luis_felipe_ramos_ferreira

May 22, 2023

1 [CDAF] Atividade 5

1.1 Nome e matrícula

Nome: Luís Felipe Ramos Ferreira Matrícula: 2019022553

1.2 Objetivos

- Nessa atividade, estou entregando a pipeline inteira do VAEP implementada para os dados do Wyscout das Top 5 ligas.
- Para cada subtítulo 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.

1.3 Referências

- [1] https://tomdecroos.github.io/reports/kdd19_tomd.pdf
- [2] https://socceraction.readthedocs.io/en/latest/api/vaep.html

1.3.1 Carregando os dados

Nesa seção, basicamente são carregados para memória os dados de evento, das partidas e dos jogadores referentes às ligas da Espanha e da Inglaterra da temporada 2017/1018, disponibilizadas gratuitamente pela empresa Wyscout.

De maneira geral, esse carregamento tenta fazer alguns pré processamentos para facilitar a conversão dos dados em questão para o formato SPADL, então o subtítulo se refere principalmente à seção 2 do artigo, que trata da linguagem de descrição de ações de jogadores SPADL e de suas diferenças com os outros formatos de dados disponibilizados pelas grandes empresas de analytics no ramo de futebol.

[1]:
```
import numpy as np
import pandas as pd
```

[2]:
```
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
```

[3]:
```
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
```

[4]:
```
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
```

[5]:
```
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
```

[6]:
```
leagues = ["England", "Spain"]
events = {}
matches = {}
minutes = {}
for league in leagues:
    path = f"../data/atv03/matches/matches_{league}.json"
    matches[league] = load_matches(path)
    path = f"../data/atv03/events/events_{league}.json"
    events[league] = load_events(path)
    path = f"../data/atv03/minutes_played/minutes_played_per_game_{league}.json"
    minutes[league] = load_minutes_played_per_game(path)
```

[7]:
```
path = "../data/atv03/players/players.json"
players = load_players(path)
players["player_name"] = players["player_name"].str.decode("unicode-escape")
```

1.3.2 SPADL

Neste subtítulo, é utilizada a biblioteca *socceraction* para converter os dados carregados anteriormente para o formato SPADL, com o objetivo de facilitar o uso do framework VAEP.

Assim como no último caso, essa parte se refere à seção 2 do artigo, mais especificamente a seção 2.2.

[8]:
```
from tqdm import tqdm
import socceraction.spadl as spd
```

[9]:
```
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
```

[10]:
```
spadl = {}
""" for league in leagues:
    spadl[league] = spadl_transform(events=events[league],␣
 ↪matches=matches[league])
spadl["England"].to_json("../data/atv05/spadl_England.json", orient="records")
spadl["Spain"].to_json("../data/atv05/spadl_Spain.json", orient="records") """
```

[10]:
```
' for league in leagues:\n    spadl[league] =
spadl_transform(events=events[league], matches=matches[league])\nspadl["England"
].to_json("../data/atv05/spadl_England.json",
orient="records")\nspadl["Spain"].to_json("../data/atv05/spadl_Spain.json",
orient="records") '
```

[11]:
```
spadl["England"] = pd.read_json("../data/atv05/spadl_England.json")
spadl["Spain"] = pd.read_json("../data/atv05/spadl_Spain.json")
```

1.3.3 Features

Neste subtítulo, é utilizada novamente a biblioteca *socceraction* para extração das *features* desejadas para cálculo da VAEP.

Como descrito na seção 4 do artigo, o *framework* da VAEP é feito em cima de descrições de *features* relacionadas às ações dos jogadores. Na subseção 4.2, mais especificamente, é descrito como construir as *features* desejadas para obter maior performance no modelo proposto.

Na função implementada *features_transform*, podemos ver o uso da biblioteca *socceraction* para calcular as *features* desejadas para cada partida no conjunto de dados passado como parâmetro da função.

