Package ‘contextual’

Type Package
Title Simulation and Analysis of Contextual Multi-Armed Bandit Policies
Version 0.9.8.2
Maintainer Robin van Emden <robinvanemden@gmail.com>
Description Facilitates the simulation and evaluation of context-free and contextual multi-Armed Bandit policies or algorithms to ease the implementation, evaluation, and dissemination of both existing and new bandit algorithms and policies.
License GPL-3
Encoding UTF-8
LazyData true
RoxygenNote 6.1.1
Imports R6 (>= 2.3.0), data.table, R.devices, foreach, doParallel, iterators, Formula, rjson
Suggests testthat, RCurl, splitstackshape, covr, knitr, here, markdown, devtools, ggplot2, vdiffr
VignetteBuilder knitr
URL https://github.com/Nth-iteration-labs/contextual
BugReports https://github.com/Nth-iteration-labs/contextual/issues
NeedsCompilation no
Author Robin van Emden [aut, cre] (<https://orcid.org/0000-0001-5820-8638>), Maurits Kaptein [ctb] (<https://orcid.org/0000-0002-6316-7524>)
Depends R (>= 3.5.0)
Repository CRAN
Date/Publication 2019-07-08 18:20:03 UTC
R topics documented:

Agent ................................................. 3
Bandit ................................................... 5
BasicBernoulliBandit ................................ 6
BasicGaussianBandit ................................. 8
BootstrapTSPolicy ..................................... 9
clipr .................................................... 10
ContextualBernoulliBandit .......................... 10
ContextualBinaryBandit .............................. 12
ContextualEpochGreedyPolicy ...................... 13
ContextualEpsilonGreedyPolicy .................... 13
ContextualHybridBandit .............................. 14
ContextualLinearBandit .............................. 16
ContextualLinTSPolicy ............................... 17
ContextualLogitBandit .............................. 19
ContextualLogitBTSPolicy ........................... 20
ContextualPrechainingBandit ....................... 21
ContextualTSPbiProbitPolicy ....................... 22
ContextualWheelBandit .............................. 23
ContinuumBandit ...................................... 24
data_table_factors_to_numeric .................... 26
dec<- .................................................. 26
EpsilonFirstPolicy .................................... 27
EpsilonGreedyPolicy ................................... 28
Exp3Policy ............................................. 30
FixedPolicy ............................................ 31
formatted_difftime .................................. 32
get_arm_context ....................................... 33
get_full_context ....................................... 33
GittinsBrezziLaiPolicy .............................. 34
GradientPolicy ........................................ 35
History .................................................... 37
inc<- ................................................... 39
ind ...................................................... 39
inv ...................................................... 40
invgamma .............................................. 40
invlogit ............................................... 41
is_rstudio ............................................. 42
LifPolicy .............................................. 42
LinUCBDisjointOptimizedPolicy .................... 43
LinUCBDisjointPolicy ................................ 45
LinUCBGeneralPolicy .................................. 46
LinUCBHybridOptimizedPolicy ...................... 47
LinUCBHybridPolicy ................................... 48
mvrnorm ................................................. 49
OfflineBootstrappedReplayBandit ................... 50
OfflineDirectMethodBandit ........................... 52
Agent

<table>
<thead>
<tr>
<th>Agent</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OfflineDoublyRobustBandit</td>
<td>55</td>
</tr>
<tr>
<td>OfflineLookupReplayEvaluatorBandit</td>
<td>57</td>
</tr>
<tr>
<td>OfflinePropensityWeightingBandit</td>
<td>61</td>
</tr>
<tr>
<td>OfflineReplayEvaluatorBandit</td>
<td>63</td>
</tr>
<tr>
<td>ones_in_zeroes</td>
<td>66</td>
</tr>
<tr>
<td>one_hot</td>
<td>67</td>
</tr>
<tr>
<td>OraclePolicy</td>
<td>68</td>
</tr>
<tr>
<td>Plot</td>
<td>69</td>
</tr>
<tr>
<td>plot.history</td>
<td>71</td>
</tr>
<tr>
<td>Policy</td>
<td>72</td>
</tr>
<tr>
<td>print.history</td>
<td>74</td>
</tr>
<tr>
<td>prob_winner</td>
<td>74</td>
</tr>
<tr>
<td>RandomPolicy</td>
<td>75</td>
</tr>
<tr>
<td>sample_one_of</td>
<td>76</td>
</tr>
<tr>
<td>set_external</td>
<td>77</td>
</tr>
<tr>
<td>sherman_morrisson</td>
<td>78</td>
</tr>
<tr>
<td>Simulator</td>
<td>78</td>
</tr>
<tr>
<td>sim_post</td>
<td>80</td>
</tr>
<tr>
<td>SoftmaxPolicy</td>
<td>81</td>
</tr>
<tr>
<td>summary.history</td>
<td>83</td>
</tr>
<tr>
<td>sum_of</td>
<td>83</td>
</tr>
<tr>
<td>ThompsonSamplingPolicy</td>
<td>84</td>
</tr>
<tr>
<td>UCB1Policy</td>
<td>85</td>
</tr>
<tr>
<td>UCB2Policy</td>
<td>87</td>
</tr>
<tr>
<td>value_remaining</td>
<td>88</td>
</tr>
<tr>
<td>var_welford</td>
<td>89</td>
</tr>
<tr>
<td>which_max_list</td>
<td>89</td>
</tr>
<tr>
<td>which_max_tied</td>
<td>90</td>
</tr>
</tbody>
</table>

Index 91

Description

Keeps track of one Bandit and Policy pair.

