socceraction

Release 1.5.1

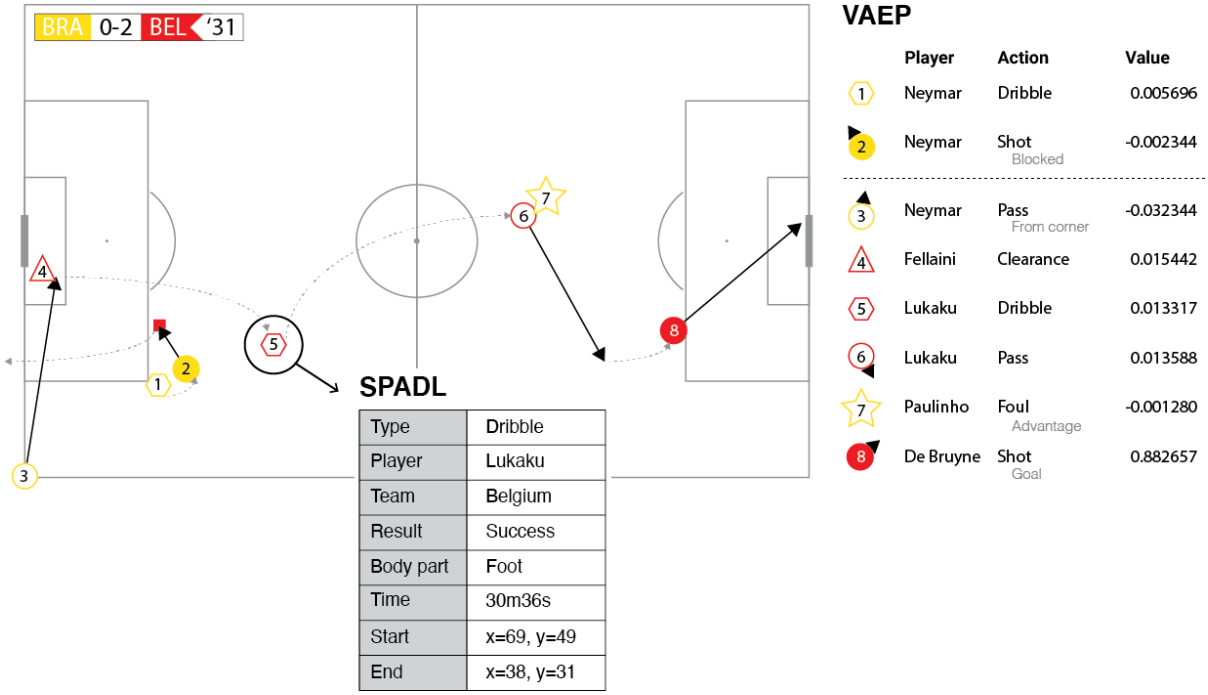
Tom Decroos

Mar 07, 2024

DOCUMENTATION

socceraction is a Python package for objectively quantifying the value of the individual actions performed by soccer players using event stream data. It contains the following components:

- A set of API clients for **loading event stream data** from StatsBomb, Wyscout and Opta.
- Converters for each of these provider's proprietary data format to the **SPADL** and **atomic-SPADL** formats, which are unified and expressive languages for on-the-ball player actions.
- An implementation of the **Expected Threat (xT)** possession value framework.
- An implementation of the **VAEP** and **Atomic-VAEP** possession value frameworks.



CHAPTER

ONE

QUICKSTART

Eager to get started valuing some soccer actions? This page gives a quick introduction on how to get started.

1.1 Installation

First, make sure that socceraction is installed:

```
$ pip install socceraction[statsbomb]
```

For detailed instructions and other installation options, check out our detailed *installation instructions*.

1.2 Loading event stream data

First of all, you will need some data. Luckily, both StatsBomb and Wyscout provide a small freely available dataset. The *data module* of socceraction makes it trivial to load these datasets as Pandas DataFrames. In this short introduction, we will work with Statsbomb's dataset of the 2018 World Cup.

```
import pandas as pd
from socceraction.data.statsbomb import StatsBombLoader

# Set up the StatsBomb data loader
SBL = StatsBombLoader()

# View all available competitions
df_competitions = SBL.competitions()

# Create a dataframe with all games from the 2018 World Cup
df_games = SBL.games(competition_id=43, season_id=3).set_index("game_id")
```

Note: Keep in mind that by using the public StatsBomb data you are agreeing to their user agreement.

For each game, you can then retrieve a dataframe containing the teams, all players that participated, and all events that were recorded in that game. Specifically, we'll load the data from the third place play-off game between England and Belgium.

```
game_id = 8657
df_teams = SBL.teams(game_id)
```

(continues on next page)

(continued from previous page)

```
df_players = SBL.players(game_id)
df_events = SBL.events(game_id)
```

1.3 Converting to SPADL actions

The event stream format is not well-suited for data analysis: some of the recorded information is irrelevant for valuing actions, each vendor uses their own custom format and definitions, and the events are stored as unstructured JSON objects. Therefore, socceraction uses the *SPADL format* for describing actions on the pitch. With the code below, you can convert the events to SPADL actions.

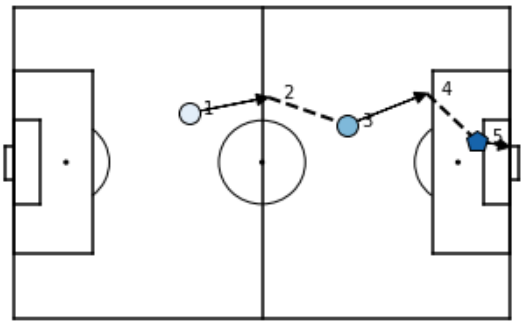
```
import socceraction.spadl as spadl

home_team_id = df_games.at[game_id, "home_team_id"]
df_actions = spadl.statsbomb.convert_to_actions(df_events, home_team_id)
```

With the matplotsoccer package, you can try plotting some of these actions:

```
import matplotsoccer as mps

# Select relevant actions
df_actions_goal = df_actions.loc[2196:2200]
# Replace result, actiontype and bodypart IDs by their corresponding name
df_actions_goal = spadl.add_names(df_actions_goal)
# Add team and player names
df_actions_goal = df_actions_goal.merge(df_teams).merge(df_players)
# Create the plot
mps.actions(
    location=df_actions_goal[["start_x", "start_y", "end_x", "end_y"]],
    action_type=df_actions_goal.type_name,
    team=df_actions_goal.team_name,
    result=df_actions_goal.result_name == "success",
    label=df_actions_goal[["time_seconds", "type_name", "player_name", "team_name"]],
    labeltitle=["time", "actiontype", "player", "team"],
    zoom=False
)
```



	time	actiontype	player	team
1	81m19s	pass	Axel Witsel	Belgium
2	81m21s	dribble	Kevin De Bruyne	Belgium
3	81m24s	pass	Kevin De Bruyne	Belgium
4	81m25s	dribble	Eden Hazard	Belgium
5	81m27s	shot	Eden Hazard	Belgium

1.4 Valuing actions

We can now assign a numeric value to each of these individual actions that quantifies how much the action contributed towards winning the game. Socceraction implements three frameworks for doing this: xT, VAEP and Atomic-Vaep. In this quickstart guide, we will focus on the xT framework.

The expected threat or xT model overlays a $M \times N$ grid on the pitch in order to divide it into zones. Each zone z is then assigned a value $xT(z)$ that reflects how threatening teams are at that location, in terms of scoring. An example grid is visualized below.

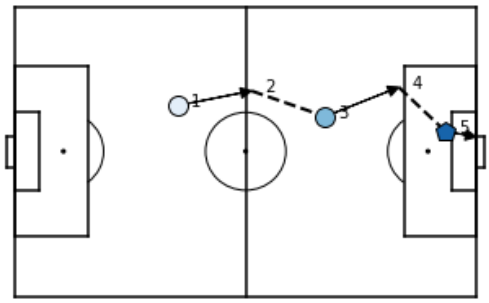
The code below allows you to load league-wide xT values from the 2017-18 Premier League season (the 12x8 grid shown above). Instructions on how to train your own model can be found in the *detailed documentation about xT*.

```
import socceraction.xthreat as xthreat

url_grid = "https://karun.in/blog/data/open_xt_12x8_v1.json"
xT_model = xthreat.load_model(url_grid)
```

Subsequently, the model can be used to value actions that successfully move the ball between two zones by computing the difference between the threat value on the start and end location of each action. The xT framework does not assign a value to failed actions, shots and defensive actions such as tackles.

```
df_actions_ltr = spadl.play_left_to_right(df_actions, home_team_id)
df_actions["xT_value"] = xT_model.rate(df_actions_ltr)
```



	time	actiontype	player	team	xT
1	2179.0	pass	Axel Witsel	Belgium	0.0042
2	2181.0	dribble	Kevin De Bruyne	Belgium	0.0070
3	2184.0	pass	Kevin De Bruyne	Belgium	0.0047
4	2185.0	dribble	Eden Hazard	Belgium	0.2289
5	2187.0	shot	Eden Hazard	Belgium	nan

Ready for more? Check out the detailed documentation about the *data representation* and *action value frameworks*.

CHAPTER

TWO

INSTALLATION

Before you can use socceraction, you'll need to get it installed. This guide will guide you to a minimal installation that'll work while you walk through the introduction.

2.1 Install Python

Being a Python library, socceraction requires Python. Currently, socceraction supports Python version 3.9 – 3.11. Get the latest version of Python at https://www.python.org/downloads/ or with your operating system's package manager.

You can verify that Python is installed by typing `python` from your shell; you should see something like:

```
Python 3.x.y
[GCC 4.x] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2.2 Install socceraction

You've got two options to install socceraction.

2.2.1 Installing an official release with `pip`

This is the recommended way to install socceraction. Simply run this simple command in your terminal of choice:

```
$ python -m pip install socceraction
```

You might have to install pip first. The easiest method is to use the standalone pip installer.

2.2.2 Installing the development version

Socceraction is actively developed on GitHub, where the code is always available. You can easily install the development version with:

```
$ pip install git+https://github.com/ML-KULeuven/socceraction.git
```

However, to be able to make modifications in the code, you should either clone the public repository:

```
$ git clone git://github.com/ML-KULeuven/socceraction.git
```

Or, download the zipball:

```
$ curl -OL https://github.com/ML-KULeuven/socceraction/archive/master.zip
```

Once you have a copy of the source, you can embed it in your own Python package, or install it into your site-packages easily:

```
$ cd socceraction
$ python -m pip install -e .
```

2.3 Verifying

To verify that socceraction can be seen by Python, type `python` from your shell. Then at the Python prompt, try to import socceraction:

```
>>> import socceraction
>>> print(socceraction.__version__)
```

CHAPTER

THREE

LOADING DATA

Socceraction provides API clients for various popular event stream data sources. These clients enable fetching event streams and their corresponding metadata as Pandas DataFrames using a unified data model. Alternatively, you can also use kloppy to load data.

3.1 Loading data with socceraction

All API clients implemented in socceraction inherit from the *EventDataLoader* interface. This interface provides the following methods to retrieve data as a Pandas DataFrames with a unified data model (i.e., `Schema`). The schema defines the minimal set of columns and their types that are returned by each method. Implementations of the *EventDataLoader* interface may add additional columns.

Method	Output schema	Description
competitions()	*CompetitionSchema*	All available competitions and seasons
games(competition_id, season_id)	*GameSchema*	All available games in a season
teams(game_id)	*TeamSchema*	Both teams that participated in a game
players(game_id)	*PlayerSchema*	All players that participated in a game
events(game_id)	*EventSchema*	The event stream of a game

Currently, the following data providers are supported:

3.1.1 Loading StatsBomb data

The *StatsBombLoader* class provides an API client enabling you to fetch StatsBomb event stream data as Pandas DataFrames. This document provides an overview of the available data sources and how to access them.

Setup

To be able to load StatsBomb data, you'll first need to install a few additional dependencies which are not included in the default installation of socceraction. You can install these additional dependencies by running:

```
$ pip install "socceraction[statsbomb]"
```

Connecting to a data store

First, you have to create a *StatsBombLoader* object and configure it for the data store you want to use. The *StatsBombLoader* supports loading data from the StatsBomb Open Data repository, from the official StatsBomb API, and from local files.

Open Data repository

StatsBomb has made event stream data of certain leagues freely available for public non-commercial use at https://github.com/statsbomb/open-data. This open data can be accessed without the need of authentication, but its use is subject to a user agreement. The code below shows how to setup an API client that can fetch data from the repository.

```
# optional: suppress warning about missing authentication
import warnings
from statsbombpy.api_client import NoAuthWarning
warnings.simplefilter('ignore', NoAuthWarning)

from socceraction.data.statsbomb import StatsBombLoader

api = StatsBombLoader(getter="remote", creds=None)
```

Note: If you publish, share or distribute any research, analysis or insights based on this data, StatsBomb requires you to state the data source as StatsBomb and use their logo.

StatsBomb API

API access is for paying customers only. Authentication can be done by setting environment variables named `SB_USERNAME` and `SB_PASSWORD` to your login credentials. Alternatively, the constructor accepts an argument `creds` to pass your login credentials in the format `{"user": "", "passwd": ""}`.

```
from socceraction.data.statsbomb import StatsBombLoader

# set authentication credentials as environment variables
import os
os.environ["SB_USERNAME"] = "your_username"
os.environ["SB_PASSWORD"] = "your_password"
api = StatsBombLoader(getter="remote")

# or provide authentication credentials as a dictionary
api = StatsBombLoader(getter="remote", creds={"user": "", "passwd": ""})
```

Local directory

A final option is to load data from a local directory. This local directory can be specified by passing the `root` argument to the constructor, specifying the path to the local data directory.

```
from socceraction.data.statsbomb import StatsBombLoader

api = StatsBombLoader(getter="local", root="data/statsbomb")
```

Note that the data should be organized in the same way as the StatsBomb Open Data repository, which corresponds to the following file hierarchy:

```
root
├── competitions.json
├── events
│   ├── <match_id>.json
│   ├── ...
│   └── ...
├── lineups
│   ├── <match_id>.json
│   └── ...
├── matches
│   ├── <competition_id>
│   │   └── <season_id>.json
│   │   └── ...
│   └── ...
└── three-sixty
    ├── <match_id>.json
    └── ...
```

Loading data

Next, you can load the match event stream data and metadata by calling the corresponding methods on the *StatsBombLoader* object.

StatsBombLoader.competitions()

```
df_competitions = api.competitions()
```

season_id	competition_id	competition_name	country_name	competition_gender	season_name
106	43	FIFA World Cup	International	male	2022
30	72	Women's World Cup	International	female	2019
3	43	FIFA World Cup	International	male	2018

StatsBombLoader.games()

```
df_games = api.games(competition_id=43, season_id=3)
```

game	season_id	competition_id	competition_stage	game_	game_date	home_t	away_te	home_	away_	venue	referee_id
8658	3	43	Final	7	2018-07-15 17:00:00	771	785	4	2	Stadion Luzhniki	730
8657	3	43	3rd Place Final	7	2018-07-14 16:00:00	782	768	2	0	Saint-Petersburg Stadium	741

StatsBombLoader.teams()

```
df_teams = api.teams(game_id=8658)
```

team_id	team_name
771	France
785	Croatia

StatsBombLoader.players()

```
df_players = api.players(game_id=8658)
```

game_	team_	player_	player_name	nickname	jersey_numb	is_star	starting_position	starting_position_n	minutes_played
8658	771	3009	Kylian Mbappé Lottin	Kylian Mbappé	10	True	12	Right Midfield	95
8658	785	5463	Luka Modrić		10	True	13	Right Center Midfield	95

StatsBombLoader.events()

```
df_events = api.events(game_id=8658)
```

ev	in- de	pe rio	tim tar	mi	se on	typ	typ	po se sio	po se sio	po se sio	pla	pla	tea	tea	du ra- tio	ex- tra	re- lat	pla	pla	po si- tio	po si- tio	lo- ca- tio	un de	co ter pre	game_id ure
47(fd4 46: b4! cff	1	1	19(01- 01	0	0	35	Sta ing XI	1	77	Fra	1	Re u- lar Pla	77	Fra	0.0	{..	[]						Fal	Fal	8658
0c(56 45 9b 7c	2	1	19(01- 01	0	0	35	Sta ing XI	1	77	Fra	1	Re u- lar Pla	78	Cr tia	1.4	{..	[]						Fal	Fal	8658
c5 efe 48(9cf 16(	3	1	19(01- 01	0	0	18	Ha Sta	1	77	Fra	1	Re u- lar Pla	77	Fra	0.0	{}	['7 c57 40 8cc cec						Fal	Fal	8658

If 360 data snapshots are available for the game, they can be loaded by passing `load_360=True` to the `events()` method. This will add two columns to the events dataframe: `visible_area_360` and `freeze_frame_360`. The former contains the visible area of the pitch in the 360 snapshot, while the latter contains the player locations in the 360 snapshot.