[12]:
```
from socceraction.vaep import features as ft
```

[13]:
```
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?

A primeira e a segunda linha da função mudam o *id* do resultados de ações que foram impedimento ou gol contra para 0, ao invés de 2 e 3, respectivamente, assim como mudam o nome do resultado deste tipo de ação para *fail*. Em suma, as linhas manipulam o conjunto de dados de modo a considerar ações que resultaram em impedimento ou gol contra como ações que falharam.

O intuito dessas transofrmações é simplificar o modelo de modo que possamos olhar o resultado de ações de uma forma binária. Ou seu resultado foi sucesso, ou seu resultado foi fracasso. Podemos caracterizar isso dessa maneira pois lances que levaram a impedimentos ou gols contra intuitivamente podem ser vistos como lances que falharam.

Como gols contra são lances raros, acredito que essa mudança de caracterização não gere grande impacto no modelo. Em relação a impedimentos, no entanto, enxergo que em algumas situações isso possa prejudicar a classificação. Podemos imaginar cenários em que os times construíram uma boa jogada e devido a uma pequena diferença de posicionamento entre jogadores o impedimento foi marcado e o resultado classificado como falha, quando no panorama geral a construção da jogada possa ser classificada como positiva.

No entanto, creio que o impacto causado por estes detalhes também não seja tão grande, e que a VAEP ainda gere resultados muito interessantes mesmo após a caracterização descrita na função.

[14]:
```
features = {}
for league in ["England", "Spain"]:
    features[league] = features_transform(spadl[league])
```

```
100%|      | 380/380 [00:06<00:00, 54.67it/s]
100%|      | 380/380 [00:07<00:00, 50.79it/s]
```

1.3.4 Labels

Neste subtítulo, assim como no anterior, são tratados temas relacionados à seção 4 do artigo. Mais especificamente, sobre a subseção 4.1, que trata da construção de *labels*.

A função *labels_transform* faz também uso da biblioteca *socceraction* para calcular as *labels* das partidas desejadas, sejam as *labels* de marcar um gol ou de conceder um gol. Por padrão da biblioteca, é utilizado um número de ações vistas no futuro igual a 10 para classificar um estado S como 1 ou como 0.

[15]:
```
import socceraction.vaep.labels as lab
```

[16]:
```
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
```

[17]:
```
labels = {}
for league in ["England", "Spain"]:
    labels[league] = labels_transform(spadl[league])
```

```
100%|      | 380/380 [00:08<00:00, 43.06it/s]
100%|      | 380/380 [00:08<00:00, 42.97it/s]
```

[18]:
```
labels["England"]["scores"].sum()
```

[18]: 7553

[19]:
```
labels["England"]["concedes"].sum()
```

[19]: 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?

O motivo dessa diferença é consequência direta do fato de que todas as ações de um time sempre tem como objetivo reduzir a probabilidade de se conceder um gol. Dessa forma, dado que um time T está com a posse de bola num estado S, suas próximas k ações terão como objetivo diminuir as chances de se tomar um gol, ou também aumentar as chances de marcar. Dessa forma, o número de *labels* positivas do tipo 8concedes* será muito menor do que do tipo *scores*.

Tal relação entre as *labels* citadas pode impactar o modelo final de diferentes maneiras. De forma resumida, pode-se assumir que o classificador irá valorizar mais jogadas que aumentem as chances de marcar e fique emviesado para as *labels* positivas do tipo *scores*. Esta situação deve ser um dos focos do cientista de dados modelando a tarefa, de modo que essa diferença não prejudique os valores fianis gerados pela métrica.

1.3.5 Training Model

Neste subtítulo, utilizamos o treino do modelo da VAEP, utilizando os dados do campeonato ingês como treino e os dados do campeonato espanhol como teste. Foram criados modelos para classificar as probabilidades de se marcar gol e de se conceder um gol a partir de cada ação.

O classificador utilizado foi o *XGBoost* (no artigo, é destacado o uso de *CatBoost*), e, para avaliação dos resultados, foi utilizada a métrica *brier_score_loss*. Essa parte corresponde principalmente à seção 3 do artigo, em que é definida a maneira como o framework classifica as ações e calcula seus valores, além de mostrar como converter probabilidades de conceder ou marcar gols nessa métrica de valor por ação. Na seção 4 do artigo, além da construção de *labels* e *features*, também é discutida a estimativa das capacidades de marcar e conceder gols, utilizadas aqui, então esta seção do artigo também é correspondente ao que foi implementado nos treinos.

[20]:
```
import xgboost as xgb
import sklearn.metrics as mt
```

[21]:
```
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
```

[22]:
```
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?