Details

Controls the running of one Bandit and Policy pair over \( t = 1, \ldots, T \) looping over consecutively, \texttt{bandit$\text{get\_context}()}, \texttt{policy$\text{get\_action}()}, \texttt{bandit$\text{get\_reward}()} and \texttt{policy$\text{set\_reward}()} for each time step \( t \).
**Schematic**

Usage

```r
agent <- Agent$new(policy, bandit, name=NULL, sparse = 0.0)
```

Arguments

- `policy` *Policy* instance.
- `bandit` *Bandit* instance.
- `name` character; sets the name of the Agent. If NULL (default), Agent generates a name based on its *Policy* instance’s name.
- `sparse` numeric; artificially reduces the data size by setting a sparsity level for the current *Bandit* and *Policy* pair. When set to a value between 0.0 (default) and 1.0 only a fraction `sparse` of the *Bandit*’s data is randomly chosen to be available to improve the Agent’s *Policy* through `policy$set_reward`.

Methods

- `new()` generates and instantializes a new Agent instance.
- `do_step()` advances a simulation by one time step by consecutively calling `bandit$get_context()`, `policy$get_action()`, `bandit$get_reward()` and `policy$set_reward()`. Returns a list of lists containing context, action, reward and theta.
- `set_t(t)` integer; sets the current time step to `t`.
- `get_t()` returns current time step `t`.

See Also

- Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
- Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
- Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
## Not run:
policy  <- EpsilonGreedyPolicy$new(epsilon = 0.1)
bandit  <- BasicBernoulliBandit$new(weights = c(0.6, 0.1, 0.1))
agent   <- Agent$new(policy, bandit, name = "E.G.", sparse = 0.5)
```
Bandit

## Description

Parent or superclass of all {contextual} Bandit subclasses.

## Details

In {contextual}, Bandits are responsible for the generation of (either synthetic or offline) contexts and rewards.

On initialisation, a Bandit subclass has to define the number of arms `self$k` and the number of contextual feature dimensions `self$d`.

For each `t = 1, ..., T` a Bandit then generates a list containing current context in `d x k` dimensional matrix `context$X`, the number of arms in `context$k` and the number of features in `context$d`.

Note: in context-free scenario’s, `context$X` can be omitted.

On receiving the index of a Policy-chosen arm through `action$choice`, Bandit is expected to return a named list containing at least `reward$reward` and, where computable, `reward$optimal`.

## Usage

```r
bandit <- Bandit$new()
```
Methods

new() generates and instantializes a new Bandit instance.

get_context(t) argument:

  • t: integer, time step t.

  returns a named list containing the current d x k dimensional matrix context$X$, the number of arms context$k$ and the number of features context$d$.

get_reward(t, context, action) arguments:

  • t: integer, time step t.
  • context: list, containing the current context$X$ (d x k context matrix), context$k$ (number of arms) and context$d$ (number of context features) (as set by bandit).
  • action: list, containing action$choice$ (as set by policy).

  returns a named list containing reward$reward$ and, where computable, reward$optimal$ (used by "oracle" policies and to calculate regret).

post_initialization() Called after class and seed initialisation, but before the start of the simulation. Set random values that remain available throughout the life of a Bandit here.

generate_bandit_data() Called after class and seed initialisation, but before the start of a simulation. Pregenerate contexts and rewards here.

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

---

BasicBernoulliBandit  Bandit: BasicBernoulliBandit

Description

Context-free Bernoulli or Binary multi-armed bandit.

Details

Simulates k Bernoulli arms where each arm issues a reward of one with uniform probability \( p \), and otherwise a reward of zero.

In a bandit scenario, this can be used to simulate a hit or miss event, such as if a user clicks on a headline, ad, or recommended product.

Usage

```r
bandit <- BasicBernoulliBandit$new(weights)
```
BasicBernoulliBandit

Arguments

weights numeric vector; probability of reward values for each of the bandit’s k arms

Methods

new(weights) generates and instantializes a new BasicBernoulliBandit instance.

get_context(t) argument:

• t: integer, time step t.

  returns a named list containing the current d x k dimensional matrix context$X, the number of arms context$k and the number of features context$d.

get_reward(t, context, action) arguments:

• t: integer, time step t.

• context: list, containing the current context$X (d x k context matrix), context$k (number of arms) and context$d (number of context features) (as set by bandit).

• action: list, containing action$choice (as set by policy).

  returns a named list containing reward$reward and, where computable, reward$optimal (used by "oracle" policies and to calculate regret).

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
## Not run:

horizon <- 100
sims <- 100

policy <- EpsilonGreedyPolicy$new(epsilon = 0.1)

bandit <- BasicBernoulliBandit$new(weights = c(0.6, 0.1, 0.1))
agent <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, sims)$run()

plot(history, type = "cumulative", regret = TRUE)

## End(Not run)
```
BasicGaussianBandit

Description

Context-free Gaussian multi-armed bandit.