```
df_events = api.events(game_id=3788741, load_360=True)
```

3.1.2 Loading Wyscout data

The *`WyscoutLoader`* class provides an API client enabling you to fetch Wyscout event stream data as Pandas DataFrames. This document provides an overview of the available data sources and how to access them.

Note: Currently, only version 2 of the Wyscout API is supported. See https://github.com/ML-KULeuven/socceraction/issues/156 for progress on version 3 support.

Connecting to a data store

First, you have to create a *WyscoutLoader* object and configure it for the data store you want to use. The *WyscoutLoader* supports loading data from the official Wyscout API and from local files. Additionally, the *PublicWyscoutLoader* class can be used to load a publicly available dataset.

Wyscout API

Wyscout API access requires a separate subscription. Wyscout currently offers three different packs: a Database Pack (match sheet data), a Stats Pack (statistics derived from match event data), and an Events Pack (raw match event data). A subscription to the Events Pack is required to access the event stream data.

Authentication can be done by setting environment variables named `WY_USERNAME` and `WY_PASSWORD` to your login credentials (i.e., client id and secret). Alternatively, the constructor accepts an argument `creds` to pass your login credentials in the format `{"user": "", "passwd": ""}`.

```
from socceraction.data.wyscout import WyscoutLoader

# set authentication credentials as environment variables
import os
os.environ["WY_USERNAME"] = "your_client_id"
os.environ["WY_PASSWORD"] = "your_secret"
api = WyscoutLoader(getter="remote")

# or provide authentication credentials as a dictionary
api = WyscoutLoader(getter="remote", creds={"user": "", "passwd": ""})
```

Local directory

Data can also be loaded from a local directory. This local directory can be specified by passing the `root` argument to the constructor, specifying the path to the local data directory.

```
from socceraction.data.wyscout import WyscoutLoader

ap = WyscoutLoader(getter="local", root="data/wyscout")
```

The loader uses the directory structure and file names to determine which files should be parsed to retrieve the requested data. Therefore, the local directory should have a predefined file hierarchy. By default, it expects following file hierarchy:

```
root
├── competitions.json
├── seasons_<competition_id>.json
├── matches_<season_id>.json
└── matches
    ├── events_<game_id>.json
    └── ...
```

If your local directory has a different file hierarchy, you can specify this custom hierarchy by passing the `feeds` argument to the constructor. A wide range of file names and directory structures are supported. However, the competition, season, and game identifiers must be included in the file names to be able to locate the corresponding files for each entity.

```
from socceraction.data.wyscout import WyscoutLoader

ap = WyscoutLoader(getter="local", root="data/wyscout", feeds={
  "competitions": "competitions.json",
  "seasons": "seasons_{competition_id}.json",
  "games": "matches_{season_id}.json",
  "events": "matches/events_{game_id}.json",
})
```

The {competition_id}, {season_id}, and {game_id} placeholders will be replaced by the corresponding id values when data is retrieved.

Soccer logs dataset

As part of the "A public data set of spatio-temporal match events in soccer competitions" paper, Wyscout made an event stream dataset available for research purposes. The dataset covers the 2017/18 season of the Spanish, Italian, English, German, and French first division. In addition, it includes the data of the 2018 World Cup and the 2016 European championship. The dataset is available at https://figshare.com/collections/Soccer_match_event_dataset/4415000/2.

As the format of this dataset is slightly different from the format of the official Wyscout API, a separate *PublicWyscoutLoader* class is provided to load this dataset. This loader will download the dataset once and extract it to the specified root directory.

```
from socceraction.data.wyscout import PublicWyscoutLoader

api = PublicWyscoutLoader(root="data/wyscout")
```

Loading data

Next, you can load the match event stream data and metadata by calling the corresponding methods on the *WyscoutLoader* object.

- *WyscoutLoader.competitions()*
- *WyscoutLoader.games()*
- *WyscoutLoader.teams()*
- *WyscoutLoader.players()*
- *WyscoutLoader.events()*

3.1.3 Loading Opta data

Opta's event stream data comes in many different flavours. The *OptaLoader* class provides an API client enabling you to fetch data from the following data feeds as Pandas DataFrames:

- Opta F1, F9 and F24 JSON feeds
- Opta F7 and F24 XML feeds
- StatsPerform MA1 and MA3 JSON feeds
- WhoScored.com JSON data

Currently, only loading data from local files is supported.

Connecting to a data store

First, you have to create a *OptaLoader* object and configure it for the data feeds you want to use.

Generic setup

To set up a *OptaLoader* you have to specify the root directory, the filename hierarchy of the feeds and a parser for each feed. For example:

```
from socceraction.data.opta import OptaLoader, parsers

api = OptaLoader(
  root="data/opta",
  feeds = {
      "f7": "f7-{competition_id}-{season_id}-{game_id}.xml",
      "f24": "f24-{competition_id}-{season_id}-{game_id}.xml",
  }
  parser={
      "f7": parsers.F7XMLParser,
      "f24": parsers.F24XMLParser
  }
)
```

Since the loader uses the directory structure and file names to determine which files should be parsed, the root directory should have a predefined file hierarchy defined in the `feeds` argument. A wide range of file names and directory structures are supported. However, the competition, season, and game identifiers must be included in the file names to be able to locate the corresponding files for each entity. For example, you might have grouped feeds by competition and season as follows:

```
root
├── competition_<competition_id>
│   ├── season_<season_id>
│   │   ├── f7_<game_id>.xml
│   │   └── f24_<game_id>.xml
│   └── ...
└── ...
```

In this case, you can use the following feeds configuration:

```
feeds = {
    "f7": "competition_{competition_id}/season_{season_id}/f7_{game_id}.xml",
    "f24": "competition_{competition_id}/season_{season_id}/f24_{game_id}.xml",
}
```

Note: On Windows, the backslash character should be used as a path separator.

Furthermore, a few standard configurations are provided. These are listed below.

Opta F7 and F24 XML feeds

```
from socceraction.data.opta import OptaLoader

api = OptaLoader(root="data/opta", parser="xml")
```

The root directory should have the following structure:

```
root
├── f7-{competition_id}-{season_id}.xml
├── f24-{competition_id}-{season_id}-{game_id}.xml
└── ...
```

Opta F1, F9 and F24 JSON feeds

```
from socceraction.data.opta import OptaLoader

api = OptaLoader(root="data/opta", parser="json")
```

The root directory should have the following structure:

```
root
├── f1-{competition_id}-{season_id}.json
├── f9-{competition_id}-{season_id}.json
├── f24-{competition_id}-{season_id}-{game_id}.json
└── ...
```

StatsPerform MA1 and MA3 JSON feeds

```
from socceraction.data.opta import OptaLoader

api = OptaLoader(root="data/statsperform", parser="statsperform")
```

The root directory should have the following structure:

```
root
├── ma1-{competition_id}-{season_id}.json
├── ma3-{competition_id}-{season_id}-{game_id}.json
└── ...
```

WhoScored

WhoScored.com is a popular website that provides detailed live match statistics. These statistics are compiled from Opta's event feed, which can be scraped from the website's source code using a library such as soccerdata. Once you have downloaded the raw JSON data, you can parse it using the *OptaLoader* with:

```
from socceraction.data.opta import OptaLoader

api = OptaLoader(root="data/whoscored", parser="whoscored")
```

The root directory should have the following structure:

```
root
├── {competition_id}-{season_id}-{game_id}.json
└── ...
```

Alternatively, the soccerdata library provides a wrapper that immediately returns a *OptaLoader* object for a scraped dataset.

```
import soccerdata as sd

# Setup a scraper for the 2021/2022 Premier League season
ws = sd.WhoScored(leagues="ENG-Premier League", seasons=2021)
# Scrape all games and return a OptaLoader object
api = ws.read_events(output_fmt='loader')
```

Warning: Scraping data from WhoScored.com violates their terms of service. Legally, scraping this data is therefore a grey area. If you decide to use this data anyway, this is your own responsibility.

Loading data

Next, you can load the match event stream data and metadata by calling the corresponding methods on the *OptaLoader* object.

- *OptaLoader.competitions()*
- *OptaLoader.games()*
- *OptaLoader.teams()*
- *OptaLoader.players()*
- *OptaLoader.events()*

3.2 Loading data with kloppy

Similarly to socceraction, kloppy implements a unified data model for soccer data. The main differences between kloppy and socceraction are: (1) kloppy supports more data sources (including tracking data), (2) kloppy uses a more flexible object-based data model in contrast to socceraction's dataframe-based model, and (3) kloppy covers a more complete set of events while socceraction focuses on-the-ball events. Thus, we recommend using kloppy if you want to load data from a source that is not supported by socceraction or when your analysis is not limited to on-the-ball events.

The following code snippet shows how to load data from StatsBomb using kloppy:

```
from kloppy import statsbomb

dataset = statsbomb.load_open_data(match_id=8657)
```

Instructions for loading data from other sources can be found in the kloppy documentation.

You can then convert the data to the SPADL format using the *convert_to_actions()* function:

```
from socceraction.spadl.kloppy import convert_to_actions

spadl_actions = convert_to_actions(dataset, game_id=8657)
```

Note: Currently, the data model of kloppy is only complete for StatsBomb data. If you use kloppy to load data from other sources and convert it to the SPADL format, you may lose some information.

CHAPTER

FOUR

DATA REPRESENTATION

Socceraction uses **a tabular action-oriented data format**, as opposed to the formats by commercial vendors that describe events. The distinction is that actions are a subset of events that require a player to perform the action. For example, a passing event is an action, whereas an event signifying the end of the game is not an action. Unlike all other event stream formats, we always store the same attributes for each action. Excluding optional information snippets enables us to store the data in a table and more easily apply automatic analysis tools.

Socceraction implements two versions of this action-oriented data format: *SPADL* and *Atomic-SPADL*.

4.1 SPADL

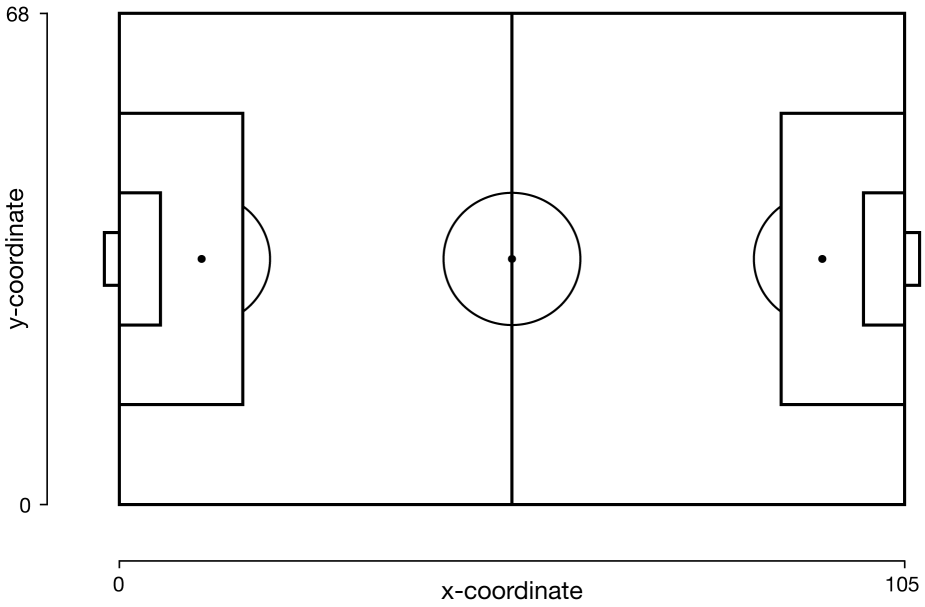
4.1.1 Definitions

SPADL (*Soccer Player Action Description Language*) represents a game as a sequence of on-the-ball actions $[a_1, a_2, ..., a_m]$, where m is the total number of actions that happened in the game. Each action is a tuple of the same twelve attributes:

Attribute	Description
game_id	the ID of the game in which the action was performed
period_id	the ID of the game period in which the action was performed
seconds	the action's start time
player	the player who performed the action
team	the player's team
start_x	the x location where the action started
start_y	the y location where the action started
end_x	the x location where the action ended
end_y	the y location where the action ended
action_type	the type of the action (e.g., pass, shot, dribble)
result	the result of the action (e.g., success or fail)
bodypart	the player's body part used for the action

Start and End Locations

SPADL uses a standardized coordinate system with the origin on the bottom left of the pitch, and a uniform field of 105m x 68m. For direction of play, SPADL uses the "home team attacks to the right" convention, but this can be converted conveniently with the `play_left_to_right()` function such that the lower x-coordinates represent the own half of the team performing the action.



Action Type
: The action type attribute can have 22 possible values. These are *pass*, *cross*, *throw-in*, *crossed free kick*, *short free kick*, *crossed corner*, *short corner*, *take-on*, *foul*, *tackle*, *interception*, *shot*, *penalty shot*, *free kick shot*, *keeper save*, *keeper claim*, *keeper punch*, *keeper pick-up*, *clearance*, *bad touch*, *dribble* and *goal kick*. A detailed definition of each action type is available here.

Result
: The result attribute can either have the value *success*, to indicate that an action achieved it's intended result; or the value *fail*, if this was not the case. An example of a successful action is a pass which reaches a teammate. An example of an unsuccessful action is a pass which goes over the sideline. Some action types can have special results. These are *offside* (for passes, corners and free-kicks), *own goal* (for shots), and *yellow card* and *red card* (for fouls).

Body Part
: The body part attribute can have 4 possible values. These are *foot*, *head*, *other* and *none*. For Wyscout, which does not distinguish between the head and other body parts a special body part *head/other* is used.

All actions, except for some dribbles, are derived from an event in the original event stream data. They can be linked back to the original data by the *original_event_id* attribute. Synthetic dribbles are added to fill gaps between two events. These synthetic dribbles do not have an *original_event_id*.

4.1.2 Example

Socceraction currently implements converters for `StatsBomb`, `Wyscout`, and `Opta` event stream data. We'll use StatsBomb data to illustrate the API, but the API of the other converters is identical.

First, we load the event stream data of the third place play-off in the 2018 FIFA World Cup between Belgium and England.

```
from socceraction.data.statsbomb import StatsBombLoader

SBL = StatsBombLoader()
df_events = SBL.events(game_id=8657)
```

These events can now be converted to SPADL using the *convert_to_actions()* function of the StatsBomb converter.

```
import socceraction.spadl as spadl

df_actions = spadl.statsbomb.convert_to_actions(df_events, home_team_id=777)
```

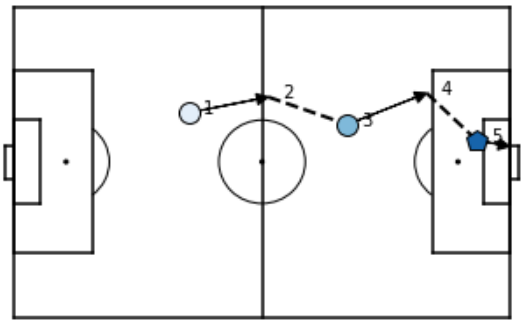
The obtained dataframe represents the body part, result, action type, players and teams with numeric IDs. The code below adds their corresponding names.

```
df_actions = (
  spadl
  .add_names(df_actions)  # add actiontype and result names
  .merge(SBL.teams(game_id=8657))  # add team names
  .merge(SBL.players(game_id=8657))  # add player names
)
```

Below are the five actions in the SPADL format leading up to Belgium's second goal.

game_id	period_id	seconds	team	player	start_x	start_y	end_x	end_y	actiontype	result	bodypart
8657	2	2179	Belgium	Witsel	37.1	44.8	53.8	48.2	pass	success	foot
8657	2	2181	Belgium	De Bruyne	53.8	48.2	70.6	42.2	dribble	success	foot
8657	2	2184	Belgium	De Bruyne	70.6	42.2	87.4	49.1	pass	success	foot
8657	2	2185	Belgium	Hazard	87.4	49.1	97.9	38.7	dribble	success	foot
8657	2	2187	Belgium	Hazard	97.9	38.7	105	37.4	shot	success	foot

Here is the same phase visualized using the `matplotsoccer` package



	time	actiontype	player	team
1	81m19s	pass	Axel Witsel	Belgium
2	81m21s	dribble	Kevin De Bruyne	Belgium
3	81m24s	pass	Kevin De Bruyne	Belgium
4	81m25s	dribble	Eden Hazard	Belgium
5	81m27s	shot	Eden Hazard	Belgium

See also:

This notebook gives an example of the complete pipeline to download public StatsBomb data and convert it to the SPADL format.

4.2 Atomic-SPADL

4.2.1 Definitions

Atomic-SPADL is an alternative version of SPADL which removes the *result* attribute from SPADL and adds a few new action types. Each action is a now a tuple of the following eleven attributes:

Attribute	Description
game_id	the ID of the game in which the action was performed
period_id	the ID of the game period in which the action was performed
seconds	the action's start time
player	the player who performed the action
team	the player's team
x	the x location where the action started
y	the y location where the action started
dx	the distance covered by the action along the x-axis
dy	the distance covered by the action along the y-axis
action_type	the type of the action (e.g., pass, shot, dribble)
bodypart	the player's body part used for the action

In this representation, all actions are *atomic* in the sense that they are always completed successfully without interruption. Consequently, while SPADL treats a pass as one action consisting of both the initiation and receival of the pass, Atomic-SPADL sees giving and receiving a pass as two separate actions. Because not all passes successfully reach a teammate, Atomic-SPADL introduces an *interception* action if the ball was intercepted by the other team or an *out* event if the ball went out of play. Atomic-SPADL similarly divides shots, freekicks, and corners into two separate actions. Practically, the effect is that this representation helps to distinguish the contribution of the player who initiates the action (e.g., gives the pass) and the player who completes the action (e.g., receives the pass).

4.2.2 Example

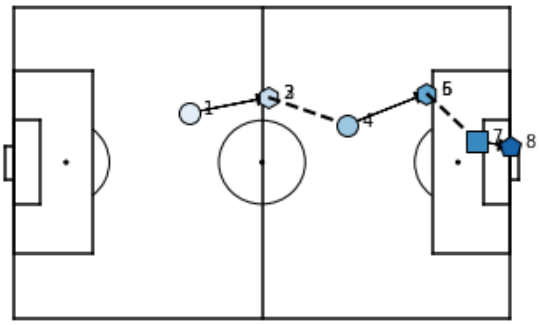
SPADL actions can be converted to their atomic version with the *convert_to_atomic()* function.