Dois modelos diferentes foram treinados pois um deles se refere à um modelo para classificar a probabilidade de se marcar um gol dada uma determinada ação e outro para classificar a probabilidade de se conceder um gol dada determinada ação. É importante ressaltar como essas duas probabilidades não são complementares, ou seja, a probabilidade de se marcar um gol não é igual a um menos a probabilidade de se conceder um gol.

Por esse motivo, podemos ver uma performance distinta para os classificadores. Ademais, podemos ver que o *concedes* teve uma performance superior por que NAO SEI

1.3.6 Predictions

Neste subtítulo, são utilizados os modelos construídos previamente para calcular as probabilidades de se marcar ou conceder gols para cada estado de jogo desejado. O cálculo dessas probabilidades é utilizado para o cálculo do valor das ações dos jogadores. No artigo, o uso e cálculo das probabilidades de se marcar um gol estão descritas principalmente nas seções 3 e 4, além da seção 5, onde são analisados os experimentos feitos utilizadno essa métrica.

[23]:
```
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
```

[24]:
```
preds = {}
preds["England"] = generate_predictions(features=features["England"],␣
 ↪models=models)
preds["England"]
```

[24]:

	scores	concedes
0	0.002992	0.000412
1	0.003928	0.000329
2	0.002779	0.000345
3	0.002234	0.000298
4	0.005827	0.000308
…	…	…
482896	0.076417	0.001592
482897	0.023226	0.003552

```
482898  0.005620  0.068251
482899  0.082877  0.003011
482900  0.034658  0.003071

[482901 rows x 2 columns]
```

1.3.7 Action Values

Nesta parte do código, a biblioteca *socceraction* é utilizada mais uma vez para calcular o valor de cada uma das ações desejadas, fazendo uso das predições de probabilidade feitas previamente. As definições deste cálculo estão descritos na seção 3 do artigo, mais especificamente na seção 3.1.

[25]:
```
import socceraction.vaep.formula as fm
```

[26]:
```
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", "player_id", "action_id", "game_id",␣
 ↪"start_x", "start_y", "end_x", "end_y", "type_name", "result_name"]],
        predictions.rename(columns={"scores": "Pscores", "concedes":␣
 ↪"Pconcedes"}),
        action_values
    ], axis=1)

    return action_values
```

[27]:
```
action_values = {}
action_values["England"] = calculate_action_values(spadl=spadl["England"],␣
 ↪predictions=preds["England"])
action_values["England"]
```

[27]:
```
        original_event_id  player_id  action_id  game_id  start_x  start_y  \
0             177959171.0      25413          0  2499719    51.45    34.68
1             177959172.0     370224          1  2499719    32.55    14.96
2             177959173.0       3319          2  2499719    53.55    17.00
3             177959174.0     120339          3  2499719    36.75    19.72
4             177959175.0     167145          4  2499719    43.05     3.40
…                     …          …          …        …        …        …
482896        251596226.0      20620       1139  2500098    55.65     7.48
482897        251596229.0      14703       1140  2500098   103.95    19.04
482898        251596408.0       8239       1141  2500098     2.10    46.92
482899        251596232.0      70965       1142  2500098   105.00     0.00
482900        251596236.0       8005       1143  2500098    90.30    34.00

         end_x  end_y       type_name result_name   Pscores  Pconcedes  \
0        32.55  14.96            pass     success  0.002992   0.000412
```

```
1         53.55  17.00            pass     success  0.003928   0.000329
2         36.75  19.72            pass     success  0.002779   0.000345
3         43.05   3.40            pass     success  0.002234   0.000298
4         75.60   8.16            pass     success  0.005827   0.000308
…           …    …                …          …        …          …
482896   103.95  19.04            pass     success  0.076417   0.001592
482897   103.95  19.04           cross        fail  0.023226   0.003552
482898     0.00  46.24    interception     success  0.005620   0.068251
482899    92.40  36.04  corner_crossed     success  0.082877   0.003011
482900   105.00  27.20            shot        fail  0.034658   0.003071

        offensive_value  defensive_value  vaep_value
0              0.000000        -0.000000    0.000000
1              0.000935         0.000083    0.001018
2             -0.001149        -0.000016   -0.001164
3             -0.000545         0.000047   -0.000498
4              0.003593        -0.000010    0.003583
…                 …                …          …
482896         0.066197         0.000381    0.066578
482897        -0.053191        -0.001960   -0.055151
482898         0.002068        -0.045026   -0.042958
482899         0.036377        -0.003011    0.033366
482900        -0.048219        -0.000061   -0.048279

[482901 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?

[28]:
```
action_values["England"].query("Pscores > 0.95").head(10)
```

[28]:
```
      original_event_id  player_id  action_id  game_id  start_x  start_y  \
34          177959212.0      25413         34  2499719    92.40    40.12
60          177959280.0      14763         60  2499719   100.80    32.64
421         177959759.0      12829        421  2499719    98.70    31.28
677         177960130.0       7945        677  2499719    96.60    34.00
820         177960379.0      12829        820  2499719    96.60    31.28
1157        177960849.0       7870       1157  2499719    98.70    25.16
1187        177960902.0      26010       1187  2499719    95.55    38.08
2259        178148575.0       8325        971  2499720    93.45    30.60
2879        178122511.0       9127        315  2499721    91.35    32.64
3151        178122911.0       8433        587  2499721    95.55    51.68