Details

Simulates \( k \) Gaussian arms where each arm models the reward as a normal distribution with provided mean \( \mu \) and standard deviation \( \sigma \).

Usage

\[
\text{bandit} \leftarrow \text{BasicGaussianBandit$new(mu\_per\_arm, sigma\_per\_arm)\right}
\]

Arguments

\begin{itemize}
  \item \texttt{mu\_per\_arm} numeric vector; mean \( \mu \) for each of the bandit's \( k \) arms
  \item \texttt{sigma\_per\_arm} numeric vector; standard deviation of additive Gaussian noise for each of the bandit's \( k \) arms. Set to zero for no noise.
\end{itemize}

Methods

\texttt{new(mu\_per\_arm, sigma\_per\_arm)} generates and instantializes a new \texttt{BasicGaussianBandit} instance.
\texttt{get\_context(t)} argument:
  \begin{itemize}
    \item \( t \): integer, time step \( t \).
  \end{itemize}
returns a named list containing the current \( d \times k \) dimensional matrix \texttt{context}$\_X$, the number of arms \texttt{context}$\_k$ and the number of features \texttt{context}$\_d$.
\texttt{get\_reward(t, context, action)} arguments:
  \begin{itemize}
    \item \( t \): integer, time step \( t \).
    \item \texttt{context}: list, containing the current \texttt{context}$\_X$ (\( d \times k \) context matrix), \texttt{context}$\_k$ (number of arms) and \texttt{context}$\_d$ (number of context features) (as set by \texttt{bandit}).
    \item \texttt{action}: list, containing \texttt{action}$\_choice$ (as set by \texttt{policy}).
  \end{itemize}
returns a named list containing \texttt{reward}$\_reward$ and, where computable, \texttt{reward}$\_optimal$ (used by "oracle" policies and to calculate regret).

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualTSPolicy
Examples

```r
## Not run:

horizon      <- 100
sims         <- 100

policy       <- EpsilonGreedyPolicy$new(epsilon = 0.1)

bandit       <- BasicGaussianBandit$new(c(0,0,1), c(1,1,1))
agent         <- Agent$new(policy, bandit)

history      <- Simulator$new(agent, horizon, sims)$run()

plot(history, type = "cumulative", regret = TRUE)

## End(Not run)
```

BootstrapTSPolicy  Policy: Thompson sampling with the online bootstrap

Description

Bootstrap Thompson Sampling

Details

Bootstrap Thompson Sampling (BTS) is a heuristic method for solving bandit problems which modifies Thompson Sampling (see ThompsonSamplingPolicy) by replacing the posterior distribution used in Thompson sampling by a bootstrap distribution.

Usage

```r
policy <- BootstrapTSPolicy(J = 100, a = 1, b = 1)
policy <- BootstrapTSPolicy(1000)
```

Arguments

- `new(J = 100, a = 1, b = 1)` Generates a new BootstrapTSPolicy object. Arguments are defined in the Argument section above.

- `set.parameters()` each policy needs to assign the parameters it wants to keep track of to list `self$theta_to_arms` that has to be defined in `set.parameters()`’s body. The parameters defined here can later be accessed by arm index in the following way: `theta[[index_of_arm]]$parameter_name`

- `get.action(context)` here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

- `set.rewards(reward, context)` in `set.rewards(reward, context)`, a policy updates its parameter values based on the reward received, and, potentially, the current context.
References


See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

---

clipr

Clip vectors

Description

Clips values to a minimum and maximum value. That is, all values below the lower clamp value and the upper clamp value become the lower/upper value specified.

Usage

`clipr(x, min, max)`

Arguments

- `x` to be clipped vector
- `min` numeric. lowest value
- `max` numeric. highest value

---

ContextualBernoulliBandit

Bandit: Naive Contextual Bernoulli Bandit

Description

Contextual Bernoulli multi-armed bandit where at least one context feature is active at a time.

Usage

`bandit <- ContextualBernoulliBandit$new(weights)`
**ContextualBernoulliBandit**

**Arguments**

weights numeric matrix; \(d \times k\) matrix with probabilities of reward for \(d\) contextual features per \(k\) arms

**Methods**

new(weights) generates and initializes a new ContextualBernoulliBandit instance.

get_context(t) argument:

- \(t\): integer, time step \(t\).

  returns a named list containing the current \(d \times k\) dimensional matrix context\$X, the number of arms context\$k and the number of features context\$d.

get_reward(t, context, action) arguments:

- \(t\): integer, time step \(t\).
- context: list, containing the current context\$X (d x k context matrix), context\$k (number of arms) and context\$d (number of context features) (as set by bandit).
- action: list, containing action\$choice (as set by policy).

  returns a named list containing reward\$reward and, where computable, reward\$optimal (used by "oracle" policies and to calculate regret).