```
import socceraction.atomic.spadl as atomicspadl

df_atomic_actions = atomicspadl.convert_to_atomic(df_actions)
```

This is what Belgium's second goal against England in the third place play-off in the 2018 FIFA world cup looks like in the Atomic-SPADL format.

game_id	period_id	seconds	team	player	x	y	dx	dy	actiontype	bodypart
8657	2	2179	Belgium	Witsel	37.1	44.8	0.0	0.0	dribble	foot
8657	2	2179	Belgium	Witsel	37.1	44.8	16.8	3.4	pass	foot
8657	2	2180	Belgium	De Bruyne	53.8	48.2	0.0	0.0	receival	foot
8657	2	2181	Belgium	De Brunne	53.8	48.2	16.8	-6.0	dribble	foot
8657	2	2184	Belgium	De Bruyne	70.6	42.2	16.8	6.9	pass	foot
8657	2	2184	Belgium	Hazard	87.4	49.1	0.0	0.0	receival	foot
8657	2	2185	Belgium	Hazard	87.4	49.1	10.6	-10.3	dribble	foot
8657	2	2187	Belgium	Hazard	97.9	38.7	7.1	-1.4	shot	foot
8657	2	2187	Belgium	Hazard	105.0	37.4	0.0	0.0	goal	foot



	time	actiontype	player_name	team
1	81m19s	pass	Axel Witsel	Belgium
2	81m20s	receival	Kevin De Bruyne	Belgium
3	81m21s	dribble	Kevin De Bruyne	Belgium
4	81m24s	pass	Kevin De Bruyne	Belgium
5	81m24s	receival	Eden Hazard	Belgium
6	81m25s	dribble	Eden Hazard	Belgium
7	81m27s	shot	Eden Hazard	Belgium
8	81m27s	goal	Eden Hazard	Belgium

See also:

This notebook gives an example of the complete pipeline to download public StatsBomb data and convert it to the Atommic-SPADL format.

CHAPTER

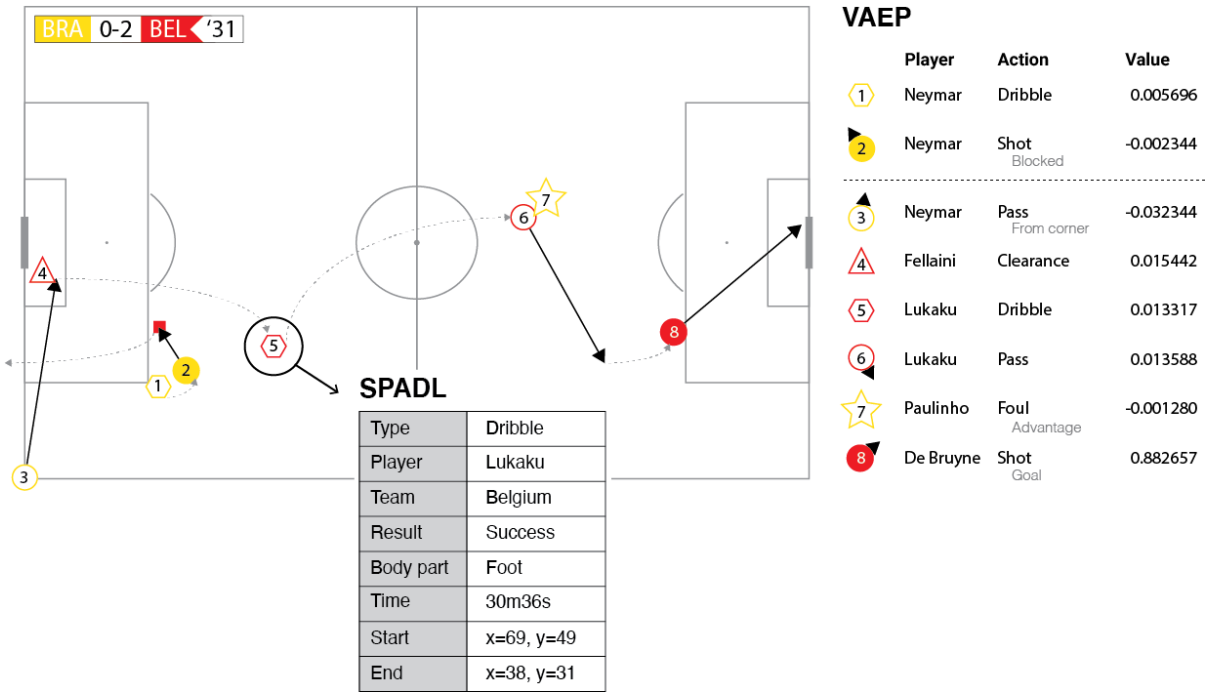
FIVE

VALUING ACTIONS

Once you've *collected the data* and *converted it to the SPADL format*, you can start valuing the contributions of soccer players. This document gives a general introduction to action-valuing frameworks and links to a detailed discussion of the three implemented frameworks.

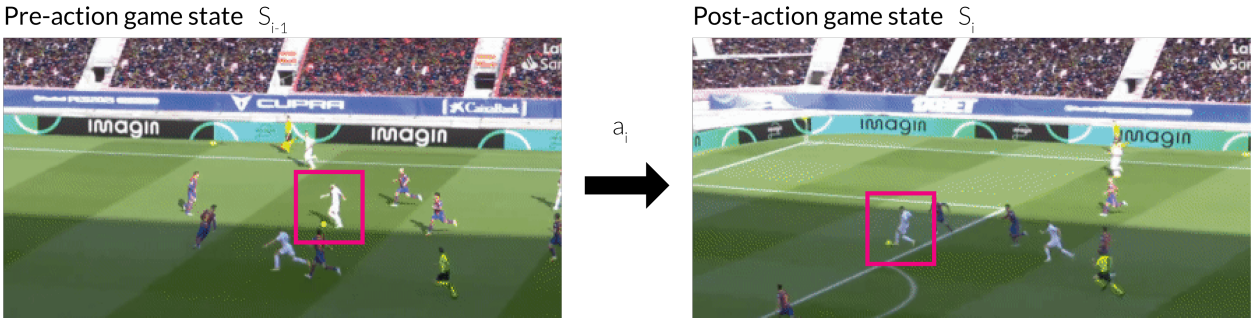
5.1 General idea

When considering event stream data, a soccer match can be viewed as a sequence of n consecutive on-the-ball actions $[a_1, a_2, \ldots, a_n]$ (e.g., [*pass*, *dribble*,..., *interception*]). Action-valuing frameworks aim to assign a numeric value to each of these individual actions that quantifies how much the action contributed towards winning the game. This value should reflect both the circumstances under which it was performed as well as its longer-term effects. This is illustrated in the figure below:



However, rather than directly assigning values to actions, the existing approaches all start by assigning values to game states. To illustrate the underlying intuition, consider the pass below:

The effect of the pass was to change the game state:



The figure on the left shows the game in state $S_{i1} = \{a_1, \ldots, a_{i1}\}$, right before Benzema passes to Valverde and the one on the right shows the game in state $S_i = \{a_1, \ldots, a_{i1}, a_i\}$ just after Valverde successfully controlled the pass.

Consequently, a natural way to assess the usefulness of an action is to assign a value to each game state. Then an action's usefulness is simply the difference between the post-action game state S_i and pre-action game state S_{i-1}. This can be expressed as:

$$U(a_i) = V(S_i) - V(S_{i-1}),$$

where V captures the value of a particular game state.

The differences between different action-valuing frameworks arise in terms of (1) how they represent a game state S_i, that is, define features such as the ball's location or score difference that capture relevant aspects of the game at a specific point in time; and (2) assign a value V to a specific game state.

5.2 Implemented frameworks

The socceraction package implements three frameworks to assess the impact of the individual actions performed by soccer players: Expected Threat (xT), VAEP and Atomic-VAEP.

5.2.1 Expected Threat (xT)

The expected threat or xT model is a possession-based model. That is, it divides matches into possessions, which are periods of the game where the same team has control of the ball. The key insights underlying xT are that (1) players perform actions with the intention to increase their team's chance of scoring, and (2) the chance of scoring can be adequately captured by only considering the location of the ball.

Point (2) means that xT represents a game state solely by using the current location of the ball. Therefore, xT overlays a $M \times N$ grid on the pitch in order to divide it into zones. Each zone z is then assigned a value $xT(z)$ that reflects how threatening teams are at that location, in terms of scoring. These xT values are illustrated in the figure below.

The value of each zone can be learned with a Markov decision process. The corresponding code is shown below. For an intuitive explanation of how this works, we refer to Karun's blog post.

```
import pandas as pd
from socceraction.data.statsbomb import StatsBombLoader
import socceraction.spadl as spadl
import socceraction.xthreat as xthreat

# 1. Load a set of actions to train the model on
SBL = StatsBombLoader()
df_games = SBL.games(competition_id=43, season_id=3)
dataset = [
    {
        **game,
        'actions': spadl.statsbomb.convert_to_actions(
            events=SBL.events(game['game_id']),
            home_team_id=game['home_team_id']
        )
    }
    for game in df_games.to_dict(orient='records')
]

# 2. Convert direction of play + add names
df_actions_ltr = pd.concat([
  spadl.play_left_to_right(game['actions'], game['home_team_id'])
  for game in dataset
])
df_actions_ltr = spadl.add_names(df_actions_ltr)
```

(continues on next page)

(continued from previous page)

```
# 3. Train xT model with 16 x 12 grid
xTModel = xthreat.ExpectedThreat(l=16, w=12)
xTModel.fit(df_actions_ltr)

# 4. Rate ball-progressing actions
# xT should only be used to value actions that move the ball
# and that keep the current team in possession of the ball
df_mov_actions = xthreat.get_successful_move_actions(df_actions_ltr)
df_mov_actions["xT_value"] = xTModel.rate(df_mov_actions)
```

See also:

This notebook gives an example of the complete pipeline to train and apply an xT model.

5.2.2 VAEP

VAEP (Valuing Actions by Estimating Probabilities) is based on the insight that players tend to perform actions with two possible intentions:

1. increase the chance of scoring a goal in the short-term future and/or,
2. decrease the chance of conceding a goal in the short-term future.

Valuing an action then requires assessing the change in probability for both scoring and conceding as a result of an action. Thus, VAEP values a game state as:

$$V(S_i) = P_{score}(S_i, t) - P_{concede}(S_i, t),$$

where $P_{score}(S_i, t)$ and $P_{concede}(S_i, t)$ are the probabilities that team t which possesses the ball in state S_i will respectively score or concede in the next 10 actions.

The remaining challenge is to "learn" $P_{score}(S_i, t)$ and $P_{concede}(S_i, t)$. That is, a gradient boosted binary classifier is trained on historical data to predict how a game state will turn out based on what happened in similar game states that arose in past games. VAEP also uses a more complex representation of the game state: it considers the three last actions that happened during the game: $S_i = \{a_{i-2}, a_{i1}, a_i\}$. With the code below, you can convert the SPADL action of the game to these game states:

```
import socceraction.vaep.features as fs

# 1. convert actions to game states
gamestates = fs.gamestates(actions, 3)
gamestates = fs.play_left_to_right(gamestates, home_team_id)
```

Then each game state is represented using three types of features. The first category of features includes characteristics of the action itself such as its location and type as well as more complex relationships such as the distance and angle to the goal. The second category of features captures the context of the action, such as the current tempo of the game, by comparing the properties of consecutive actions. Examples of this type of feature include the distance covered and time elapsed between consecutive actions. The third category of features captures the current game context by looking at things such as the time remaining in the match and the current score differential. The table below gives an overview the features that can be used to encoded a gamestate $S_i = \{a_{i-2}, a_{i1}, a_i\}$:

Transformer	Feature	Description
actiontype()	action-type(_onehot)_ai	The (one-hot encoding) of the action's type.
result()	re-sult(_onehot)_ai	The (one-hot encoding) of the action's result.
bodypart()	action-type(_onehot)_ai	The (one-hot encoding) of the bodypart used to perform the action.
time()	time_ai	Time in the match the action takes place, recorded to the second.
startlocatic	start_x_ai	The x pitch coordinate of the action's start location.
	start_y_ai	The y pitch coordinate of the action's start location.
endlocation(	end_x_ai	The x pitch coordinate of the action's end location.
	end_y_ai	The y pitch coordinate of the action's end location.
startpolar()	start_dist_to_goal	The distance to the center of the goal from the action's start location.
	start_angle_to_go	The angle between the action's start location and center of the goal.
endpolar()	end_dist_to_goal_	The distance to the center of the goal from the action's end location.
	end_angle_to_goa	The angle between the action's end location and center of the goal.
movement()	dx_ai	The distance covered by the action along the x-axis.
	dy_ai	The distance covered by the action along the y-axis.
	movement_ai	The total distance covered by the action.
team()	team_ai	Boolean indicating whether the team that had possesion in action a_{i-2} still has possession in the current action.
time_delta()	time_delta_i	Seconds elapsed between a_{i-2} and the current action.
space_delta(	dx_a0i	The distance covered by action a_{i-2} to a_i along the x-axis.
	dy_a0i	The distance covered by action a_{i-2} to a_i along the y-axis.
	mov_a0i	The total distance covered by action a_{i-2} to a_i.
goalscore()	goalscore_team	The number of goals scored by the team executing the action.
	goalscore_oppone	The number of goals scored by the other team.
	goalscore_diff	The goal difference between both teams.

```
import socceraction.vaep.features as fs

# 2. compute features
xfns = [fs.actiontype, fs.result, ...]
X = pd.concat([fn(gamestates) for fn in xfns], axis=1)
```

For estimating $P_{score}(S_i, t)$, each game state is given a positive label (= 1) if the team that possesses the ball after action a_i scores a goal in the subsequent k actions. Otherwise, a negative label (= 0) is given to the game state. Analogously, for estimating $P_{concede}(S_i, t)$, each game state is given a positive label (= 1) if the team that possesses the ball after action a_i concedes a goal in the subsequent k actions. If not, a negative label (= 0) is given to the game state.

```
import socceraction.vaep.labels as lab

# 3. compute labels
yfns = [lab.scores, lab.concedes]
Y = pd.concat([fn(actions) for fn in yfns], axis=1)
```

VAEP models the scoring and conceding probabilities separately as these effects may be asymmetric in nature and context-dependent. Hence, it trains one gradient boosted tree model to predict each one based on the current game state.

