

    xfns = [
        ft.actiontype_onehot,
        ft.bodypart_onehot,
        ft.result_onehot,
        ft.goalscore,
        ft.startlocation,
        ft.endlocation,
        ft.team,
        ft.time,
        ft.time_delta,
    ]

    features = []
    for game in tqdm(np.unique(spadl.game_id).tolist()):
        match_actions = spadl.loc[spadl.game_id == game].reset_index(drop=True)
        match_states = ft.gamestates(actions=match_actions)
        match_feats = pd.concat([fn(match_states) for fn in xfns], axis=1)
        features.append(match_feats)
    features = pd.concat(features).reset_index(drop=True)
```

```
    return features
```

1- O que a primeira e a segunda linhas da função acima fazem? Qual sua hipótese sobre intuito dessas transformações? Como você acha que isso pode impactar o modelo final?

Resposta No formato SPADL, existem mais de dois tipo de resultado para uma ação. Isto é, o resultado pode ser sucesso, falha ou outra coisa, que, no geral, pode ser considerada falha. Desse modo, o resultado dessas ações é convertido para falha. Creio que impedimentos são um tanto que neutros para um time (ao menos não positivos), e fazer gols contra nem se fala, então, a princípio, não vejo como essa transformação pode ser ruim. Imagino que isso não deve causar grandes impactos no modelo final, porque esses resultados são bastante incomuns.

[ ]:
```
features = {}
for league in ["England", "Spain"]:
    features[league] = features_transform(spadl[league])
```

Análise Esse trecho faz alusão à subseção 4.2 do artigo, em que são feitos ajustes nos dados para que o desempenho dos modelos seja melhor.

1.4.4 Labels

[ ]:
```
def labels_transform(spadl):
    yfns = [lab.scores, lab.concedes]

    labels = []
    for game in tqdm(np.unique(spadl.game_id).tolist()):
        match_actions = spadl.loc[spadl.game_id == game].reset_index(drop=True)
        labels.append(pd.concat([fn(actions=match_actions) for fn in yfns],␣
 ↪axis=1))

    labels = pd.concat(labels).reset_index(drop=True)

    return labels
```

[ ]:
```
labels = {}
for league in ["England", "Spain"]:
    labels[league] = labels_transform(spadl[league])
```

[1]:
```
labels["England"]["scores"].sum()
```

[1]: 7553

[2]:
```
labels["England"]["concedes"].sum()
```

[2]: 2313

2- Explique o por que da quantidade de labels positivos do tipo scores ser muito maior que do concedes. Como você acha que isso pode impactar o modelo final?

Resposta Geralmente, a maioria das ações que os jogadores fazem, tem como objetivo aumentar a chance de fazer gols. Dessa maneira, o esperado é que existam menos ações em que o efeito contrário é atingido. Isso impacta fortemente o treinamento do modelo, uma vez que ele pode ficar enviesado para labels positivos do tipo *scores*.

Análise Neste trecho, são construídas as labels da seção 4.1. Isso é bem sugestivo pelo `yfns` da função `labels_transform`.

1.4.5 Training Model

[ ]:
```
def train_vaep(X_train, y_train, X_test, y_test):
    models = {}
    for m in ["scores", "concedes"]:
        models[m] = xgb.XGBClassifier(random_state=0, n_estimators=50,␣
 ↪max_depth=3)

        print("training " + m + " model")
        models[m].fit(X_train, y_train[m])

        p = sum(y_train[m]) / len(y_train[m])
        base = [p] * len(y_train[m])
        y_train_pred = models[m].predict_proba(X_train)[:, 1]
        train_brier = mt.brier_score_loss(
            y_train[m], y_train_pred
        ) / mt.brier_score_loss(y_train[m], base)
        print(m + " Train NBS: " + str(train_brier))
        print()

        p = sum(y_test[m]) / len(y_test[m])
        base = [p] * len(y_test[m])
        y_test_pred = models[m].predict_proba(X_test)[:, 1]
        test_brier = mt.brier_score_loss(y_test[m], y_test_pred) / mt.
 ↪brier_score_loss(
            y_test[m], base
        )
        print(m + " Test NBS: " + str(test_brier))
        print()

        print("---------------------------------------")

    return models
```

[3]:
```
models = train_vaep(
    X_train=features["England"],
```