      end_x  end_y type_name result_name   Pscores  Pconcedes  \
34    105.0   37.4      shot     success  0.978621   0.001997
60    105.0   34.0      shot     success  0.987080   0.003912
```

```
421   105.0   34.0      shot     success  0.982079   0.002473
677   105.0   34.0      shot     success  0.985336   0.005580
820   105.0   37.4      shot     success  0.982742   0.002357
1157  105.0   37.4      shot     success  0.985336   0.003204
1187  105.0   37.4      shot     success  0.984085   0.002321
2259  105.0   34.0      shot     success  0.984568   0.002237
2879  105.0   37.4      shot     success  0.980534   0.001895
3151  105.0   30.6      shot     success  0.984178   0.001700

      offensive_value  defensive_value  vaep_value
34           0.906938        -0.001233    0.905705
60           0.781600        -0.000663    0.780937
421          0.846244        -0.000405    0.845839
677          0.851441         0.002292    0.853733
820          0.901592         0.001012    0.902604
1157         0.881998         0.001642    0.883640
1187         0.884413         0.000996    0.885409
2259         0.899141        -0.000248    0.898893
2879         0.911810         0.000018    0.911828
3151         0.919819         0.000284    0.920103
```

[40]:
```
action_values["England"].query("Pscores > 0.95").result_name.unique()
```

[40]:
```
array(['success'], dtype=object)
```

[29]:
```
action_values["England"].query("Pscores > 0.95").describe()[["start_x",␣
↪"start_y"]]
```

[29]:
```
          start_x     start_y
count  914.000000  914.000000
mean    94.768818   34.364551
std      6.172598    6.644241
min     46.200000    7.480000
25%     92.400000   29.920000
50%     95.550000   34.000000
75%     98.700000   38.760000
max    105.000000   58.480000
```

[30]:
```
action_values["England"].query("Pscores > 0.95").type_name.unique()
```

[30]:
```
array(['shot'], dtype=object)
```

Ao filtrar o *DataFrame* para analisar apenas as ações com *Pscores* maior que 0.95, podemos ver que a esmagadora maioria dessas ações ocorreram em pontos no campo muito próximos ao gol adversário. Mais especificamente, se olharmos a descrição dos valores de *start_x* e *start_y*, vamos notar que a média deles para estas ações é de aproximadamente 94.7 e 34.3, respectivamente, com um desvio padrão de aproximadamente 6 para cada um. Considerando que o centro do gol

adversário está na coordenada (105, 68), nota-se que as ações analisadas ocorreram muito próximas ao gol, o que explica o por quê do valor de *Pscores* para elas ser tão alto. Seguindo a mesma linha de raciocínio, nota-se que TODAS as ações em questão foram do tipo chute, o que mais uma vez não é nenhuma surpresa, dado a posição em que ocorreram, além do fato de que TODAS elas tiveram como resultado sucesso, isto é, o gol foi convertido.

[39]:
```
action_values["England"].query("type_name == 'shot' and Pscores < 0.95").
↪head(10)
```

[39]:

	original_event_id	player_id	action_id	game_id	start_x	start_y	\
40	177959247.0	26150	40	2499719	89.25	32.64	
87	177959289.0	7868	87	2499719	85.05	45.56	
175	177959429.0	7868	175	2499719	78.75	47.60	
288	177959606.0	7945	288	2499719	94.50	41.48	
293	177959611.0	49876	293	2499719	72.45	43.52	
370	177959714.0	7945	370	2499719	90.30	28.56	
388	177959740.0	14869	388	2499719	94.50	46.24	
392	177959749.0	120339	392	2499719	76.65	43.52	
488	177959853.0	14763	488	2499719	96.60	28.56	
550	177959948.0	49876	550	2499719	75.60	40.80	

	end_x	end_y	type_name	result_name	Pscores	Pconcedes	\
40	105.00	40.80	shot	fail	0.022286	0.008598	
87	105.00	40.80	shot	fail	0.015082	0.006453	
175	105.00	37.40	shot	fail	0.018859	0.004449	
288	94.50	41.48	shot	fail	0.028141	0.002925	