**See Also**

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: ContextualBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

**Examples**

```r
## Not run:

horizon <- 100
sims <- 100

policy <- EpsilonGreedyPolicy$new(epsilon = 0.1)

bandit <- ContextualBernoulliBandit$new(weights = c(0.6, 0.1, 0.1))
age <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, sims)$run()

plot(history, type = "cumulative", regret = TRUE)
```

## End(Not run)
ContextualBinaryBandit

Bandit: ContextualBinaryBandit

Description

Contextual Bernoulli multi-armed bandit where at least one context feature is active at a time.

Usage

```r
bandit <- ContextualBinaryBandit$new(weights)
```

Arguments

- `weights`: numeric matrix; `d x k` matrix with probabilities of reward for `d` contextual features per `k` arms

Methods

- `new(weights)` generates and initializes a new `ContextualBinaryBandit` instance.
- `get_context(t)` argument:
  - `t`: integer, time step `t`.
    - returns a named list containing the current `d x k` dimensional matrix `context$X`, the number of arms `context$k` and the number of features `context$d`.
- `get_reward(t, context, action)` arguments:
  - `t`: integer, time step `t`.
  - `context`: list, containing the current `context$X` (`d x k` context matrix), `context$k` (number of arms) and `context$d` (number of context features) (as set by `bandit`).
  - `action`: list, containing `action$choice` (as set by `policy`).
    - returns a named list containing `reward$reward` and, where computable, `reward$optimal` (used by "oracle" policies and to calculate regret).

See Also

Core contextual classes: `Bandit, Policy, Simulator, Agent, History, Plot`

Bandit subclass examples: `ContextualBinaryBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit`

Policy subclass examples: `EpsilonGreedyPolicy, ContextualLinTSPolicy`
Examples

```r
## Not run:

horizon <- 100
sims <- 100

policy <- EpsilonGreedyPolicy$new(epsilon = 0.1)
bandit <- ContextualBinaryBandit$new(weights = c(0.6, 0.1, 0.1))
agent <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, sims)$run()

plot(history, type = "cumulative", regret = TRUE)

## End(Not run)
```

---

**ContextualEpochGreedyPolicy**

Policy: A Time and Space Efficient Algorithm for Contextual Linear Bandits

**Description**

Policy: A Time and Space Efficient Algorithm for Contextual Linear Bandits

**Usage**

```r
policy <- EpsilonGreedyPolicy(epsilon = 0.1)
```

**See Also**

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

---

**ContextualEpsilonGreedyPolicy**

Policy: ContextualEpsilonGreedyPolicy with unique linear models

**Description**

Policy: ContextualEpsilonGreedyPolicy with unique linear models
Usage

```
policy <- ContextualEpsilonGreedyPolicy(epsilon = 0.1)
```

Arguments

```
epsilon  double, a positive real value \(R^+\)
```

Parameters

```
A  d*\(d\) identity matrix
b  a zero vector of length \(d\)
```

Methods

```
new(epsilon = 0.1)  Generates a new ContextualEpsilonGreedyPolicy object. Arguments are defined in the Argument section above.

set_parameters()  each policy needs to assign the parameters it wants to keep track of to list self$theta_to_arms that has to be defined in set_parameters()’s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name

get_action(context)  here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

set_reward(reward, context)  in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.
```

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

ContextualHybridBandit

```
Bandit: ContextualHybridBandit
```

Description

TODO: Optimization.

Details

Extension of ContextualLogitBandit modeling hybrid rewards with a combination of unique (or "disjoint") and shared contextual features.
Usage

\[
\text{bandit} \leftarrow \text{ContextualHybridBandit}\$\text{new}(k, \text{shared\_features}, \text{unique\_features}, \sigma = 1.0)
\]

Arguments

\[
k \quad \text{integer; number of bandit arms} \\
\text{shared\_features} \quad \text{integer; number of shared features} \\
\text{unique\_features} \quad \text{integer; number of unique/disjoint features} \\
\sigma \quad \text{integer; standard deviation of additive Gaussian noise}
\]

Methods

\[
\text{new}(k, \text{shared\_features}, \text{unique\_features}, \sigma = 1.0) \quad \text{generates and instantializes a new ContextualHybridBandit instance.}
\]

\[
\text{get\_context}(t) \quad \text{argument:} \\
\quad \text{t: integer, time step } t. \\
\quad \text{returns a named list containing the current } d \times k \text{ dimensional matrix context\$X, the number of arms context\$k and the number of features context\$d.}
\]

\[
\text{get\_reward}(t, \text{context}, \text{action}) \quad \text{arguments:} \\
\quad \text{t: integer, time step } t. \\
\quad \text{context: list, containing the current context\$X (d \times k context matrix), context\$k (number of arms) and context\$d (number of context features) (as set by bandit).} \\
\quad \text{action: list, containing action\$choice (as set by policy).} \\
\quad \text{returns a named list containing reward\$reward and, where computable, reward\$optimal (used by "oracle" policies and to calculate regret).}
\]

\[
\text{post\_initialization}() \quad \text{initializes } d \times k \beta \text{ matrix.}
\]