```
# 4. load or train models
models = {
```

(continues on next page)

(continued from previous page)

```
  "scores": Classsifier(...)
  "concedes": Classsifier(...)
}

# 5. predict scoring and conceding probabilities for each game state
for col in ["scores", "concedes"]:
    Y_hat[col] = models[col].predict_proba(testX)
```

Using these probabilities, VAEP defines the *offensive value* of an action as the change in scoring probability before and after the action.

$$\Delta P_{\text{score}}(a_i, t) = P^k_{\text{score}}(S_i, t) - P^k_{\text{score}}(S_{i-1}, t)$$

This change will be positive if the action increased the probability that the team which performed the action will score (e.g., a successful tackle to recover the ball). Similarly, VAEP defines the *defensive value* of an action as the change in conceding probability.

$$\Delta P_{\text{concede}}(a_i, t) = P^k_{\text{concede}}(S_i, t) - P^k_{\text{concede}}(S_{i-1}, t)$$

This change will be positive if the action increased the probability that the team will concede a goal (e.g., a failed pass). Finally, the total VAEP value of an action is the difference between that action's offensive value and defensive value.

$$V_{\text{VAEP}}(a_i) = \Delta P_{\text{score}}(a_i, t) - \Delta P_{\text{concede}}(a_i, t)$$

```
import socceraction.vaep.formula as vaepformula

# 6. compute VAEP value
values = vaepformula.value(actions, Y_hat["scores"], Y_hat["concedes"])
```

See also:

A set of notebooks illustrates the complete pipeline to train and apply a VAEP model:

1. compute features and labels
2. estimate scoring and conceding probabilities
3. compute VAEP values and top players

5.2.3 Atomic-VAEP

When building models to value actions, a heavy point of debate is how to handle the results of actions. In other words, should our model make a distinction between a failed and a successful pass or not? On the one hand, an action should be valued on all its properties, and whether or not the action was successful (e.g., did a pass receive a teammate, was a shot converted into a goal) plays a crucial role in how useful the action was. That is, if you want to measure a player's contribution during a match, successful actions are important. This is the viewpoint of SPADL and VAEP.

On the other hand, including the result of an action intertwines the contribution of the player who started the action (e.g., provides the pass) and the player who completes it (e.g., receives the pass). Perhaps a pass was not successful because of its recipient's poor touch or because he was not paying attention. It would seem unfair to penalize the player who provided the pass in such a circumstance. Hence, it can be useful to generalize over possible results of an action to arrive at an action's "expected value".

The combination of Atomic-SPADL and VAEP accomodates this alternative viewpoint. Atomic-SPADL removes the "result" attribute from SPADL and adds a few new action and event types. This affects the features that can be computed

to represent each game state. By default, Atomic-VAEP uses the following features to encoded a gamestate $S_i = \{a_{i-2}, a_{i1}, a_i\}$:

Transformer	Feature	Description
`actiontype()`	actiontype(_onehot)_ai	The (one-hot encoding) of the action's type.
`bodypart()`	actiontype(_onehot)_ai	The (one-hot encoding) of the bodypart used to perform the action.
`time()`	time_ai	Time in the match the action takes place, recorded to the second.
`team()`	team_ai	Boolean indicating whether the team that had possesion in action a_{i-2} still has possession in the current action.
`time_delta()`	time_delta_i	Seconds elapsed between a_{i-2} and the current action.
location()	x_ai	The x pitch coordinate of the action.
	y_ai	The y pitch coordinate of the action.
polar()	dist_to_goal_ai	The distance to the center of the goal.
	angle_to_goal_ai	The angle between the start location and center of the goal.
movement_pol	mov_d_ai	The distance covered by the action.
	mov_angle_ai	The direction in which the action was executed (relative to the top left of the field).
direction()	dx_ai	Direction of the action, expressed as the x-component of the unit vector.
	dy_ai	Direction of the action, expressed as the y-component of the unit vector.
goalscore()	goalscore_team	The number of goals scored by the team executing the action.
	goalscore_oppone	The number of goals scored by the other team.
	goalscore_diff	The goal difference between both teams.

The computation of the labels and the VAEP formula are similar to the standard VAEP model.

Empirically, we have noticed two benefits of using the Atomic-SPADL representation. First, the standard SPADL representation tends to assign shots a value that is the difference between the shot's true outcome and its xG score. Hence, goals or a number of misses, particularly for players who do not take a lot of shots can have an outsized effect on their VAEP score. In contrast, Atomic-SPADL assigns shots a value closer to their xG score, which often better matches domain experts' intuitions on action values.

Second, Atomic-SPADL leads to more robust action values and player ratings. A good rating system should capture the true quality of all players. Although some fluctuations in performances are possible across games, over the course of a season a few outstanding performances (possibly stemming from a big portion of luck) should not dramatically alter an assessment of a player. In our prior work comparing VAEP to xT, one advantage of xT was that it produced more stable ratings. Using Atomic-SPADL helps alleviate this weakness.

See also:

A set of notebooks illustrates the complete pipeline to train and apply an Atomic-VAEP model:

1. compute features and labels
2. estimate scoring and conceding probabilities
3. compute VAEP values and top players

CHAPTER

SIX

FAQ

Q: What is socceraction? Socceraction is an open source Python package that primarily provides an implementation of the VAEP possession value framework. However, the package also provides a number of other features, such as API clients for loading data from the most popular data providers and converters for each of these data provider's proprietary data formats to a common action-based data format (i.e., SPADL) that enables subsequent data analysis. Therefore, socceraction can take away some of the heavy data preprocessing burden from researchers and data scientists who are interested in working with soccer event stream data.

Q: Where can I get event stream data? Both StatsBomb and Wyscout provide a free sample of their data. Alternatively, you can buy a subscription to the event data feed from StatsBomb, Wyscout or Opta (Stats Perform). Instructions on how to load the data from each of these sources with socceraction are available in the *documentation*.

Q: What license is socceraction released under? Socceraction is released under the MIT license. You are free to use, modify and redistribute socceraction in any way you see fit. However, if you do use socceraction in your research, please cite our research papers. When you use socceraction in public work or when building a product or service using socceraction, we kindly request that you include the following attribution text in all advertising and documentation:

```
This product includes socceraction created by the <a href="https://dtai.cs.kuleuven.be/
↪sports/">DTAI Sports Analytics lab</a>,
available from <a href="https://github.com/ML-KULeuven/socceraction">https://github.com/
↪ML-KULeuven/socceraction</a>.
```

CHAPTER

SEVEN

SOCCERACTION.DATA

StatsBomb	Module for loading StatsBomb event data
Opta	Module for loading Opta event data and the derived formats used by Stats Perform and WhoScored
Wyscout	Module for loading Wyscout event data

7.1 socceraction.data.base

Implements serializers for the event data of various providers.

7.1.1 Serializers

socceraction.data.base.EventDataLoader	Load event data either from a remote location or from a local folder.

socceraction.data.base.EventDataLoader

class socceraction.data.base.**EventDataLoader**

Load event data either from a remote location or from a local folder.

Parameters

- **root** (*str*) – Root-path of the data.
- **getter** (*str*) – "remote" or "local"

Methods

competitions	Return a dataframe with all available competitions and seasons.
events	Return a dataframe with the event stream of a game.
games	Return a dataframe with all available games in a season.
players	Return a dataframe with all players that participated in a game.
teams	Return a dataframe with both teams that participated in a game.

socceraction.data.base.EventDataLoader.competitions

abstract EventDataLoader.**competitions**()

Return a dataframe with all available competitions and seasons.

Returns
A dataframe containing all available competitions and seasons. See `CompetitionSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.base.EventDataLoader.events

abstract EventDataLoader.**events**(*game_id*)

Return a dataframe with the event stream of a game.

Parameters
game_id (*int*) – The ID of the game.

Returns
A dataframe containing the event stream. See `EventSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.base.EventDataLoader.games

abstract EventDataLoader.**games**(*competition_id*, *season_id*)

Return a dataframe with all available games in a season.

Parameters

- **competition_id** (*int*) – The ID of the competition.
- **season_id** (*int*) – The ID of the season.

Returns
A dataframe containing all available games. See `GameSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.base.EventDataLoader.players

abstract EventDataLoader.**players**(*game_id*)

Return a dataframe with all players that participated in a game.

Parameters
game_id (*int*) – The ID of the game.

Returns
A dataframe containing all players. See PlayerSchema for the schema.

Return type
pd.DataFrame

socceraction.data.base.EventDataLoader.teams

abstract EventDataLoader.**teams**(*game_id*)

Return a dataframe with both teams that participated in a game.

Parameters
game_id (*int*) – The ID of the game.

Returns
A dataframe containing both teams. See TeamSchema for the schema.

Return type
pd.DataFrame

7.1.2 Schema

socceraction.data.schema.CompetitionSchema	Definition of a dataframe containing a list of competitions and seasons.
socceraction.data.schema.TeamSchema	Definition of a dataframe containing the list of teams of a game.
socceraction.data.schema.PlayerSchema	Definition of a dataframe containing the list of players on the teamsheet of a game.
socceraction.data.schema.GameSchema	Definition of a dataframe containing a list of games.
socceraction.data.schema.EventSchema	Definition of a dataframe containing event stream data of a game.

socceraction.data.schema.CompetitionSchema

class socceraction.data.schema.**CompetitionSchema**(**args*, ***kwargs*)

Definition of a dataframe containing a list of competitions and seasons.

Attributes

competition_id	The unique identifier for the competition.
competition_name	The name of the competition.
season_id	The unique identifier for the season.
season_name	The name of the season.

socceraction.data.schema.TeamSchema

class socceraction.data.schema.**TeamSchema**(**args*, ***kwargs*)

Definition of a dataframe containing the list of teams of a game.

Attributes

team_id	The unique identifier for the team.
team_name	The name of the team.

socceraction.data.schema.PlayerSchema

class socceraction.data.schema.**PlayerSchema**(**args*, ***kwargs*)

Definition of a dataframe containing the list of players on the teamsheet of a game.

Attributes

game_id	The unique identifier for the game.
is_starter	Whether the player is in the starting lineup.
jersey_number	The player's jersey number.
minutes_played	The number of minutes the player played in the game.
player_id	The unique identifier for the player.
player_name	The name of the player.
team_id	The unique identifier for the player's team.

socceraction.data.schema.GameSchema

class socceraction.data.schema.**GameSchema**(**args*, ***kwargs*)

Definition of a dataframe containing a list of games.

Attributes

away_team_id	The unique identifier for the away team in this game.
competition_id	The unique identifier for the competition.
game_date	The date when the game was played.
game_day	Number corresponding to the weeks or rounds into the competition this game is.
game_id	The unique identifier for the game.
home_team_id	The unique identifier for the home team in this game.
season_id	The unique identifier for the season.

socceraction.data.schema.EventSchema

class socceraction.data.schema.**EventSchema**(**args*, ***kwargs*)

Definition of a dataframe containing event stream data of a game.

Attributes

event_id	The unique identifier for the event.
game_id	The unique identifier for the game.
period_id	The unique identifier for the part of the game in which the event took place.
player_id	The unique identifier for the player this event relates to.
team_id	The unique identifier for the team this event relates to.
type_id	The unique identifier for the type of this event.
type_name	The name of the type of this event.

7.2 socceraction.data.statsbomb

Module for loading StatsBomb event data.

7.2.1 Serializers

socceraction.data.statsbomb.StatsBombLoader	Load Statsbomb data either from a remote location or from a local folder.

socceraction.data.statsbomb.StatsBombLoader

class socceraction.data.statsbomb.**StatsBombLoader**(*getter='remote'*, *root=None*, *creds=None*)

Load Statsbomb data either from a remote location or from a local folder.

To load remote data, this loader uses the statsbombpy package. Data can be retrieved from the StatsBomb API and from the Open Data GitHub repo. API access is for paying customers only. Authentication can be done by setting environment variables named `SB_USERNAME` and `SB_PASSWORD` to your login credentials. Alternatively, pass your login credentials using the `creds` parameter. StatsBomb's open data can be accessed without the need of authentication but its use is subject to a user agreement.

To load local data, point `root` to the root folder of the data. This folder should use the same directory structure as used in the Open Data GitHub repo.

Parameters

- **getter** (*str*) – "remote" or "local"
- **root** (*str, optional*) – Root-path of the data. Only used when getter is "local".
- **creds** (*dict, optional*) – Login credentials in the format {"user": "", "passwd": ""}. Only used when getter is "remote".

Methods

__init__	
competitions	Return a dataframe with all available competitions and seasons.
events	Return a dataframe with the event stream of a game.
games	Return a dataframe with all available games in a season.
players	Return a dataframe with all players that participated in a game.
teams	Return a dataframe with both teams that participated in a game.

socceraction.data.statsbomb.StatsBombLoader.__init__

StatsBombLoader.**__init__**(*getter='remote'*, *root=None*, *creds=None*)

socceraction.data.statsbomb.StatsBombLoader.competitions

StatsBombLoader.**competitions**()

Return a dataframe with all available competitions and seasons.

Raises

ParseError – When the raw data does not adhere to the expected format.

Returns

A dataframe containing all available competitions and seasons. See `StatsBombCompetitionSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.statsbomb.StatsBombLoader.events

StatsBombLoader.**events**(*game_id*, *load_360=False*)

Return a dataframe with the event stream of a game.

Parameters

- **game_id** (*int*) – The ID of the game.
- **load_360** (*bool*) – Whether to load the 360 data.

Raises
ParseError – When the raw data does not adhere to the expected format.

Returns
A dataframe containing the event stream. See StatsBombEventSchema for the schema.

Return type
pd.DataFrame

socceraction.data.statsbomb.StatsBombLoader.games

StatsBombLoader.**games**(*competition_id*, *season_id*)

Return a dataframe with all available games in a season.

Parameters

- **competition_id** (*int*) – The ID of the competition.
- **season_id** (*int*) – The ID of the season.

Raises
ParseError – When the raw data does not adhere to the expected format.

Returns
A dataframe containing all available games. See StatsBombGameSchema for the schema.

Return type
pd.DataFrame

socceraction.data.statsbomb.StatsBombLoader.players

StatsBombLoader.**players**(*game_id*)

Return a dataframe with all players that participated in a game.

Parameters
game_id (*int*) – The ID of the game.

Raises
ParseError # noqa – DAR402: When the raw data does not adhere to the expected format.

Returns
A dataframe containing all players. See StatsBombPlayerSchema for the schema.

Return type
pd.DataFrame

socceraction.data.statsbomb.StatsBombLoader.teams

StatsBombLoader.**teams**(*game_id*)

Return a dataframe with both teams that participated in a game.

Parameters
game_id (*int*) – The ID of the game.

Raises
ParseError # noqa – DAR402: When the raw data does not adhere to the expected format.

Returns
A dataframe containing both teams. See StatsBombTeamSchema for the schema.

Return type
pd.DataFrame

7.2.2 Schema

socceraction.data.statsbomb.StatsBombCompetitionSchema	Definition of a dataframe containing a list of competitions and seasons.
socceraction.data.statsbomb.StatsBombTeamSchema	Definition of a dataframe containing the list of teams of a game.
socceraction.data.statsbomb.StatsBombPlayerSchema	Definition of a dataframe containing the list of players of a game.
socceraction.data.statsbomb.StatsBombGameSchema	Definition of a dataframe containing a list of games.
socceraction.data.statsbomb.StatsBombEventSchema	Definition of a dataframe containing event stream data of a game.

socceraction.data.statsbomb.StatsBombCompetitionSchema

class socceraction.data.statsbomb.**StatsBombCompetitionSchema**(**args*, ***kwargs*)

Definition of a dataframe containing a list of competitions and seasons.

Attributes

competition_gender	The gender of the players competing in the competition.
competition_id	The unique identifier for the competition.
competition_name	The name of the competition.
country_name	The name of the country the competition relates to.
season_id	The unique identifier for the season.
season_name	The name of the season.

socceraction.data.statsbomb.StatsBombTeamSchema

class socceraction.data.statsbomb.**StatsBombTeamSchema**(**args*, ***kwargs*)

Definition of a dataframe containing the list of teams of a game.

Attributes

team_id	The unique identifier for the team.
team_name	The name of the team.

socceraction.data.statsbomb.StatsBombPlayerSchema

class socceraction.data.statsbomb.**StatsBombPlayerSchema**(**args*, ***kwargs*)

Definition of a dataframe containing the list of players of a game.

Attributes

game_id	The unique identifier for the game.
is_starter	Whether the player is in the starting lineup.
jersey_number	The player's jersey number.
minutes_played	The number of minutes the player played in the game.
nickname	The nickname of the player on the team.
player_id	The unique identifier for the player.
player_name	The name of the player.
starting_position_id	The unique identifier for the starting position of the player on the team.
starting_position_name	The name of the starting position of the player on the team.
team_id	The unique identifier for the player's team.

socceraction.data.statsbomb.StatsBombGameSchema

class socceraction.data.statsbomb.**StatsBombGameSchema**(**args*, ***kwargs*)

Definition of a dataframe containing a list of games.

Attributes

away_score	The final score of the away team.
away_team_id	The unique identifier for the away team in this game.
competition_id	The unique identifier for the competition.
competition_stage	The name of the phase of the competition this game is in.
game_date	The date when the game was played.
game_day	Number corresponding to the weeks or rounds into the competition this game is.
game_id	The unique identifier for the game.
home_score	The final score of the home team.
home_team_id	The unique identifier for the home team in this game.
referee	The name of the referee.
season_id	The unique identifier for the season.
venue	The name of the stadium where the game was played.

socceraction.data.statsbomb.StatsBombEventSchema

class socceraction.data.statsbomb.**StatsBombEventSchema**(**args*, ***kwargs*)

Definition of a dataframe containing event stream data of a game.

Attributes

counterpress	Pressing actions within 5 seconds of an open play turnover.
duration	If relevant, the length in seconds the event lasted.
event_id	The unique identifier for the event.
extra	A JSON string containing type-specific information.
freeze_frame_360	An array of freeze frame objects.
game_id	The unique identifier for the game.
index	Sequence notation for the ordering of events within each match.
location	Array containing the x and y coordinates of the event.
minute	The minutes on the clock at the time of this event.
period_id	The unique identifier for the part of the game in which the event took place.
play_pattern_id	The ID of the play pattern relevant to this event.
play_pattern_name	The name of the play pattern relevant to this event.
player_id	The unique identifier for the player this event relates to.
player_name	The name of the player this event relates to.
position_id	The ID of the position the player was in at the time of this event.
position_name	The name of the position the player was in at the time of this event.
possession	Indicates the current unique possession in the game.
possession_team_id	The ID of the team that started this possession in control of the ball.
possession_team_name	The name of the team that started this possession in control of the ball.
related_events	A comma separated list of the IDs of related events.
second	The second part of the timestamp.
team_id	The unique identifier for the team this event relates to.
team_name	The name of the team this event relates to.
timestamp	Time in the match the event takes place, recorded to the millisecond.
type_id	The unique identifier for the type of this event.
type_name	The name of the type of this event.
under_pressure	Whether the action was performed while being pressured by an opponent.
visible_area_360	An array of coordinates describing the polygon visible to the camera / in the 360 frame.

7.3 socceraction.data.opta

Module for loading Opta event data.

7.3.1 Serializers

socceraction.data.opta.OptaLoader	Load Opta data feeds from a local folder.

socceraction.data.opta.OptaLoader

class socceraction.data.opta.**OptaLoader**(*root*, *parser='xml'*, *feeds=None*)

Load Opta data feeds from a local folder.

Parameters

- **root** (*str*) – Root-path of the data.
- **parser** (*str or dict*) – Either 'xml', 'json', 'statsperform', 'whoscored' or a dict with a custom parser for each feed. The default xml parser supports F7 and F24 feeds; the default json parser supports F1, F9 and F24 feeds, the StatsPerform parser supports MA1 and MA3 feeds. Custom parsers can be specified as:

```
{
    'feed1_name': Feed1Parser
    'feed2_name': Feed2Parser
}
```

  where Feed1Parser and Feed2Parser are classes implementing `OptaParser` and 'feed1_name' and 'feed2_name' are a unique ID for each feed that matches to the keys in *feeds*.

- **feeds** (*dict*) – Glob pattern describing from which files the data from a specific game can be retrieved. For example, if files are named:

```
f7-1-2021-17362.xml
f24-1-2021-17362.xml
```

  use:

```
feeds = {
    'f7': "f7-{competition_id}-{season_id}-{game_id}.xml",
    'f24': "f24-{competition_id}-{season_id}-{game_id}.xml"
}
```

Raises

ValueError – If an invalid parser is provided.

Methods

__init__	
competitions	Return a dataframe with all available competitions and seasons.
events	Return a dataframe with the event stream of a game.
games	Return a dataframe with all available games in a season.
players	Return a dataframe with all players that participated in a game.
teams	Return a dataframe with both teams that participated in a game.

socceraction.data.opta.OptaLoader.__init__

OptaLoader.**__init__**(*root*, *parser='xml'*, *feeds=None*)

socceraction.data.opta.OptaLoader.competitions

OptaLoader.**competitions**()

Return a dataframe with all available competitions and seasons.

Returns
A dataframe containing all available competitions and seasons. See `OptaCompetitionSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.opta.OptaLoader.events

OptaLoader.**events**(*game_id*)

Return a dataframe with the event stream of a game.

Parameters
game_id (*int*) – The ID of the game.

Returns
A dataframe containing the event stream. See `OptaEventSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.opta.OptaLoader.games

OptaLoader.**games**(*competition_id*, *season_id*)

Return a dataframe with all available games in a season.

Parameters

- **competition_id** (*int*) – The ID of the competition.
- **season_id** (*int*) – The ID of the season.

Returns
A dataframe containing all available games. See OptaGameSchema for the schema.

Return type
pd.DataFrame

socceraction.data.opta.OptaLoader.players

OptaLoader.**players**(*game_id*)

Return a dataframe with all players that participated in a game.

Parameters
game_id (*int*) – The ID of the game.

Returns
A dataframe containing all players. See OptaPlayerSchema for the schema.

Return type
pd.DataFrame

socceraction.data.opta.OptaLoader.teams

OptaLoader.**teams**(*game_id*)

Return a dataframe with both teams that participated in a game.

Parameters
game_id (*int*) – The ID of the game.

Returns
A dataframe containing both teams. See OptaTeamSchema for the schema.

Return type
pd.DataFrame

7.3.2 Schema

socceraction.data.opta.OptaCompetitionSchema	Definition of a dataframe containing a list of competitions and seasons.
socceraction.data.opta.OptaTeamSchema	Definition of a dataframe containing the list of teams of a game.
socceraction.data.opta.OptaPlayerSchema	Definition of a dataframe containing the list of players of a game.
socceraction.data.opta.OptaGameSchema	Definition of a dataframe containing a list of games.
socceraction.data.opta.OptaEventSchema	Definition of a dataframe containing event stream data of a game.

socceraction.data.opta.OptaCompetitionSchema

class socceraction.data.opta.**OptaCompetitionSchema**(**args*, ***kwargs*)

Definition of a dataframe containing a list of competitions and seasons.

Attributes

competition_id	The unique identifier for the competition.
competition_name	The name of the competition.
season_id	The unique identifier for the season.
season_name	The name of the season.

socceraction.data.opta.OptaTeamSchema

class socceraction.data.opta.**OptaTeamSchema**(**args*, ***kwargs*)

Definition of a dataframe containing the list of teams of a game.

Attributes

team_id	The unique identifier for the team.
team_name	The name of the team.

socceraction.data.opta.OptaPlayerSchema

class socceraction.data.opta.**OptaPlayerSchema**(**args*, ***kwargs*)

Definition of a dataframe containing the list of players of a game.

Attributes

game_id	The unique identifier for the game.
is_starter	Whether the player is in the starting lineup.
jersey_number	The player's jersey number.
minutes_played	The number of minutes the player played in the game.
player_id	The unique identifier for the player.
player_name	The name of the player.
starting_position	The starting position of the player.
team_id	The unique identifier for the player's team.

socceraction.data.opta.OptaGameSchema

class socceraction.data.opta.**OptaGameSchema**(**args*, ***kwargs*)

Definition of a dataframe containing a list of games.

Attributes

attendance	The number of people who attended the game.
away_manager	The name of the manager of the away team.
away_score	The final score of the away team.
away_team_id	The unique identifier for the away team in this game.
competition_id	The unique identifier for the competition.
duration	The total duration of the game in minutes.
game_date	The date when the game was played.
game_day	Number corresponding to the weeks or rounds into the competition this game is.
game_id	The unique identifier for the game.
home_manager	The name of the manager of the home team.
home_score	The final score of the home team.
home_team_id	The unique identifier for the home team in this game.
referee	The name of the referee.
season_id	The unique identifier for the season.
venue	The name of the stadium where the game was played.

socceraction.data.opta.OptaEventSchema

class socceraction.data.opta.**OptaEventSchema**(**args*, ***kwargs*)

Definition of a dataframe containing event stream data of a game.

Attributes

assist	Whether the event was an assist or not.
end_x	The x coordinate of the location where the event ended.
end_y	The y coordinate of the location where the event ended.
event_id	The unique identifier for the event.
game_id	The unique identifier for the game.
goal	Whether the event was a goal or not.
keypass	Whether the event was a keypass or not.
minute	The minutes on the clock at the time of this event.
outcome	Whether the event had a successful outcome or not.
period_id	The unique identifier for the part of the game in which the event took place.
player_id	The unique identifier for the player this event relates to.
qualifiers	A JSON object containing the Opta qualifiers of the event.
related_player_id	The ID of a second player that was involved in this event.
second	The second part of the timestamp.
shot	Whether the event was a shot or not.
start_x	The x coordinate of the location where the event started.
start_y	The y coordinate of the location where the event started.
team_id	The unique identifier for the team this event relates to.
timestamp	Time in the match the event takes place, recorded to the millisecond.
touch	Whether the event was a on-the-ball action or not.
type_id	The unique identifier for the type of this event.
type_name	The name of the type of this event.

7.4 socceraction.data.wyscout

Module for loading Wyscout event data.

7.4.1 Serializers

socceraction.data.wyscout.WyscoutLoader	Load event data either from a remote location or from a local folder.
socceraction.data.wyscout.PublicWyscoutLoader	Load the public Wyscout dataset.

socceraction.data.wyscout.WyscoutLoader

class socceraction.data.wyscout.**WyscoutLoader**(*root='https://apirest.wyscout.com/v2/'*, *getter='remote'*, *feeds=None*, *creds=None*)

Load event data either from a remote location or from a local folder.

Parameters

- **root** (*str*) – Root-path of the data.
- **getter** (*str or callable, default: "remote"*) – "remote", "local" or a function that returns loads JSON data from a path.
- **feeds** (*dict(str, str)*) – Glob pattern for each feed that should be parsed. The default feeds for a "remote" getter are:

```
{
    'competitions': 'competitions',
    'seasons': 'competitions/{season_id}/seasons',
    'games': 'seasons/{season_id}/matches',
    'events': 'matches/{game_id}/events?fetch=teams,players,match,
↪substitutions'
}
```

  The default feeds for a "local" getter are:

```
{
    'competitions': 'competitions.json',
    'seasons': 'seasons_{competition_id}.json',
    'games': 'matches_{season_id}.json',
    'events': 'matches/events_{game_id}.json',
}
```

- **creds** (*dict, optional*) – Login credentials in the format {"user": "", "passwd": ""}. Only used when getter is "remote".

Methods

__init__	
competitions	Return a dataframe with all available competitions and seasons.
events	Return a dataframe with the event stream of a game.
games	Return a dataframe with all available games in a season.
players	Return a dataframe with all players that participated in a game.
teams	Return a dataframe with both teams that participated in a game.

socceraction.data.wyscout.WyscoutLoader.__init__

WyscoutLoader.**__init__**(*root='https://apirest.wyscout.com/v2/'*, *getter='remote'*, *feeds=None*, *creds=None*)

socceraction.data.wyscout.WyscoutLoader.competitions

WyscoutLoader.**competitions**(*competition_id=None*)

Return a dataframe with all available competitions and seasons.

Parameters
competition_id (*int, optional*) – The ID of the competition.

Raises
ParseError – When the raw data does not adhere to the expected format.

Returns
A dataframe containing all available competitions and seasons. See `WyscoutCompetitionSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.wyscout.WyscoutLoader.events

WyscoutLoader.**events**(*game_id*)

Return a dataframe with the event stream of a game.

Parameters
game_id (*int*) – The ID of the game.

Raises
ParseError – When the raw data does not adhere to the expected format.

Returns
A dataframe containing the event stream. See `WyscoutEventSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.wyscout.WyscoutLoader.games

WyscoutLoader.**games**(*competition_id*, *season_id*)

Return a dataframe with all available games in a season.

Parameters

- **competition_id** (*int*) – The ID of the competition.
- **season_id** (*int*) – The ID of the season.

Raises
ParseError – When the raw data does not adhere to the expected format.

Returns
A dataframe containing all available games. See `WyscoutGameSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.wyscout.WyscoutLoader.players

WyscoutLoader.**players**(*game_id*)

Return a dataframe with all players that participated in a game.

Parameters
game_id (*int*) – The ID of the game.

Raises
ParseError – When the raw data does not adhere to the expected format.

Returns
A dataframe containing all players. See WyscoutPlayerSchema for the schema.

Return type
pd.DataFrame

socceraction.data.wyscout.WyscoutLoader.teams

WyscoutLoader.**teams**(*game_id*)

Return a dataframe with both teams that participated in a game.

Parameters
game_id (*int*) – The ID of the game.

Raises
ParseError – When the raw data does not adhere to the expected format.

Returns
A dataframe containing both teams. See WyscoutTeamSchema for the schema.

Return type
pd.DataFrame

socceraction.data.wyscout.PublicWyscoutLoader

class socceraction.data.wyscout.**PublicWyscoutLoader**(*root=None*, *download=False*)

Load the public Wyscout dataset.

This dataset is a public release of event stream data, collected by Wyscout (https://wyscout.com/) containing all matches of the 2017/18 season of the top-5 European leagues (La Liga, Serie A, Bundesliga, Premier League, Ligue 1), the FIFA World Cup 2018, and UEFA Euro Cup 2016. For a detailed description, see Pappalardo et al.[1].

Parameters

- **root** (*str*) – Path where a local copy of the dataset is stored or where the downloaded dataset should be stored.
- **download** (*bool*) – Whether to force a redownload of the data.

[1] Pappalardo, L., Cintia, P., Rossi, A. et al. A public data set of spatio-temporal match events in soccer competitions. Sci Data 6, 236 (2019). https://doi.org/10.1038/s41597-019-0247-7

References

Methods

__init__	
competitions	Return a dataframe with all available competitions and seasons.
events	Return a dataframe with the event stream of a game.
games	Return a dataframe with all available games in a season.
players	Return a dataframe with all players that participated in a game.
teams	Return a dataframe with both teams that participated in a game.

socceraction.data.wyscout.PublicWyscoutLoader.__init__

PublicWyscoutLoader.**__init__**(*root=None*, *download=False*)

socceraction.data.wyscout.PublicWyscoutLoader.competitions

PublicWyscoutLoader.**competitions**()

Return a dataframe with all available competitions and seasons.

Returns
A dataframe containing all available competitions and seasons. See `WyscoutCompetitionSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.wyscout.PublicWyscoutLoader.events

PublicWyscoutLoader.**events**(*game_id*)

Return a dataframe with the event stream of a game.

Parameters
game_id (*int*) – The ID of the game.

Returns
A dataframe containing the event stream. See `WyscoutEventSchema` for the schema.

Return type
pd.DataFrame

socceraction.data.wyscout.PublicWyscoutLoader.games

PublicWyscoutLoader.**games**(*competition_id*, *season_id*)

Return a dataframe with all available games in a season.

Parameters

- **competition_id** (*int*) – The ID of the competition.
- **season_id** (*int*) – The ID of the season.

Returns
A dataframe containing all available games. See WyscoutGameSchema for the schema.

Return type
pd.DataFrame

socceraction.data.wyscout.PublicWyscoutLoader.players

PublicWyscoutLoader.**players**(*game_id*)

Return a dataframe with all players that participated in a game.

Parameters
game_id (*int*) – The ID of the game.

Returns
A dataframe containing all players. See WyscoutPlayerSchema for the schema.

Return type
pd.DataFrame

socceraction.data.wyscout.PublicWyscoutLoader.teams

PublicWyscoutLoader.**teams**(*game_id*)

Return a dataframe with both teams that participated in a game.

Parameters
game_id (*int*) – The ID of the game.

Returns
A dataframe containing both teams. See WyscoutTeamSchema for the schema.

Return type
pd.DataFrame

7.4.2 Schema

socceraction.data.wyscout.WyscoutCompetitionSchema	Definition of a dataframe containing a list of competitions and seasons.
socceraction.data.wyscout.WyscoutTeamSchema	Definition of a dataframe containing the list of players of a game.
socceraction.data.wyscout.WyscoutPlayerSchema	Definition of a dataframe containing the list of teams of a game.
socceraction.data.wyscout.WyscoutGameSchema	Definition of a dataframe containing a list of games.
socceraction.data.wyscout.WyscoutEventSchema	Definition of a dataframe containing event stream data of a game.

socceraction.data.wyscout.WyscoutCompetitionSchema

class socceraction.data.wyscout.**WyscoutCompetitionSchema**(**args*, ***kwargs*)

Definition of a dataframe containing a list of competitions and seasons.

Attributes

competition_gender	
competition_id	The unique identifier for the competition.
competition_name	The name of the competition.
country_name	
season_id	The unique identifier for the season.
season_name	The name of the season.

socceraction.data.wyscout.WyscoutTeamSchema

class socceraction.data.wyscout.**WyscoutTeamSchema**(**args*, ***kwargs*)

Definition of a dataframe containing the list of players of a game.

Attributes

team_id	The unique identifier for the team.
team_name	The name of the team.
team_name_short	

socceraction.data.wyscout.WyscoutPlayerSchema

class socceraction.data.wyscout.**WyscoutPlayerSchema**(**args*, ***kwargs*)

Definition of a dataframe containing the list of teams of a game.

Attributes

birth_date	
firstname	
game_id	The unique identifier for the game.
is_starter	Whether the player is in the starting lineup.
jersey_number	The player's jersey number.
lastname	
minutes_played	The number of minutes the player played in the game.
nickname	
player_id	The unique identifier for the player.
player_name	The name of the player.
team_id	The unique identifier for the player's team.

socceraction.data.wyscout.WyscoutGameSchema

class socceraction.data.wyscout.**WyscoutGameSchema**(**args*, ***kwargs*)

Definition of a dataframe containing a list of games.

Attributes

away_team_id	The unique identifier for the away team in this game.
competition_id	The unique identifier for the competition.
game_date	The date when the game was played.
game_day	Number corresponding to the weeks or rounds into the competition this game is.
game_id	The unique identifier for the game.
home_team_id	The unique identifier for the home team in this game.
season_id	The unique identifier for the season.

socceraction.data.wyscout.WyscoutEventSchema

class socceraction.data.wyscout.**WyscoutEventSchema**(**args*, ***kwargs*)

Definition of a dataframe containing event stream data of a game.

Attributes

event_id	The unique identifier for the event.
game_id	The unique identifier for the game.
milliseconds	
period_id	The unique identifier for the part of the game in which the event took place.
player_id	The unique identifier for the player this event relates to.
positions	
subtype_id	
subtype_name	
tags	
team_id	The unique identifier for the team this event relates to.
type_id	The unique identifier for the type of this event.
type_name	The name of the type of this event.

CHAPTER

EIGHT

SOCCERACTION.SPADL

Implementation of the SPADL language.

8.1 Converters

socceraction.spadl.statsbomb.convert_to_actions	Convert StatsBomb events to SPADL actions.
socceraction.spadl.opta.convert_to_actions	Convert Opta events to SPADL actions.
socceraction.spadl.wyscout.convert_to_actions	Convert Wyscout events to SPADL actions.
socceraction.spadl.kloppy.convert_to_actions	Convert a Kloppy event data set to SPADL actions.

8.1.1 socceraction.spadl.statsbomb.convert_to_actions

socceraction.spadl.statsbomb.**convert_to_actions**(*events*, *home_team_id*, *xy_fidelity_version=None*, *shot_fidelity_version=None*)

Convert StatsBomb events to SPADL actions.

Parameters

- **events** (*pd.DataFrame*) – DataFrame containing StatsBomb events from a single game.
- **home_team_id** (*int*) – ID of the home team in the corresponding game.
- **xy_fidelity_version** (*int, optional*) – Whether low or high fidelity coordinates are used in the event data. If not specified, the fidelity version is inferred from the data.
- **shot_fidelity_version** (*int, optional*) – Whether low or high fidelity coordinates are used in the event data for shots. If not specified, the fidelity version is inferred from the data.

Returns

actions – DataFrame with corresponding SPADL actions.

Return type

pd.DataFrame

8.1.2 socceraction.spadl.opta.convert_to_actions

socceraction.spadl.opta.**convert_to_actions**(*events*, *home_team_id*)

Convert Opta events to SPADL actions.

Parameters

- **events** (*pd.DataFrame*) – DataFrame containing Opta events from a single game.
- **home_team_id** (*int*) – ID of the home team in the corresponding game.

Returns

actions – DataFrame with corresponding SPADL actions.

Return type

pd.DataFrame

8.1.3 socceraction.spadl.wyscout.convert_to_actions

socceraction.spadl.wyscout.**convert_to_actions**(*events*, *home_team_id*)

Convert Wyscout events to SPADL actions.

Parameters

- **events** (*pd.DataFrame*) – DataFrame containing Wyscout events from a single game.
- **home_team_id** (*int*) – ID of the home team in the corresponding game.

Returns

actions – DataFrame with corresponding SPADL actions.

Return type

pd.DataFrame

8.1.4 socceraction.spadl.kloppy.convert_to_actions

socceraction.spadl.kloppy.**convert_to_actions**(*dataset*, *game_id=None*)

Convert a Kloppy event data set to SPADL actions.

Parameters

- **dataset** (*EventDataset*) – A Kloppy event data set.
- **game_id** (*str or int, optional*) – The identifier of the game. If not provided, the game id will not be set in the SPADL DataFrame.

Returns

actions – DataFrame with corresponding SPADL actions.

Return type

pd.DataFrame

8.2 Schema

socceraction.spadl.SPADLSchema	Definition of a SPADL dataframe.

8.2.1 socceraction.spadl.SPADLSchema

class socceraction.spadl.**SPADLSchema**(**args*, ***kwargs*)

Definition of a SPADL dataframe.

Attributes

action_id
bodypart_id
bodypart_name
end_x
end_y
game_id
original_event_id
period_id
player_id
result_id
result_name
start_x
start_y
team_id
time_seconds
type_id
type_name

8.3 Config

socceraction.spadl.config.field_length	Convert a string or number to a floating point number, if possible.
socceraction.spadl.config.field_width	Convert a string or number to a floating point number, if possible.
socceraction.spadl.config.actiontypes	Built-in mutable sequence.
socceraction.spadl.config.bodyparts	Built-in mutable sequence.
socceraction.spadl.config.results	Built-in mutable sequence.

8.3.1 socceraction.spadl.config.field_length

socceraction.spadl.config.**field_length: float = 105.0**

Convert a string or number to a floating point number, if possible.

8.3.2 socceraction.spadl.config.field_width

socceraction.spadl.config.**field_width: float = 68.0**

Convert a string or number to a floating point number, if possible.

8.3.3 socceraction.spadl.config.actiontypes

socceraction.spadl.config.**actiontypes: list[str] = ['pass', 'cross', 'throw_in', 'freekick_crossed', 'freekick_short', 'corner_crossed', 'corner_short', 'take_on', 'foul', 'tackle', 'interception', 'shot', 'shot_penalty', 'shot_freekick', 'keeper_save', 'keeper_claim', 'keeper_punch', 'keeper_pick_up', 'clearance', 'bad_touch', 'non_action', 'dribble', 'goalkick']**

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

8.3.4 socceraction.spadl.config.bodyparts

socceraction.spadl.config.**bodyparts: list[str] = ['foot', 'head', 'other', 'head/other', 'foot_left', 'foot_right']**

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

8.3.5 socceraction.spadl.config.results

socceraction.spadl.config.**results: list[str] = ['fail', 'success', 'offside', 'owngoal', 'yellow_card', 'red_card']**

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

8.4 Utility functions

socceraction.spadl.play_left_to_right	Perform all action in the same playing direction.
socceraction.spadl.add_names	Add the type name, result name and bodypart name to a SPADL dataframe.
socceraction.spadl.actiontypes_df	Return a dataframe with the type id and type name of each SPADL action type.
socceraction.spadl.bodyparts_df	Return a dataframe with the bodypart id and bodypart name of each SPADL action type.
socceraction.spadl.results_df	Return a dataframe with the result id and result name of each SPADL action type.

8.4.1 socceraction.spadl.play_left_to_right

socceraction.spadl.**play_left_to_right**(*actions*, *home_team_id*)

Perform all action in the same playing direction.

This changes the start and end location of each action, such that all actions are performed as if the team that executes the action plays from left to right.

Parameters

- **actions** (*pd.DataFrame*) – The SPADL actins of a game.
- **home_team_id** (*int*) – The ID of the home team.

Returns
All actions performed left to right.

Return type
list(pd.DataFrame)

See also:

socceraction.vaep.features.play_left_to_right
For transforming gamestates.

8.4.2 socceraction.spadl.add_names

socceraction.spadl.**add_names**(*actions*)

Add the type name, result name and bodypart name to a SPADL dataframe.

Parameters
actions (*pd.DataFrame*) – A SPADL dataframe.

Returns
The original dataframe with a 'type_name', 'result_name' and 'bodypart_name' appended.

Return type
pd.DataFrame

8.4.3 socceraction.spadl.actiontypes_df

socceraction.spadl.**actiontypes_df**()

Return a dataframe with the type id and type name of each SPADL action type.

Returns
The 'type_id' and 'type_name' of each SPADL action type.

Return type
pd.DataFrame

8.4.4 socceraction.spadl.bodyparts_df

socceraction.spadl.**bodyparts_df**()

Return a dataframe with the bodypart id and bodypart name of each SPADL action type.

Returns
The 'bodypart_id' and 'bodypart_name' of each SPADL action type.

Return type
pd.DataFrame

8.4.5 socceraction.spadl.results_df

socceraction.spadl.**results_df**()

Return a dataframe with the result id and result name of each SPADL action type.

Returns
The 'result_id' and 'result_name' of each SPADL action type.

Return type
pd.DataFrame

CHAPTER

NINE

SOCCERACTION.XTHREAT

Implements the xT framework.

9.1 Model

socceraction.xthreat.ExpectedThreat	An implementation of the Expected Threat (xT) model.

9.1.1 socceraction.xthreat.ExpectedThreat

class socceraction.xthreat.**ExpectedThreat**(*l=16*, *w=12*, *eps=1e-05*)

An implementation of the Expected Threat (xT) model.

The xT model[1] can be used to value actions that successfully move the ball between two locations on the pitch by computing the difference between the long-term probability of scoring on the start and end location of an action.

Parameters

- **l** (*int*) – Amount of grid cells in the x-dimension of the grid.
- **w** (*int*) – Amount of grid cells in the y-dimension of the grid.
- **eps** (*float*) – The desired precision to calculate the xT value of a cell. Default is 5 decimal places of precision (1e-5).

l

Amount of grid cells in the x-dimension of the grid.

Type
int

w

Amount of grid cells in the y-dimension of the grid.

Type
int

eps

The desired precision to calculate the xT value of a cell. Default is 5 decimal places of precision (1e-5).

Type
float

[1] Singh, Karun. "Introducing Expected Threat (xT)." 15 February, 2019. https://karun.in/blog/expected-threat.html

heatmaps

The i-th element corresponds to the xT value surface after i iterations.

Type
list(np.ndarray)

xT

The final xT value surface.

Type
np.ndarray

scoring_prob_matrix

The probability of scoring when taking a shot for each cell.

Type
np.ndarray, shape(M,N)

shot_prob_matrix

The probability of choosing to shoot for each cell.

Type
np.ndarray, shape(M,N)

move_prob_matrix

The probability of choosing to move for each cell.

Type
np.ndarray, shape(M,N)

transition_matrix

When moving, the probability of moving to each of the other zones.

Type
np.ndarray, shape(M*N,M*N)

References

Methods

__init__	
fit	Fits the xT model with the given actions.
interpolator	Interpolate over the pitch.
rate	Compute the xT values for the given actions.
save_model	Save the xT value surface in JSON format.

socceraction.xthreat.ExpectedThreat.__init__

ExpectedThreat.**__init__**(*l=16*, *w=12*, *eps=1e-05*)

socceraction.xthreat.ExpectedThreat.fit

ExpectedThreat.**fit**(*actions*)

Fits the xT model with the given actions.

Parameters
actions (*pd.DataFrame*) – Actions, in SPADL format.

Returns
Fitted xT model.

Return type
self

socceraction.xthreat.ExpectedThreat.interpolator

ExpectedThreat.**interpolator**(*kind='linear'*)

Interpolate over the pitch.

This is a wrapper around `scipy.interpolate.interp2d()`.

Parameters
kind (*{'linear', 'cubic', 'quintic'} # noqa: DAR103*) – The kind of spline interpolation to use. Default is 'linear'.

Raises
ImportError – If scipy is not installed.

Returns
A function that interpolates xT values over the pitch.

Return type
callable

socceraction.xthreat.ExpectedThreat.rate

ExpectedThreat.**rate**(*actions*, *use_interpolation=False*)

Compute the xT values for the given actions.

xT should only be used to value actions that move the ball and also keep the current team in possession of the ball. All other actions in the given dataframe receive a *NaN* rating.

Parameters

- **actions** (*pd.DataFrame*) – Actions, in SPADL format.
- **use_interpolation** (*bool*) – Indicates whether to use bilinear interpolation when inferring xT values. Note that this requires Scipy to be installed (pip install scipy).

Raises
NotFittedError – If the model has not been fitted yet.

Returns
The xT value for each action.

Return type
np.ndarray

socceraction.xthreat.ExpectedThreat.save_model

ExpectedThreat.**save_model**(*filepath*, *overwrite=True*)

Save the xT value surface in JSON format.

This stores only the xT value surface, which is all you need to compute xT values for new data. The value surface can be loaded back with the *socceraction.xthreat.load_model()* function.

Pickle the *ExpectedThreat* instance to store the entire model and to retain the transition, shot probability, move probability and scoring probability matrices.

Raises

- **NotFittedError** – If the model has not been fitted yet.
- **ValueError** – If the specified output file already exists and "overwrite" is set to False.

Parameters

- **filepath** (*str*) – Path to the file to save the value surface to.
- **overwrite** (*bool*) – Whether to silently overwrite any existing file at the target location.

Return type
None

9.2 Utility functions

socceraction.xthreat.load_model	Create a model from a pre-computed xT value surface.
socceraction.xthreat.get_move_actions	Get all ball-progressing actions.
socceraction.xthreat.get_successful_move_actions	Get all successful ball-progressing actions.
socceraction.xthreat.scoring_prob	Compute the probability of scoring when taking a shot for each cell.
socceraction.xthreat.action_prob	Compute the probability of taking an action in each cell of the grid.
socceraction.xthreat.move_transition_matrix	Compute the move transition matrix from the given actions.

9.2.1 socceraction.xthreat.load_model

socceraction.xthreat.**load_model**(*path*)

Create a model from a pre-computed xT value surface.

The value surface should be provided as a JSON file containing a 2D matrix. Karun Singh provides such a grid at the follwing url: https://karun.in/blog/data/open_xt_12x8_v1.json

Parameters
path (*str*) – Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, and file.

Returns
An xT model that uses the given value surface to value actions.

Return type
ExpectedThreat

9.2.2 socceraction.xthreat.get_move_actions

socceraction.xthreat.**get_move_actions**(*actions*)

Get all ball-progressing actions.

These include passes, dribbles and crosses. Take-ons are ignored because they typically coincide with dribbles and do not move the ball to a different cell.

Parameters
actions (*pd.DataFrame*) – Actions, in SPADL format.

Returns
All ball-progressing actions in the input dataframe.

Return type
pd.DataFrame

9.2.3 socceraction.xthreat.get_successful_move_actions

socceraction.xthreat.**get_successful_move_actions**(*actions*)

Get all successful ball-progressing actions.

These include successful passes, dribbles and crosses.

Parameters
actions (*pd.DataFrame*) – Actions, in SPADL format.

Returns
All ball-progressing actions in the input dataframe.

Return type
pd.DataFrame

9.2.4 socceraction.xthreat.scoring_prob

socceraction.xthreat.**scoring_prob**(*actions*, *l=16*, *w=12*)

Compute the probability of scoring when taking a shot for each cell.

Parameters

- **actions** (*pd.DataFrame*) – Actions, in SPADL format.
- **l** (*int*) – Amount of grid cells in the x-dimension of the grid.
- **w** (*int*) – Amount of grid cells in the y-dimension of the grid.

Returns
A matrix, denoting the probability of scoring for each cell.

Return type
np.ndarray

9.2.5 socceraction.xthreat.action_prob

socceraction.xthreat.**action_prob**(*actions*, *l=16*, *w=12*)

Compute the probability of taking an action in each cell of the grid.

The options are: shooting or moving.

Parameters

- **actions** (*pd.DataFrame*) – Actions, in SPADL format.
- **l** (*int*) – Amount of grid cells in the x-dimension of the grid.
- **w** (*int*) – Amount of grid cells in the y-dimension of the grid.

Return type
tuple[ndarray[Any, dtype[float64]], ndarray[Any, dtype[float64]]]

Returns

- **shotmatrix** (*np.ndarray*) – For each cell the probability of choosing to shoot.
- **movematrix** (*np.ndarray*) – For each cell the probability of choosing to move.

9.2.6 socceraction.xthreat.move_transition_matrix

socceraction.xthreat.**move_transition_matrix**(*actions*, *l=16*, *w=12*)

Compute the move transition matrix from the given actions.

This is, when a player chooses to move, the probability that he will end up in each of the other cells of the grid successfully.

Parameters

- **actions** (*pd.DataFrame*) – Actions, in SPADL format.
- **l** (*int*) – Amount of grid cells in the x-dimension of the grid.
- **w** (*int*) – Amount of grid cells in the y-dimension of the grid.

Returns
The transition matrix.

Return type
np.ndarray

CHAPTER

TEN

SOCCERACTION.VAEP

Implements the VAEP framework.

10.1 Model

socceraction.vaep.VAEP	An implementation of the VAEP framework.

10.1.1 socceraction.vaep.VAEP

class socceraction.vaep.**VAEP**(*xfns=None*, *nb_prev_actions=3*)

An implementation of the VAEP framework.

VAEP (Valuing Actions by Estimating Probabilities)[1] defines the problem of valuing a soccer player's contributions within a match as a binary classification problem and rates actions by estimating its effect on the short-term probablities that a team will both score and concede.

Parameters

- **xfns** (*list*) – List of feature transformers (see *socceraction.vaep.features*) used to describe the game states. Uses `xfns_default` if None.
- **nb_prev_actions** (*int, default=3 # noqa: DAR103*) – Number of previous actions used to decscribe the game state.

References

[1] Tom Decroos, Lotte Bransen, Jan Van Haaren, and Jesse Davis. "Actions speak louder than goals: Valuing player actions in soccer." In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1851-1861. 2019.

Methods

__init__	
compute_features	Transform actions to the feature-based representation of game states.
compute_labels	Compute the labels for each game state in the given game.
fit	Fit the model according to the given training data.
rate	Compute the VAEP rating for the given game states.
score	Evaluate the fit of the model on the given test data and labels.

socceraction.vaep.VAEP.__init__

VAEP.**__init__**(*xfns=None*, *nb_prev_actions=3*)

socceraction.vaep.VAEP.compute_features

VAEP.**compute_features**(*game*, *game_actions*)

Transform actions to the feature-based representation of game states.

Parameters

- **game** (*pd.Series*) – The SPADL representation of a single game.
- **game_actions** (*pd.DataFrame*) – The actions performed during *game* in the SPADL representation.

Returns

features – Returns the feature-based representation of each game state in the game.

Return type

pd.DataFrame

socceraction.vaep.VAEP.compute_labels

VAEP.**compute_labels**(*game*, *game_actions*)

Compute the labels for each game state in the given game.

Parameters

- **game** (*pd.Series*) – The SPADL representation of a single game.
- **game_actions** (*pd.DataFrame*) – The actions performed during *game* in the SPADL representation.

Returns

labels – Returns the labels of each game state in the game.

Return type

pd.DataFrame

socceraction.vaep.VAEP.fit

VAEP.**fit**(*X*, *y*, *learner='xgboost'*, *val_size=0.25*, *tree_params=None*, *fit_params=None*)

Fit the model according to the given training data.

Parameters

- **X** (*pd.DataFrame*) – Feature representation of the game states.
- **y** (*pd.DataFrame*) – Scoring and conceding labels for each game state.
- **learner** (*string, default='xgboost' # noqa: DAR103*) – Gradient boosting implementation which should be used to learn the model. The supported learners are 'xgboost', 'catboost' and 'lightgbm'.
- **val_size** (*float, default=0.25 # noqa: DAR103*) – Percentage of the dataset that will be used as the validation set for early stopping. When zero, no validation data will be used.
- **tree_params** (*dict*) – Parameters passed to the constructor of the learner.
- **fit_params** (*dict*) – Parameters passed to the fit method of the learner.

Raises

ValueError – If one of the features is missing in the provided dataframe.

Returns

Fitted VAEP model.

Return type

self

socceraction.vaep.VAEP.rate

VAEP.**rate**(*game*, *game_actions*, *game_states=None*)

Compute the VAEP rating for the given game states.

Parameters

- **game** (*pd.Series*) – The SPADL representation of a single game.
- **game_actions** (*pd.DataFrame*) – The actions performed during *game* in the SPADL representation.
- **game_states** (*pd.DataFrame, default=None*) – DataFrame with the game state representation of each action. If *None*, these will be computed on-th-fly.

Raises

NotFittedError – If the model is not fitted yet.

Returns

ratings – Returns the VAEP rating for each given action, as well as the offensive and defensive value of each action.

Return type

pd.DataFrame

socceraction.vaep.VAEP.score

VAEP.**score**(*X*, *y*)

Evaluate the fit of the model on the given test data and labels.

Parameters

- **X** (*pd.DataFrame*) – Feature representation of the game states.
- **y** (*pd.DataFrame*) – Scoring and conceding labels for each game state.

Raises

NotFittedError – If the model is not fitted yet.

Returns

score – The Brier and AUROC scores for both binary classification problems.

Return type

dict

10.2 Utility functions

socceraction.vaep.features	Implements the feature tranformers of the VAEP framework.
socceraction.vaep.labels	Implements the label tranformers of the VAEP framework.
socceraction.vaep.formula	Implements the formula of the VAEP framework.

10.2.1 socceraction.vaep.features