293	72.45	43.52	shot	fail	0.006075	0.002743	
370	90.30	28.56	shot	fail	0.025559	0.004138	
388	105.00	37.40	shot	fail	0.041145	0.003901	
392	105.00	27.20	shot	fail	0.013025	0.005055	
488	105.00	27.20	shot	fail	0.019120	0.003561	
550	75.60	40.80	shot	fail	0.010539	0.005474	

	offensive_value	defensive_value	vaep_value
40	-0.064444	-0.006484	-0.070928
87	-0.014264	-0.004870	-0.019134
175	-0.034156	-0.001490	-0.035646
288	-0.083707	-0.001427	-0.085134
293	-0.010863	-0.001276	-0.012139
370	-0.062123	-0.001989	-0.064112
388	-0.065420	-0.001781	-0.067201
392	-0.008849	-0.001421	-0.010271
488	-0.090871	-0.001693	-0.092564
550	-0.007604	-0.002328	-0.009932

[32]:
```
action_values["England"].query("type_name == 'shot' and Pscores < 0.95").
↪describe()[["start_x", "start_y"]]
```

[32]:

	start_x	start_y
count	7537.00000	7537.000000
mean	88.51011	34.806040
std	8.12663	9.262053
min	43.05000	0.680000
25%	81.90000	27.880000
50%	90.30000	34.680000
75%	94.50000	42.160000
max	103.95000	63.240000

[41]:
```
action_values["England"].query("type_name == 'shot' and Pscores < 0.95").
↪result_name.unique()
```

[41]:
```
array(['fail'], dtype=object)
```

Podemos ver que ações de chute com *PScores* menor do que 0.95 são ações que ocorreram em pontos no campo também proximos ao gol adversário, mas não tanto quanto os vistos anteriormente. Além disso, nota-se que TODOS os chutes em questão tiveram como resultado uma falha, isto é, o gol não foi convertido.

Eu diria que o modelo aprende sim a lidar com essa combinação de tipo de ação. Fica claro que o modelo separa diretamente as probabilidades de marcar gol para as ações de chute no ponto de 95%. Isto é, a variável y que queremos prever, a probabilidade de se marcar o gol, está diretamente ligada ao tipo de ação chute e o resultado deste tipo de ação. Quando o chute resulta em gol (resultado igual a sucesso), a probabilidade de se marcar um gol calculado para aquela ação é de 95% ou mais. A situação oposta ocorre para chutes que não resultaram em gol. Isso ocorre devido ao fato de que uma forma de *target leakage* está acontecendo, uma vez que a coluna de *result_name* está intimamente ligada com a variável que queremos prever, isto é, a probabilidade de se marcar o gol com aquela ação em questão.

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

A coluna *offensive_value* se refere à equação (1) descrita na seção 3.1 do artigo. A coluna *defensive_value*, por sua vez, se refere à equação (2) da mesma sessão, só que sua negação, devido ao fato de que toda ação deve tentar decrementar a probabilidade de se conceder um gol.

1.3.8 Player Ratings

Nese subtítulo, é calculado o *rating* de cada jogador com base nos valores de ações encontrados anteriormente. Como descrito na seção 3.2 no artigo, podemos encontra uma soma dos valores de todas as ações dos jogadores, mas é mais interessante encontrar um valor de *rating* para cada 90 minutos (partida) jogadas. Na seção 5, de maneira geral, a fórmula de *rating* proposta é utilizada para caracterizar diversos jogadores e tais resultados analisados como forma de estudo.

[34]:
```
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
```

[35]:
```
minutes_per_season = {}
minutes_per_season["England"] = calculate_minutes_per_season(minutes["England"])
minutes_per_season["England"]
```

[35]:
```
     player_id  minutes_played
0           36            1238
1           38             382
2           48            3343
3           54            3348
4           56             266
..         …              …
510     448708              21
511     450826              35
512     486252             649
513     531655              28
514     532949               1