See Also

Core contextual classes: \texttt{Bandit, Policy, Simulator, Agent, History, Plot}  
Bandit subclass examples: \texttt{BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit}  
Policy subclass examples: \texttt{EpsilonGreedyPolicy, ContextualLinTSPolicy}

Examples

\[
\text{## Not run:}
\]

\[
\text{horizon} \quad \leftarrow 800L \\
\text{simulations} \quad \leftarrow 100L \\
\text{bandit} \quad \leftarrow \text{ContextualHybridBandit}\$\text{new}(k = 100, \text{shared\_features} = 10, \text{unique\_features} = 2) \\
\text{agents} \quad \leftarrow \text{list}(\text{Agent}\$\text{new}(\text{ContextualLinTSPolicy}\$\text{new}(0.1), \text{bandit}), \\
\hspace{1cm} \text{Agent}\$\text{new}(\text{EpsilonGreedyPolicy}\$\text{new}(0.1), \text{bandit}), \\
\hspace{1cm} \text{Agent}\$\text{new}(\text{LinUCBGeneralPolicy}\$\text{new}(0.6), \text{bandit}), \\
\hspace{1cm} \text{Agent}\$\text{new}(\text{ContextualEpochGreedyPolicy}\$\text{new}(8), \text{bandit}),)
\]

\[
\]
```r
Agent$new(LinUCBHybridOptimizedPolicy$new(0.6), bandit),
Agent$new(LinUCBDisjointOptimizedPolicy$new(0.6), bandit))

simulation <- Simulator$new(agents, horizon, simulations)
history <- simulation$run()

plot(history, type = "cumulative", regret = FALSE, rate = TRUE, legend_position = "bottomright")
## End(Not run)
```

---

**ContextualLinearBandit**

*Bandit: ContextualLinearBandit*

---

**Description**

Samples data from linearly parameterized arms.

**Details**

The reward for context X and arm j is given by X^T beta_j, for some latent set of parameters beta_j: j = 1, ..., k. The beta’s are sampled uniformly at random, the contexts are Gaussian, and sigma-noise is added to the rewards.

**Usage**

```r
bandit <- ContextualLinearBandit$new(k, d, sigma = 0.1, binary_rewards = FALSE)
```

**Arguments**

- k: integer; number of bandit arms
- d: integer; number of contextual features
- sigma: numeric; standard deviation of the additive noise. Set to zero for no noise. Default is 0.1
- binary_rewards: logical; when set to FALSE (default) ContextualLinearBandit generates Gaussian rewards. When set to TRUE, rewards are binary (0/1).

**Methods**

- `new(k, d, sigma = 0.1, binary_rewards = FALSE)` generates and instantializes a new ContextualLinearBandit instance.
- `get_context(t)` argument:
  - t: integer, time step t.
    - returns a named list containing the current d x k dimensional matrix context$X, the number of arms context$k and the number of features context$d.
- `get_reward(t, context, action)` arguments:
• \(t\): integer, time step \(t\).
• \(\text{context}\): list, containing the current context \(X\) \((d \times k \text{ context matrix})\), \(\text{context}^k\) (number of arms) and context \(d\) (number of context features) (as set by bandit).
• \(\text{action}\): list, containing action \(\text{choice}\) (as set by policy).

returns a named list containing reward \(\text{reward}\) and, where computable, reward \(\text{optimal}\) (used by "oracle" policies and to calculate regret).

post_initialization() initializes \(d \times k\) beta matrix.

References


Implementation follows https://github.com/tensorflow/models/tree/master/research/deep_contextual_bandits

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
# Not run:
horizon <- 800L
simulations <- 30L
bandit <- ContextualLinTSPolicy$new(k = 5, d = 5)
agents <- list(Agent$new(EpsilonGreedyPolicy$new(0.1), bandit),
               Agent$new(LinUCBDisjointOptimizedPolicy$new(0.6), bandit))
simulation <- Simulator$new(agents, horizon, simulations)
history <- simulation$run()