```
    y_train=labels["England"],
    X_test=features["Spain"],
    y_test=labels["Spain"],
)
```

```
training scores model
scores Train NBS: 0.8452154331687597

scores Test NBS: 0.850366923253325

----------------------------------------
training concedes model
concedes Train NBS: 0.964463215550682

concedes Test NBS: 0.9745272575372074

----------------------------------------
```

3- Por que treinamos dois modelos diferentes? Por que a performance dos dois é diferente?

Resposta As ações que favorecem um gol não necessariamente desfavorecem um gol do time inimigo, isto é, essas probabilidades não são complementares. Estranhamente, mesmo com menos dados, o `concedes` tem uma performance melhor. Isso deve acontecer pois as ações que aumentam a chance de levar gol são mais bem definidas (possuem uma variância menor) do que as ações que levam o time a fazer gols.

Análise Esse cabeçalho faz reverência à seção 4, em que é apresentada discussão do cálculo do VAEP e os algoritmos.

1.4.6 Predictions

[ ]:
```
def generate_predictions(features, models):
    preds = {}
    for m in ["scores", "concedes"]:
        preds[m] = models[m].predict_proba(features)[:, 1]
    preds = pd.DataFrame(preds)

    return preds
```

[4]:
```
preds = {}
preds["Spain"] = generate_predictions(features=features["Spain"], models=models)
preds["Spain"]
```

[4]:

	scores	concedes
0	0.004560	0.000367
1	0.003573	0.000347
2	0.002895	0.000345

```
3         0.002162  0.000318
4         0.002424  0.001799
…              …         …
473889    0.033276  0.002812
473890    0.041886  0.002787
473891    0.017484  0.004722
473892    0.007541  0.012254
473893    0.005007  0.047561

[473894 rows x 2 columns]
```

Análise A seção 5, no geral, explora a performance do modelo, avaliando diferentes estatísticas, tal qual as previsões são geradas neste cabeçalho.

1.4.7 Action Values

[ ]:
```python
def calculate_action_values(spadl, predictions):
    action_values = fm.value(
        actions=spadl, Pscores=predictions["scores"],␣
 ↪Pconcedes=predictions["concedes"]
    )
    action_values = pd.concat(
        [
            spadl[
                [
                    "original_event_id",
                    "action_id",
                    "game_id",
                    "start_x",
                    "start_y",
                    "end_x",
                    "end_y",
                    "type_name",
                    "result_name",
                    "player_id",
                ]
            ],
            predictions.rename(columns={"scores": "Pscores", "concedes":␣
 ↪"Pconcedes"}),
            action_values,
        ],
        axis=1,
    )

    return action_values
```

[5]:
```
action_values = {}
action_values["Spain"] = calculate_action_values(
    spadl=spadl["Spain"], predictions=preds["Spain"]
)
action_values["Spain"]
```

[5]:

```
        original_event_id  action_id  game_id  …  offensive_value
defensive_value  vaep_value
0               180864419          0  2565548  …         0.000000
-0.000000    0.000000
1               180864418          1  2565548  …        -0.000987
0.000020   -0.000967
2               180864420          2  2565548  …        -0.000678
0.000002   -0.000676
3               180864421          3  2565548  …        -0.000733
0.000027   -0.000706
4               180864422          4  2565548  …         0.000262
-0.001481   -0.001219
…                     …        …       …  …               …
…          …
473889          253302671       1482  2565927  …         0.017300
-0.000799    0.016501
473890          253302673       1483  2565927  …         0.008610
0.000025    0.008635
473891          253302674       1484  2565927  …        -0.024402
-0.001935   -0.026337
473892          253302698       1485  2565927  …         0.002820
0.005230    0.008050
473893          253302695       1486  2565927  …        -0.002535
-0.035307   -0.037841

[473894 rows x 15 columns]
```

[6]:
```
valuable_actions = action_values["Spain"].query("Pscores >= 0.95")
valuable_actions
```

[6]:

```
        original_event_id  action_id  game_id  …  offensive_value
defensive_value  vaep_value
1466            180054901        321  2565549  …         0.873940
-0.001370    0.872570
1569            180055162        424  2565549  …         0.962413
0.000043    0.962456
1830            180055441        685  2565549  …         0.857346
-0.000260    0.857086
```