[515 rows x 2 columns]
```

[36]:
```
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
```

[37]:
```
player_ratings = {}
player_ratings["England"] =␣
 ↪calculate_player_ratings(action_values=action_values["England"],␣
 ↪minutes_per_season=minutes_per_season["England"], players=players)
player_ratings["England"].head(15)
```

[37]:
```
    player_id                      player_name  minutes_played  vaep_total  \
0      120353              Mohamed  Salah Ghaly          2995.0   28.516333
1        3802          Philippe Coutinho Correia          1134.0    8.896437
2        8325  Sergio Leonel Agüero del Castillo          2038.0   14.206033
```

3	8717	Harry Kane	3201.0	20.985924
4	25867	Pierre-Emerick Aubameyang	1098.0	7.095187
5	25707	Eden Hazard	2504.0	16.074993
6	8249	Marouane Fellaini-Bakkioui	693.0	4.240451
7	26150	Riyad Mahrez	3063.0	18.054569
8	8317	David Josué Jiménez Silva	2519.0	13.998504
9	3319	Mesut Özil	2253.0	12.424434
10	11066	Raheem Shaquille Sterling	2697.0	14.583854
11	14911	Heung-Min Son	2383.0	12.866082
12	134513	Anthony Martial	1650.0	8.820550
13	38021	Kevin De Bruyne	3190.0	16.754573
14	54	Christian Dannemann Eriksen	3348.0	17.168349

	vaep_p90
0	0.856918
1	0.706066
2	0.627352
3	0.590045
4	0.581573
5	0.577775
6	0.550708
7	0.530497
8	0.500145
9	0.496316
10	0.486669
11	0.485920
12	0.481121
13	0.472700
14	0.461515

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?

O top 5 gerada é definitivamente bem representativo. Todos os 5 são jogadores excepcionais que foram/são grandes referências para suas equipes.

É interessante analisar como, no entanto, o top 5 gerado com a VAEP é totalmente diferente do top 5 gerado na atividade 4 com a métrica de xT. Isso no entanto deve ser visto como uma diferença técnica entre as métricas, isto é, como elas analisam/interpretam lances de maneiras diferentes, e não necessariamente ruins/erradas. Não existe uma verdade absoluta. No artigo https://dtai.cs.kuleuven.be/sports/blog/valuing-on-the-ball-actions-in-soccer-a-critical-comparison-of-xt-and-vaep/ , a diferença entre as duas métricas é analisada de forma ainda mais concisa.

Acredito, no entanto, que a VAEP traga informações mais completas acerca da performance dos jogadores, devido a sua definição que engloba mais variáveis. Dessa maneira, a VAEP seria um top 5 mais representativo. Me surpreende, no entanto, o nome de Kevin De Bruyne aparecer apenas na 14º posição. O top 15 de forma geral contêm jogadores que poderiam facilmente estar entre os 5 com as maiores pontuações, mas pelo meu ponto de vista De bruyne é um dos mais mereciam

estar. De qualquer forma, o top 5 gerado é sim muito representativo, e a métrica da VAEP propôs uma excelente forma de ranquear os jogadores da Premier League.