plot(history, type = "cumulative", regret = FALSE, rate = TRUE, legend_position = "right")
# End(Not run)
```

ContextualLinTSPolicy  Policy: Linear Thompson Sampling with unique linear models

Description

ContextualLinTSPolicy implements Thompson Sampling with Linear Payoffs, following Agrawal and Goyal (2011). Thompson Sampling with Linear Payoffs is a contextual Thompson Sampling multi-armed bandit Policy which assumes the underlying relationship between rewards and contexts are linear. Check the reference for more details.
Usage

    policy <- ContextualLinTSPolicy$new(v = 0.2)

Arguments

    v double, a positive real value R+: Hyper-parameter for adjusting the variance of posterior gaussian distribution.

Methods

    new(v) instantiates a new ContextualLinTSPolicy instance. Arguments defined in the Arguments section above.

    set_parameters(context_params) initialization of policy parameters, utilising context_params$k (number of arms) and context_params$d (number of context features).

    get_action(t, context) selects an arm based on self$theta and context, returning the index of the selected arm in action$choice. The context argument consists of a list with context$k (number of arms), context$d (number of features), and the feature matrix context$x with dimensions $d \times k$.

    set_reward(t, context, action, reward) updates parameter list theta in accordance with the current reward$reward, action$choice and the feature matrix context$x with dimensions $d \times k$. Returns the updated theta.

References


See Also

    Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

    Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

    Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

    ## Not run:

    horizon <- 100L
    simulations <- 100L

    bandit <- ContextualLinearBandit$new(k = 4, d = 3, sigma = 0.3)

    agents <- list(Agent$new(EpsilonGreedyPolicy$new(0.1), bandit, "EGreedy"),
                    Agent$new(ContextualLinTSPolicyPolicy$new(0.1), bandit, "LinTSPolicy"))

    simulation <- Simulator$new(agents, horizon, simulations, do_parallel = TRUE)
**ContextualLogitBandit**

```r
history <- simulation$run()
plot(history, type = "cumulative", rate = FALSE, legend_position = "topleft")
```

```r
# End(Not run)
```

**Description**

Samples data from a basic logistic regression model.

**Details**

ContextualLogitBandit linear predictors are generated from the dot product of a random $d$ dimensional normal weight vector and uniform random $d \times k$ dimensional context matrices with equal weights per arm. This product is then inverse-logit transformed to generate $k$ dimensional binary (0/1) reward vectors by randomly sampling from a Bernoulli distribution.

**Usage**

```r
bandit <- ContextualLogitBandit$new(k, d, intercept = TRUE)
```

**Arguments**

- **k** integer; number of bandit arms
- **d** integer; number of contextual features
- **intercept** logical; if TRUE (default) it adds a constant (1.0) dimension to each context $X$ at the end.

**Methods**

- **new(k, d, intercept = TRUE)** generates and instantialsizes a new `ContextualLogitBandit` instance.
- **get_context(t)** argument:
  - **t**: integer, time step $t$.
  - returns a named list containing the current $d \times k$ dimensional matrix `context$X`, the number of arms `context$k` and the number of features `context$d`.
- **get_reward(t, context, action)** arguments:
  - **t**: integer, time step $t$.
  - **context**: list, containing the current `context$X` ($d \times k$ context matrix), `context$k` (number of arms) and `context$d` (number of context features) (as set by `bandit`).
  - **action**: list, containing `action$choice` (as set by policy).
  - returns a named list containing `reward$reward` and, where computable, `reward$optimal` (used by "oracle" policies and to calculate regret).
- **post_initialization()** initializes $d \times k$ beta matrix.
See Also

Core contextual classes: `Bandit, Policy, Simulator, Agent, History, Plot`
Bandit subclass examples: `BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit`
Policy subclass examples: `EpsilonGreedyPolicy, ContextualLinTS Policy`

Examples

```r
## Not run:

horizon <- 800L
simulations <- 30L

bandit <- ContextualLogitBandit$new(k = 5, d = 5, intercept = TRUE)

agents <- list(Agent$new(ContextualLinTS Policy$new(0.1), bandit),
                Agent$new(EpsilonGreedyPolicy$new(0.1), bandit),
                Agent$new(LinUCBGeneralPolicy$new(0.6), bandit),
                Agent$new(ContextualEpochGreedyPolicy$new(8), bandit),
                Agent$new(LinUCBHybridOptimizedPolicy$new(0.6), bandit),
                Agent$new(LinUCBDisjointOptimizedPolicy$new(0.6), bandit))

simulation <- Simulator$new(agents, horizon, simulations)
history <- simulation$run()

plot(history, type = "cumulative", regret = FALSE,
      rate = TRUE, legend_position = "right")

## End(Not run)
```

ContextualLogitBTSPolicy

*Policy: ContextualLogitBTSPolicy*

Description

Policy: ContextualLogitBTSPolicy

Usage

`policy <- ContextualLogitBTSPolicy()`

See Also

Core contextual classes: `Bandit, Policy, Simulator, Agent, History, Plot`
Bandit subclass examples: `BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit`
Policy subclass examples: `EpsilonGreedyPolicy, ContextualLinTS Policy`
ContextualPrecachingBandit

Bandit: ContextualPrecachingBandit

Description
Illustrates precaching of contexts and rewards.

Details
TODO: Fix "attempt to select more than one element in integerOneIndex"
Contextual extension of BasicBernoulliBandit.
Contextual extension of BasicBernoulliBandit where a user specified d x k dimensional matrix takes the place of BasicBernoulliBandit k dimensional probability vector. Here, each row d represents a feature with k reward probability values per arm.
For every \( t \), ContextualPrecachingBandit randomly samples from its d features/rows at random, yielding a binary context matrix representing sampled (all 1 rows) and unsampled (all 0) features/rows. Next, ContextualPrecachingBandit generates rewards contingent on either sum or mean (default) probabilities of each arm/column over all of the sampled features/rows.

Usage