Implements the feature tranformers of the VAEP framework.

socceraction.vaep.features.**actiontype**(*actions*)

Get the type of each action.

Parameters

actions (*Actions*) – The actions of a game.

Returns

The 'type_id' of each action.

Return type

Features

socceraction.vaep.features.**actiontype_onehot**(*actions*)

Get the one-hot-encoded type of each action.

Parameters

actions (*SPADLActions*) – The actions of a game.

Returns

A one-hot encoding of each action's type.

Return type

Features

socceraction.vaep.features.**actiontype_result_onehot**(*actions*)

Get a one-hot encoding of the combination between the type and result of each action.

Parameters
actions (*SPADLActions*) – The actions of a game.

Returns
The one-hot encoding of each action's type and result.

Return type
Features

socceraction.vaep.features.**bodypart**(*actions*)

Get the body part used to perform each action.

This feature generator does not distinguish between the left and right foot.

Parameters
actions (*Actions*) – The actions of a game.

Returns
The 'bodypart_id' of each action.

Return type
Features

See also:

bodypart_detailed
An alternative version that splits between the left and right foot.

socceraction.vaep.features.**bodypart_detailed**(*actions*)

Get the body part with split by foot used to perform each action.

This feature generator distinguishes between the left and right foot, if supported by the dataprovider.

Parameters
actions (*Actions*) – The actions of a game.

Returns
The 'bodypart_id' of each action.

Return type
Features

See also:

bodypart
An alternative version that does not split between the left and right foot.

socceraction.vaep.features.**bodypart_detailed_onehot**(*actions*)

Get the one-hot-encoded bodypart with split by foot of each action.

This feature generator distinguishes between the left and right foot, if supported by the dataprovider.

Parameters
actions (*Actions*) – The actions of a game.

Returns
The one-hot encoding of each action's bodypart.

Return type
Features

See also:

bodypart_onehot
An alternative version that does not split between the left and right foot.

socceraction.vaep.features.**bodypart_onehot**(*actions*)

Get the one-hot-encoded bodypart of each action.

This feature generator does not distinguish between the left and right foot.

Parameters
actions (*Actions*) – The actions of a game.

Returns
The one-hot encoding of each action's bodypart.

Return type
Features

See also:

bodypart_detailed_onehot
An alternative version that splits between the left and right foot.

socceraction.vaep.features.**endlocation**(*actions*)

Get the location where each action ended.

Parameters
actions (*SPADLActions*) – The actions of a game.

Returns
The 'end_x' and 'end_y' location of each action.

Return type
Features

socceraction.vaep.features.**endpolar**(*actions*)

Get the polar coordinates of each action's end location.

The center of the opponent's goal is used as the origin.

Parameters
actions (*SPADLActions*) – The actions of a game.

Returns
The 'start_dist_to_goal' and 'start_angle_to_goal' of each action.

Return type
Features

socceraction.vaep.features.**feature_column_names**(*fs*, *nb_prev_actions=3*)

Return the names of the features generated by a list of transformers.

Parameters

- **fs** (*list(callable)*) – A list of feature transformers.
- **nb_prev_actions** (*int, default=3 # noqa: DAR103*) – The number of previous actions included in the game state.

Returns
The name of each generated feature.

Return type
list(str)

socceraction.vaep.features.**gamestates**(*actions*, *nb_prev_actions=3*)

Convert a dataframe of actions to gamestates.

Each gamestate is represented as the <nb_prev_actions> previous actions.

The list of gamestates is internally represented as a list of actions dataframes $[a_0, a_1, \ldots]$ where each row in the a_i dataframe contains the previous action of the action in the same row in the a_{i-1} dataframe.

Parameters

- **actions** (*Actions*) – A DataFrame with the actions of a game.
- **nb_prev_actions** (*int, default=3 # noqa: DAR103*) – The number of previous actions included in the game state.

Raises
ValueError – If the number of actions is smaller 1.

Returns
The <nb_prev_actions> previous actions for each action.

Return type
GameStates

socceraction.vaep.features.**goalscore**(*gamestates*)

Get the number of goals scored by each team after the action.

Parameters
gamestates (*GameStates*) – The gamestates of a game.

Returns
The number of goals scored by the team performing the last action of the game state ('goalscore_team'), by the opponent ('goalscore_opponent'), and the goal difference between both teams ('goalscore_diff').

Return type
Features

socceraction.vaep.features.**movement**(*actions*)

Get the distance covered by each action.

Parameters
actions (*SPADLActions*) – The actions of a game.

Returns
The horizontal ('dx'), vertical ('dy') and total ('movement') distance covered by each action.

Return type
Features

socceraction.vaep.features.**play_left_to_right**(*gamestates*, *home_team_id*)

Perform all actions in a gamestate in the same playing direction.

This changes the start and end location of each action in a gamestate, such that all actions are performed as if the team that performs the first action in the gamestate plays from left to right.

Parameters

- **gamestates** (*GameStates*) – The game states of a game.
- **home_team_id** (*int*) – The ID of the home team.

Returns
The game states with all actions performed left to right.

Return type
GameStates

See also:

socceraction.vaep.features.play_left_to_right
For transforming actions.

socceraction.vaep.features.**player_possession_time**(*actions*)

Get the time (sec) a player was in ball possession before attempting the action.

We only look at the dribble preceding the action and reset the possession time after a defensive interception attempt or a take-on.

Parameters
actions (*SPADLActions*) – The actions of a game.

Returns
The 'player_possession_time' of each action.

Return type
Features

socceraction.vaep.features.**result**(*actions*)

Get the result of each action.

Parameters
actions (*SPADLActions*) – The actions of a game.

Returns
The 'result_id' of each action.

Return type
Features

socceraction.vaep.features.**result_onehot**(*actions*)

Get the one-hot-encode result of each action.

Parameters
actions (*SPADLActions*) – The actions of a game.

Returns
The one-hot encoding of each action's result.

Return type
Features

socceraction.vaep.features.**simple**(*actionfn*)

Make a function decorator to apply actionfeatures to game states.

Parameters
actionfn (*callable*) – A feature transformer that operates on actions.

Returns
A feature transformer that operates on game states.

Return type
FeatureTransfomer

socceraction.vaep.features.**space_delta**(*gamestates*)

Get the distance covered between the last and previous actions.

Parameters
gamestates (*GameStates*) – The gamestates of a game.

Returns
A dataframe with a column for the horizontal ('dx_a0i'), vertical ('dy_a0i') and total ('mov_a0i') distance covered between each <nb_prev_actions> action ai and action a0.

Return type
Features

socceraction.vaep.features.**speed**(*gamestates*)

Get the speed at which the ball moved during the previous actions.

Parameters
gamestates (*GameStates*) – The game states of a game.

Returns
A dataframe with columns 'speedx_a0i', 'speedy_a0i', 'speed_a0i' for each <nb_prev_actions> containing the ball speed in m/s between action ai and action a0.

Return type
Features

socceraction.vaep.features.**startlocation**(*actions*)

Get the location where each action started.

Parameters
actions (*SPADLActions*) – The actions of a game.

Returns
The 'start_x' and 'start_y' location of each action.

Return type
Features

socceraction.vaep.features.**startpolar**(*actions*)

Get the polar coordinates of each action's start location.

The center of the opponent's goal is used as the origin.

Parameters
actions (*SPADLActions*) – The actions of a game.

Returns
The 'start_dist_to_goal' and 'start_angle_to_goal' of each action.

Return type
Features

socceraction.vaep.features.**team**(*gamestates*)

Check whether the possession changed during the game state.

For each action in the game state, True if the team that performed the action is the same team that performed the last action of the game state; otherwise False.

Parameters
gamestates (*GameStates*) – The game states of a game.

Returns
A dataframe with a column 'team_ai' for each <nb_prev_actions> indicating whether the team that performed action a0 is in possession.

Return type
Features

socceraction.vaep.features.**time**(*actions*)

Get the time when each action was performed.

This generates the following features:

period_id
The ID of the period.

time_seconds
Seconds since the start of the period.

time_seconds_overall
Seconds since the start of the game. Stoppage time during previous periods is ignored.

Parameters
actions (*Actions*) – The actions of a game.

Returns
The 'period_id', 'time_seconds' and 'time_seconds_overall' when each action was performed.

Return type
Features

socceraction.vaep.features.**time_delta**(*gamestates*)

Get the number of seconds between the last and previous actions.

Parameters
gamestates (*GameStates*) – The game states of a game.

Returns
A dataframe with a column 'time_delta_i' for each <nb_prev_actions> containing the number of seconds between action ai and action a0.

Return type
Features

10.2.2 socceraction.vaep.labels

Implements the label tranformers of the VAEP framework.

socceraction.vaep.labels.**concedes**(*actions*, *nr_actions=10*)

Determine whether the team possessing the ball conceded a goal within the next x actions.

Parameters

- **actions** (*pd.DataFrame*) – The actions of a game.
- **nr_actions** (*int, default=10 # noqa: DAR103*) – Number of actions after the current action to consider.

Returns
A dataframe with a column 'concedes' and a row for each action set to True if a goal was conceded by the team possessing the ball within the next x actions; otherwise False.

Return type
pd.DataFrame

socceraction.vaep.labels.**goal_from_shot**(*actions*)

Determine whether a goal was scored from the current action.

This label can be use to train an xG model.

Parameters
actions (*pd.DataFrame*) – The actions of a game.

Returns
A dataframe with a column 'goal' and a row for each action set to True if a goal was scored from the current action; otherwise False.

Return type
pd.DataFrame

socceraction.vaep.labels.**scores**(*actions*, *nr_actions=10*)

Determine whether the team possessing the ball scored a goal within the next x actions.

Parameters

- **actions** (*pd.DataFrame*) – The actions of a game.
- **nr_actions** (*int, default=10 # noqa: DAR103*) – Number of actions after the current action to consider.

Returns
A dataframe with a column 'scores' and a row for each action set to True if a goal was scored by the team possessing the ball within the next x actions; otherwise False.

Return type
pd.DataFrame

10.2.3 socceraction.vaep.formula

Implements the formula of the VAEP framework.

socceraction.vaep.formula.**defensive_value**(*actions*, *scores*, *concedes*)

Compute the defensive value of each action.

VAEP defines the *defensive value* of an action as the change in conceding probability.

$$\Delta P_{concede}(a_i, t) = P^k_{concede}(S_i, t) - P^k_{concede}(S_{i-1}, t)$$

where $P_{concede}(S_i, t)$ is the probability that team t which possesses the ball in state S_i will concede in the next 10 actions.

Parameters

- **actions** (*pd.DataFrame*) – SPADL action.
- **scores** (*pd.Series*) – The probability of scoring from each corresponding game state.
- **concedes** (*pd.Series*) – The probability of conceding from each corresponding game state.

Returns
The defensive value of each action.

Return type
pd.Series

socceraction.vaep.formula.**offensive_value**(*actions*, *scores*, *concedes*)

Compute the offensive value of each action.

VAEP defines the *offensive value* of an action as the change in scoring probability before and after the action.

$$\Delta P_{score}(a_i, t) = P^k_{score}(S_i, t) - P^k_{score}(S_{i-1}, t)$$

where $P_{score}(S_i, t)$ is the probability that team t which possesses the ball in state S_i will score in the next 10 actions.

Parameters

- **actions** (*pd.DataFrame*) – SPADL action.
- **scores** (*pd.Series*) – The probability of scoring from each corresponding game state.
- **concedes** (*pd.Series*) – The probability of conceding from each corresponding game state.

Returns
The offensive value of each action.

Return type
pd.Series

socceraction.vaep.formula.**value**(*actions*, *Pscores*, *Pconcedes*)

Compute the offensive, defensive and VAEP value of each action.

The total VAEP value of an action is the difference between that action's offensive value and defensive value.

$$V_{VAEP}(a_i) = \Delta P_{score}(a_i, t) - \Delta P_{concede}(a_i, t)$$

Parameters

- **actions** (*pd.DataFrame*) – SPADL action.
- **Pscores** (*pd.Series*) – The probability of scoring from each corresponding game state.
- **Pconcedes** (*pd.Series*) – The probability of conceding from each corresponding game state.

Returns
The 'offensive_value', 'defensive_value' and 'vaep_value' of each action.

Return type
pd.DataFrame

See also:

offensive_value()
The offensive value

defensive_value()
The defensive value

CHAPTER

ELEVEN

SOCCERACTION.ATOMIC.SPADL

11.1 Converters

socceraction.atomic.spadl.convert_to_atomic	Convert regular SPADL actions to atomic actions.

11.1.1 socceraction.atomic.spadl.convert_to_atomic

socceraction.atomic.spadl.**convert_to_atomic**(*actions*)

Convert regular SPADL actions to atomic actions.

Parameters
actions (*pd.DataFrame*) – A SPADL dataframe.

Returns
The Atomic-SPADL dataframe.

Return type
pd.DataFrame

11.2 Schema

socceraction.atomic.spadl.AtomicSPADLSchema	Definition of an Atomic-SPADL dataframe.

11.2.1 socceraction.atomic.spadl.AtomicSPADLSchema

class socceraction.atomic.spadl.**AtomicSPADLSchema**(**args*, ***kwargs*)

Definition of an Atomic-SPADL dataframe.

Attributes

action_id
bodypart_id
bodypart_name
dx
dy
game_id
original_event_id
period_id
player_id
team_id
time_seconds
type_id
type_name
x
y

11.3 Config

socceraction.atomic.spadl.config.field_length	Convert a string or number to a floating point number, if possible.
socceraction.atomic.spadl.config.field_width	Convert a string or number to a floating point number, if possible.
socceraction.atomic.spadl.config.actiontypes	Built-in mutable sequence.
socceraction.atomic.spadl.config.bodyparts	Built-in mutable sequence.

11.3.1 socceraction.atomic.spadl.config.field_length

socceraction.atomic.spadl.config.**field_length = 105.0**

Convert a string or number to a floating point number, if possible.

11.3.2 socceraction.atomic.spadl.config.field_width

socceraction.atomic.spadl.config.**field_width = 68.0**

Convert a string or number to a floating point number, if possible.

11.3.3 socceraction.atomic.spadl.config.actiontypes

socceraction.atomic.spadl.config.**actiontypes = ['pass', 'cross', 'throw_in', 'freekick_crossed', 'freekick_short', 'corner_crossed', 'corner_short', 'take_on', 'foul', 'tackle', 'interception', 'shot', 'shot_penalty', 'shot_freekick', 'keeper_save', 'keeper_claim', 'keeper_punch', 'keeper_pick_up', 'clearance', 'bad_touch', 'non_action', 'dribble', 'goalkick', 'receival', 'interception', 'out', 'offside', 'goal', 'owngoal', 'yellow_card', 'red_card', 'corner', 'freekick']**

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

11.3.4 socceraction.atomic.spadl.config.bodyparts

socceraction.atomic.spadl.config.**bodyparts = ['foot', 'head', 'other', 'head/other', 'foot_left', 'foot_right']**

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

11.4 Utility functions

socceraction.atomic.spadl.play_left_to_right	Perform all action in the same playing direction.
socceraction.atomic.spadl.add_names	Add the type name, result name and bodypart name to an Atomic-SPADL dataframe.
socceraction.atomic.spadl.actiontypes_df	Return a dataframe with the type id and type name of each Atomic-SPADL action type.
socceraction.atomic.spadl.bodyparts_df	Return a dataframe with the bodypart id and bodypart name of each SPADL action type.

11.4.1 socceraction.atomic.spadl.play_left_to_right

socceraction.atomic.spadl.**play_left_to_right**(*actions*, *home_team_id*)

Perform all action in the same playing direction.

This changes the location of each action, such that all actions are performed as if the team that executes the action plays from left to right.

Parameters

- **actions** (*pd.DataFrame*) – The SPADL actins of a game.
- **home_team_id** (*int*) – The ID of the home team.

Returns
All actions performed left to right.

Return type
list(pd.DataFrame)

See also:

socceraction.atomic.vaep.features.play_left_to_right
For transforming gamestates.

11.4.2 socceraction.atomic.spadl.add_names

socceraction.atomic.spadl.**add_names**(*actions*)

Add the type name, result name and bodypart name to an Atomic-SPADL dataframe.

Parameters
actions (*pd.DataFrame*) – An Atomic-SPADL dataframe.

Returns
The original dataframe with a 'type_name', 'result_name' and 'bodypart_name' appended.

Return type
pd.DataFrame

11.4.3 socceraction.atomic.spadl.actiontypes_df

socceraction.atomic.spadl.**actiontypes_df**()

Return a dataframe with the type id and type name of each Atomic-SPADL action type.

Returns
The 'type_id' and 'type_name' of each Atomic-SPADL action type.

Return type
pd.DataFrame

11.4.4 socceraction.atomic.spadl.bodyparts_df

socceraction.atomic.spadl.**bodyparts_df**()

Return a dataframe with the bodypart id and bodypart name of each SPADL action type.

Returns
The 'bodypart_id' and 'bodypart_name' of each SPADL action type.

Return type
pd.DataFrame

CHAPTER

TWELVE

SOCCERACTION.ATOMIC.VAEP

Implements the Atomic-VAEP framework.

12.1 Model

socceraction.atomic.vaep.AtomicVAEP	An implementation of the VAEP framework for atomic actions.

12.1.1 socceraction.atomic.vaep.AtomicVAEP

class socceraction.atomic.vaep.**AtomicVAEP**(*xfns=None*, *nb_prev_actions=3*)

An implementation of the VAEP framework for atomic actions.

In contrast to the original VAEP framework[1] this extension distinguishes the contribution of the player who initiates the action (e.g., gives the pass) and the player who completes the action (e.g., receives the pass)[2].

Parameters

- **xfns** (*list*) – List of feature transformers (see *socceraction.atomic.vaep.features*) used to describe the game states. Uses `xfns_default` if None.
- **nb_prev_actions** (*int, default=3*) – Number of previous actions used to decscribe the game state.

See also:

socceraction.vaep.VAEP
Implementation of the original VAEP framework.

[1] Tom Decroos, Lotte Bransen, Jan Van Haaren, and Jesse Davis. "Actions speak louder than goals: Valuing player actions in soccer." In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1851-1861. 2019.

[2] Tom Decroos, Pieter Robberechts and Jesse Davis. "Introducing Atomic-SPADL: A New Way to Represent Event Stream Data". DTAI Sports Analytics Blog. https://dtai.cs.kuleuven.be/sports/blog/introducing-atomic-spadl:-a-new-way-to-represent-event-stream-data # noqa May 2020.

References

Methods

__init__

socceraction.atomic.vaep.AtomicVAEP.__init__

AtomicVAEP.**__init__**(*xfns=None*, *nb_prev_actions=3*)

12.2 Utility functions

socceraction.atomic.vaep.features	Implements the feature tranformers of the VAEP framework.
socceraction.atomic.vaep.labels	Implements the label tranformers of the Atomic-VAEP framework.
socceraction.atomic.vaep.formula	Implements the formula of the Atomic-VAEP framework.

12.2.1 socceraction.atomic.vaep.features

Implements the feature tranformers of the VAEP framework.

socceraction.atomic.vaep.features.**actiontype_onehot**(*actions*)

Get the one-hot-encoded type of each action.

Parameters
actions (*Actions*) – The actions of a game.

Returns
A one-hot encoding of each action's type.

Return type
Features

socceraction.atomic.vaep.features.**direction**(*actions*)

Get the direction of the action as components of the unit vector.

Parameters
actions (*Actions*) – The actions of a game.

Returns
The x-component ('dx') and y-compoment ('mov_angle') of the unit vector of each action.

Return type
Features

socceraction.atomic.vaep.features.**feature_column_names**(*fs*, *nb_prev_actions=3*)

Return the names of the features generated by a list of transformers.

Parameters

- **fs** (*list(callable)*) – A list of feature transformers.
- **nb_prev_actions** (*int, default=3 # noqa: DAR103*) – The number of previous actions included in the game state.

Returns
The name of each generated feature.

Return type
list(str)

socceraction.atomic.vaep.features.**goalscore**(*gamestates*)

Get the number of goals scored by each team after the action.

Parameters
gamestates (*GameStates*) – The gamestates of a game.

Returns
The number of goals scored by the team performing the last action of the game state ('goalscore_team'), by the opponent ('goalscore_opponent'), and the goal difference between both teams ('goalscore_diff').

Return type
Features

socceraction.atomic.vaep.features.**location**(*actions*)

Get the location where each action started.

Parameters
actions (*Actions*) – The actions of a game.

Returns
The 'x' and 'y' location of each action.

Return type
Features

socceraction.atomic.vaep.features.**movement_polar**(*actions*)

Get the distance covered and direction of each action.

Parameters
actions (*Actions*) – The actions of a game.

Returns
The distance covered ('mov_d') and direction ('mov_angle') of each action.

Return type
Features

socceraction.atomic.vaep.features.**play_left_to_right**(*gamestates*, *home_team_id*)

Perform all action in the same playing direction.

This changes the start and end location of each action, such that all actions are performed as if the team plays from left to right.

Parameters

- **gamestates** (*GameStates*) – The game states of a game.
- **home_team_id** (*int*) – The ID of the home team.

Returns
The game states with all actions performed left to right.

Return type
list(pd.DataFrame)

socceraction.atomic.vaep.features.**polar**(*actions*)

Get the polar coordinates of each action's start location.

The center of the opponent's goal is used as the origin.

Parameters
actions (*Actions*) – The actions of a game.

Returns
The 'dist_to_goal' and 'angle_to_goal' of each action.

Return type
Features

12.2.2 socceraction.atomic.vaep.labels

Implements the label tranformers of the Atomic-VAEP framework.

socceraction.atomic.vaep.labels.**concedes**(*actions*, *nr_actions=10*)

Determine whether the team possessing the ball conceded a goal within the next x actions.

Parameters

- **actions** (*pd.DataFrame*) – The actions of a game.
- **nr_actions** (*int, default=10 # noqa: DAR103*) – Number of actions after the current action to consider.

Returns
A dataframe with a column 'concedes' and a row for each action set to True if a goal was conceded by the team possessing the ball within the next x actions; otherwise False.

Return type
pd.DataFrame

socceraction.atomic.vaep.labels.**goal_from_shot**(*actions*)

Determine whether a goal was scored from the current action.

This label can be use to train an xG model.

Parameters
actions (*pd.DataFrame*) – The actions of a game.

Returns
A dataframe with a column 'goal' and a row for each action set to True if a goal was scored from the current action; otherwise False.

Return type
pd.DataFrame

socceraction.atomic.vaep.labels.**scores**(*actions*, *nr_actions=10*)

Determine whether the team possessing the ball scored a goal within the next x actions.

Parameters

- **actions** (*pd.DataFrame*) – The actions of a game.
- **nr_actions** (*int, default=10 # noqa: DAR103*) – Number of actions after the current action to consider.

Returns
A dataframe with a column 'scores' and a row for each action set to True if a goal was scored by the team possessing the ball within the next x actions; otherwise False.

Return type
pd.DataFrame

12.2.3 socceraction.atomic.vaep.formula

Implements the formula of the Atomic-VAEP framework.

`socceraction.atomic.vaep.formula.`**`defensive_value`**(*actions*, *scores*, *concedes*)

Compute the defensive value of each action.

VAEP defines the *defensive value* of an action as the change in conceding probability.

$$\Delta P_{concede}(a_i, t) = P^k_{concede}(S_i, t) - P^k_{concede}(S_{i-1}, t)$$

where $P_{concede}(S_i, t)$ is the probability that team t which possesses the ball in state S_i will concede in the next 10 actions.

Parameters

- **actions** (*pd.DataFrame*) – SPADL action.
- **scores** (*pd.Series*) – The probability of scoring from each corresponding game state.
- **concedes** (*pd.Series*) – The probability of conceding from each corresponding game state.

Returns
The defensive value of each action.

Return type
pd.Series

`socceraction.atomic.vaep.formula.`**`offensive_value`**(*actions*, *scores*, *concedes*)

Compute the offensive value of each action.

VAEP defines the *offensive value* of an action as the change in scoring probability before and after the action.

$$\Delta P_{score}(a_i, t) = P^k_{score}(S_i, t) - P^k_{score}(S_{i-1}, t)$$

where $P_{score}(S_i, t)$ is the probability that team t which possesses the ball in state S_i will score in the next 10 actions.

Parameters

- **actions** (*pd.DataFrame*) – SPADL action.
- **scores** (*pd.Series*) – The probability of scoring from each corresponding game state.
- **concedes** (*pd.Series*) – The probability of conceding from each corresponding game state.

Returns
he ffensive value of each action.

Return type
pd.Series

socceraction.atomic.vaep.formula.**value**(*actions*, *Pscores*, *Pconcedes*)

Compute the offensive, defensive and VAEP value of each action.

The total VAEP value of an action is the difference between that action's offensive value and defensive value.

$$V_{VAEP}(a_i) = \Delta P_{score}(a_i, t) - \Delta P_{concede}(a_i, t)$$

Parameters

- **actions** (*pd.DataFrame*) – SPADL action.
- **Pscores** (*pd.Series*) – The probability of scoring from each corresponding game state.
- **Pconcedes** (*pd.Series*) – The probability of conceding from each corresponding game state.

Returns

The 'offensive_value', 'defensive_value' and 'vaep_value' of each action.

Return type

pd.DataFrame

See also:

offensive_value()

The offensive value

defensive_value()

The defensive value

CHAPTER

THIRTEEN

CONTRIBUTOR GUIDE

This document lays out guidelines and advice for contributing to this project. If you're thinking of contributing, please start by reading this document and getting a feel for how contributing to this project works. If you have any questions, feel free to reach out to either Tom Decroos, or Pieter Robberechts, the primary maintainers.

The guide is split into sections based on the type of contribution you're thinking of making.

13.1 Bug reports

Bug reports are hugely important! Before you raise one, though, please check through the GitHub issues, **both open and closed**, to confirm that the bug hasn't been reported before.

When filing an issue, make sure to answer these questions:

- Which Python version are you using?
- Which version of socceraction are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

13.2 Feature requests

Socceraction is not actively developed. It's primary use is to enable reproducability of our research. If you believe there is a feature missing, feel free to raise a feature request on the Issue Tracker, but please do be aware that the overwhelming likelihood is that your feature request will not be accepted.

13.3 Documentation contributions

Documentation improvements are always welcome! The documentation files live in the `docs/` directory of the codebase. They're written in reStructuredText, and use Sphinx to generate the full suite of documentation.

You do not have to setup a development environment to make small changes to the docs. Instead, you can edit files directly on GitHub and suggest changes.

When contributing documentation, please do your best to follow the style of the documentation files. This means a soft-limit of 79 characters wide in your text files and a semi-formal, yet friendly and approachable, prose style.

When presenting Python code, use single-quoted strings (`'hello'` instead of `"hello"`).

13.4 Code contributions

If you intend to contribute code, do not feel the need to sit on your contribution until it is perfectly polished and complete. It helps everyone involved for you to seek feedback as early as you possibly can. Submitting an early, unfinished version of your contribution for feedback can save you from putting a lot of work into a contribution that is not suitable for the project.

13.4.1 Setting up your development environment

You need Python 3.7.1+ and the following tools:

- Poetry
- Nox
- nox-poetry

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session.

