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 period: $S_i = \{a_{i-2}, a_{i-1}, a_i\}$. 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 $S_i = \{a_{i-2}, a_{i-1}, a_i\}$:

Skip to content latest

Transformer	Feature	Description
`actiontype()`	actiontype(_onehot)_ai	The (one-hot encoding) of the action's type.
`result()`	result(_onehot)_ai	The (one-hot encoding) of the action's result.
`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.
`startlocation()`	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_ai	The distance to the center of the goal from the action's start location.
	start_angle_to_goal_ai	The angle between the action's start location and center of the goal.
`endpolar()`	end_dist_to_goal_ai	The distance to the center of the goal from the action's end location.
	end_angle_to_goal_ai	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 a

Transformer	Feature	Description
`goalscore()`	goalscore_team	The number of goals scored by the team executing the action.
	goalscore_opponent	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 = {
  "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