```R
bandit <- ContextualPrecachingBandit$new(weights)
```

Arguments

weights numeric matrix; d x k dimensional matrix where each row d represents a feature with k reward probability values per arm.

Methods

new(weights) generates and instantializes a new ContextualPrecachingBandit instance.
get_context(t) argument:
  • t: integer, time step t.
    returns a named list containing the current d x k dimensional matrix context\(X\), the number of arms context\(k\) and the number of features context\(d\).
get_reward(t, context, action) arguments:
  • t: integer, time step t.
  • context: list, containing the current context\(X\) (d x k context matrix), context\(k\) (number of arms) and context\(d\) (number of context features) (as set by bandit).
  • action: list, containing action\$choice (as set by policy).
    returns a named list containing reward\$reward and, where computable, reward\$optimal (used by "oracle" policies and to calculate regret).
generate_bandit_data() helper function called before Simulator starts iterating over all time steps \( t \) in \( T \). Pregenerates contexts and rewards.
See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTS Policy

Examples

```r
## Not run:

horizon <- 100L
simulations <- 100L

# rows represent features, columns represent arms:
context_weights <- matrix(c(0.4, 0.2, 0.4,
                             0.3, 0.4, 0.3,
                             0.1, 0.8, 0.1), nrow = 3, ncol = 3, byrow = TRUE)

bandit <- ContextualPreCachingBandit$new(weights)

agents <- list(
  Agent$new(EpsilonGreedyPolicy$new(0.1), bandit),
  Agent$new(LinUCBDisjointOptimizedPolicy$new(0.6), bandit))

simulation <- Simulator$new(agents, horizon, simulations)

history <- simulation$run()

plot(history, type = "cumulative")

## End(Not run)
```

---

ContextualTSProbitPolicy

*Policy: ContextualTSProbitPolicy*

**Description**

Makes use of BOPR, ergo only use binary independent variables.

**Usage**

```r
policy <- ContextualTSProbitPolicy()
```

**See Also**

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTS Policy
ContextualWheelBandit

Bandit: ContextualWheelBandit

Description

Samples from Wheel bandit game.

Details

The Wheel bandit game offers an artificial problem where the need for exploration is smoothly parameterized through exploration parameter \( \delta \).

In the game, contexts are sampled uniformly at random from a unit circle divided into one central and four edge areas for a total of \( k = 5 \) possible actions. The central area offers a random normal sampled reward independent of the context, in contrast to the outer areas which offer a random normal sampled reward dependent on a \( d = 2 \) dimensional context.

For more information, see https://arxiv.org/abs/1802.09127.

Usage

```r
bandit <- ContextualWheelBandit$new(delta, mean_v, std_v, mu_large, std_large)
```

Arguments

- \( \delta \) numeric; exploration parameter: high reward in one region if norm above \( \delta \).
- \( \text{mean}_v \) numeric vector; mean reward for each action if context norm is below \( \delta \).
- \( \text{std}_v \) numeric vector; gaussian reward sd for each action if context norm is below \( \delta \).
- \( \mu_{\text{large}} \) numeric; mean reward for optimal action if context norm is above \( \delta \).
- \( \text{std}_{\text{large}} \) numeric; standard deviation of the reward for optimal action if context norm is above \( \delta \).

Methods

- `new(delta, mean_v, std_v, mu_large, std_large)` generates and instantializes a new `ContextualWheelBandit` instance.
- `get_context(t)` argument:
  - \( t \): integer, time step \( t \).
  - returns a named list containing the current \( d \times k \) dimensional matrix `context$X`, the number of arms `context$k` and the number of features `context$d`.
- `get_reward(t, context, action)` arguments:
  - \( t \): integer, time step \( t \).
  - context: list, containing the current `context$X` (\( d \times k \) context matrix), `context$k` (number of arms) and `context$d` (number of context features) (as set by `bandit`).
  - action: list, containing `action$choice` (as set by policy).
  - returns a named list containing `reward$reward` and, where computable, `reward$optimal` (used by “oracle” policies and to calculate regret).
ContinuumBandit

References


Implementation follows https://github.com/tensorflow/models/tree/master/research/deep_contextual_bandits

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
## Not run:

horizon <- 1000L
simulations <- 10L

delta <- 0.95
num_actions <- 5
context_dim <- 2
mean_v <- c(1.0, 1.0, 1.0, 1.0, 1.2)
std_v <- c(0.05, 0.05, 0.05, 0.05, 0.05)
mu_large <- 50
std_large <- 0.01

bandit <- ContextualWheelBandit$new(delta, mean_v, std_v, mu_large, std_large)
agents <- list(Agent$new(UCB1Policy$new(), bandit), Agent$new(LinUCBDisjointOptimizedPolicy$new(0.6), bandit))

simulation <- Simulator$new(agents, horizon, simulations)

plot(history, type = "cumulative", regret = FALSE, rate = TRUE, legend_position = "bottomright")

## End(Not run)
```

ContinuumBandit

**Bandit: ContinuumBandit**

Description

A function based continuum multi-armed bandit where arms are chosen from a subset of the real line and the mean rewards are assumed to be a continuous function of the arms.