```
$ poetry run python
```

13.4.2 Steps for submitting code

When contributing code, you'll want to follow this checklist:

1. Fork the repository on GitHub.
2. Run the tests to confirm they all pass on your system. If they don't, you'll need to investigate why they fail. If you're unable to diagnose this yourself, raise it as a bug report.
3. Write tests that demonstrate your bug or feature. Ensure that they fail.
4. Make your change.
5. Run the entire test suite again, confirming that all tests pass *including the ones you just added*.
6. Make sure your code follows the code style discussed below.
7. Send a GitHub Pull Request to the main repository's `master` branch. GitHub Pull Requests are the expected method of code collaboration on this project.

13.4.3 Testing the project

Download the test data:

```
$ poetry run python tests/datasets/download.py
```

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the `tests` directory, and are written using the pytest testing framework.

13.4.4 Code style

The socceraction codebase uses the PEP 8 code style. In addition, we have a few guidelines:

- Line-length can exceed 79 characters, to 100, when convenient.
- Line-length can exceed 100 characters, when doing otherwise would be *terribly* inconvenient.
- Always use single-quoted strings (e.g. `'#soccer'`), unless a single-quote occurs within the string.

To ensure all code conforms to this format. You can format the code using the pre-commit hooks.

```
$ nox --session=pre-commit
```

Docstrings are to follow the numpydoc guidelines.

13.4.5 Submitting changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though. We can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything.

CHAPTER

FOURTEEN

FIRST STEPS

Are you new to soccer event stream data and possession value frameworks? Check out our interactive explainer and watch Lotte Bransen's and Jan Van Haaren's presentation in Friends of Tracking. Once familiar with the basic concepts, you can move on to the *quickstart guide* or continue with the hands-on video tutorials of the Friends of Tracking series:

- **Valuing actions in soccer (video, slides)**
  This presentation expands on the content of the introductory presentation by discussing the technicalities behind the VAEP framework for valuing actions of soccer players as well as the content of the hands-on video tutorials in more depth.
- **Tutorial 1: Run pipeline (video, notebook, notebook on Google Colab)**
  This tutorial demonstrates the entire pipeline of ingesting the raw Wyscout match event data to producing ratings for soccer players. This tutorial touches upon the following four topics: downloading and preprocessing the data, valuing game states, valuing actions and rating players.
- **Tutorial 2: Generate features (video, notebook, notebook on Google Colab)**
  This tutorial demonstrates the process of generating features and labels. This tutorial touches upon the following three topics: exploring the data in the SPADL representation, constructing features to represent actions and constructing features to represent game states.
- **Tutorial 3: Learn models (video, notebook, notebook on Google Colab)**
  This tutorial demonstrates the process of splitting the dataset into a training set and a test set, learning baseline models using conservative hyperparameters for the learning algorithm, optimizing the hyperparameters for the learning algorithm and learning the final models.
- **Tutorial 4: Analyze models and results (video, notebook, notebook on Google Colab)**
  This tutorial demonstrates the process of analyzing the importance of the features that are included in the trained machine learning models, analyzing the predictions for specific game states, and analyzing the resulting player ratings.

Note: The video tutorials are based on version 0.2.0 of the socceraction library. If a more recent version of the library is installed, the code may need to be adapted.

CHAPTER

FIFTEEN

GETTING HELP

Having trouble? We'd like to help!

- Try the *FAQ* – it's got answers to many common questions.
- Looking for specific information? Try the genindex or modindex.
- Report bugs in our ticket tracker.

CHAPTER

SIXTEEN

CONTRIBUTING

Learn about the development process itself and about how you can contribute in our *developer guide*.

CHAPTER

SEVENTEEN

RESEARCH

If you make use of this package in your research, please consider citing the following papers.

- Tom Decroos, Lotte Bransen, Jan Van Haaren, and Jesse Davis. **"Actions speak louder than goals: Valuing player actions in soccer."** In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1851-1861. 2019.

  [pdf, bibtex]

- Maaike Van Roy, Pieter Robberechts, Tom Decroos, and Jesse Davis. **"Valuing on-the-ball actions in soccer: a critical comparison of xT and VAEP."** In Proceedings of the AAAI-20 Workshop on Artifical Intelligence in Team Sports. AI in Team Sports Organising Committee, 2020.

  [pdf, bibtex]

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

O

P

R

S

T

V

W

X