```
2190          180055988       1045  2565549  …         0.880705
-0.000886    0.879819
3881          180136736        340  2565551  …         0.891102
-0.001473    0.889629
…                  …        …       …  …              …
…         …
472245        253222480       1028  2565926  …         0.931163
-0.000460    0.930703
472594        253300739        187  2565927  …         0.911694
-0.001317    0.910378
472948        253301228        541  2565927  …         0.854947
-0.000104    0.854843
473572        253302208       1165  2565927  …         0.920801
0.001016    0.921817
473806        253302547       1399  2565927  …         0.922795
-0.000122    0.922674

[884 rows x 15 columns]
```

4- Explore as ações com Pscores >= 0.95. Por que elas tem um valor tão alto? As compare com ações do mesmo tipo e resultado opostado. Será que o modelo aprende que essa combinação de tipo de ação e resultado está diretamente relacionado à variável y que estamos tentando prever?

5- Qual formula do paper corresponde à coluna `offensive_value` do dataframe `action_values`? E a coluna `defensive_value`?

Respostas

4) Todas as ações com $P_{scores} \geq 0.95$ são chutes (ao gol), que, no geral, incrementam muito a chance de se fazer gol. Por outro lado, quando essas ações dão errado, elas diminuem muito a chance de se marcar. Outras ações não sofrem desse viés: essa alteração drástica ocorre exclusivamente com chutes. Dessa maneira, essas outras ações acabam não sendo tão valorizadas.

5) Às equações 1 e 2, respectivamente.

Análise Não sei avaliar a qual seção este código em particular se refere, uma vez que ele estende a tabela para conter os dados do trecho anterior.

1.4.8 Player Ratings

[ ]:
```python
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
```

[ ]:
```
minutes_per_season = {}
minutes_per_season["Spain"] = calculate_minutes_per_season(minutes["Spain"])
```

[ ]:
```
def calculate_player_ratings(action_values, minutes_per_season, players):
    player_ratings = (
        action_values.groupby(by="player_id", as_index=False)
        .agg({"vaep_value": "sum"})
        .rename(columns={"vaep_value": "vaep_total"})
    )
    player_ratings = player_ratings.merge(
        minutes_per_season, on=["player_id"], how="left"
    )
    player_ratings["vaep_p90"] = (
        player_ratings["vaep_total"] / player_ratings["minutes_played"] * 90
    )
    player_ratings = (
        player_ratings[player_ratings["minutes_played"] >= 600]
        .sort_values(by="vaep_p90", ascending=False)
        .reset_index(drop=True)
    )
    player_ratings = player_ratings.merge(players, on=["player_id"], how="left")
    player_ratings = player_ratings[
        ["player_id", "player_name", "minutes_played", "vaep_total", "vaep_p90"]
    ]

    return player_ratings
```

[7]:
```
player_ratings = {}
player_ratings["Spain"] = calculate_player_ratings(
    action_values=action_values["Spain"],
    minutes_per_season=minutes_per_season["Spain"],
    players=players,
)
player_ratings["Spain"].nlargest(5, "vaep_p90")
```

[7]:

	player_id	player_name	minutes_played	vaep_total	vaep_p90
0	3359	Lionel Andrés Messi Cuccittini	3108.0	35.891377	1.039326
1	8278	Gareth Frank Bale	1850.0	14.323647	0.696826
2	3802	Philippe Coutinho Correia	1329.0	10.036555	0.679676
3	3322	Cristiano Ronaldo dos Santos Aveiro	2355.0	17.279105	0.660348
4	3682	Antoine Griezmann	2591.0	18.149570	

```
0.630437
```

6- Acha que o Top 5 da lista é bem representativo? Compare esse ranqueamento do VAEP com o do xT da Atividade 4. Qual você acha que é mais representativo?

Resposta Com toda certeza esse top 5 é representativo, todos os jogadores são excepcionais. Eu diria que ele é mais representativo que o Top 10 da atividade 4. Como apresentado no artigo, o VAEP tende a ser uma métrica mais acurada para avaliar jogadores em diferentes contextos (tendo como base, por exemplo, o valor dos jogadores).

Análise Este trecho final corresponde à subseção 5.5, como é evidente pelas avaliações dos jogadores.