Usage

```r
bandit <- ContinuumBandit$new(FUN)
```
Arguments

**FUN** continuous function.

Methods

new(FUN) generates and instantiates a new ContinuumBandit instance.

get_context(t) argument:
- t: integer, time step t.
  
  returns a named list containing the current \( d \times k \) dimensional matrix \( \text{context}_X \), the number of arms \( \text{context}_k \) and the number of features \( \text{context}_d \).

get_reward(t, context, action) arguments:
- t: integer, time step t.
- context: list, containing the current \( \text{context}_X \) (\( d \times k \) context matrix), \( \text{context}_k \) (number of arms) and \( \text{context}_d \) (number of context features) (as set by bandit).
- action: list, containing \( \text{action}_\text{choice} \) (as set by policy).

returns a named list containing \( \text{reward}_\text{reward} \) and, where computable, \( \text{reward}_\text{optimal} \) (used by "oracle" policies and to calculate regret).

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
## Not run:

horizon <- 1500
simulations <- 100

continuous_arms <- function(x) {
  -0.1*(x - 5)^2 + 3.5 + rnorm(length(x),0,0.4)
}

int_time <- 100
amplitude <- 0.2
learn_rate <- 0.3
omega <- 2*pi/int_time
x0_start <- 2.0

policy <- LifPolicy$new(int_time, amplitude, learn_rate, omega, x0_start)

bandit <- ContinuumBandit$new(FUN = continuous_arms)

agent <- Agent$new(policy,bandit)

history <- Simulator$new(agents = agent,
```
horizon = horizon,
simulations = simulations,
save_theta = TRUE
)$run()

plot(history, type = "average", regret = FALSE)

## End(Not run)

data_table_factors_to_numeric

Convert all factor columns in data.table to numeric

Description
Convert all factor columns in data.table to numeric

Usage
data_table_factors_to_numeric(dt)

Arguments
dt a data.table

Value
the data.table with column factors converted to numeric

dec<- Decrement

Description
dec<- decrements x by value. Equivalent to x <- x - value.

Usage
dec(x) <- value

Arguments
x object to be decremented
value value by which x will be modified

Examples
x <- 6:10
dec(x) <- 5
x
EpsilonFirstPolicy

Policy: Epsilon First

Description

EpsilonFirstPolicy implements a "naive" policy where a pure exploration phase is followed by a pure exploitation phase.

Details

Exploration happens within the first $\epsilon \times N$ time steps. During this time, at each time step $t$, EpsilonFirstPolicy selects an arm at random.

Exploitation happens in the following $(1-\epsilon) \times N$ steps, selecting the best arm up until $\epsilon \times N$ for either the remaining $N$ trials or horizon $T$.

In case of a tie in the exploitation phase, EpsilonFirstPolicy randomly selects an arm.

Usage

```
policy <- EpsilonFirstPolicy(epsilon = 0.1, N = 1000, time_steps = NULL)
```

Arguments

- $\epsilon$ numeric; value in the closed interval $(0,1]$ that sets the number of time steps to explore through $\epsilon \times N$.
- $N$ integer; positive integer which sets the number of time steps to explore through $\epsilon \times N$.
- $\text{time_steps}$ integer; positive integer which sets the number of time steps to explore - can be used instead of $\epsilon$ and $N$.

Methods

- `new(epsilon = 0.1, N = 1000, time_steps = NULL)` generates a new EpsilonFirstPolicy object. Arguments are defined in the Argument section above.

- `set_parameters()` each policy needs to assign the parameters it wants to keep track of to list `self$\theta_to_arms` that has to be defined in `set_parameters()`'s body. The parameters defined here can later be accessed by arm index in the following way: `theta[[index_of_arm]]$parameter_name`

- `get_action(context)` here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

- `set_reward(reward, context)` in `set_reward(reward, context)`, a policy updates its parameter values based on the reward received, and, potentially, the current context.
EpsilonGreedyPolicy

References


See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
horizon <- 100L
simulations <- 100L
weights <- c(0.9, 0.1, 0.1)

policy <- EpsilonFirstPolicy$new(time_steps = 50)
bandit <- BasicBernoulliBandit$new(weights = weights)
agent <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, simulations, do_parallel = FALSE)$run()

plot(history, type = "cumulative")
plot(history, type = "arms")
```

EpsilonGreedyPolicy Policy: Epsilon Greedy

Description

EpsilonGreedyPolicy chooses an arm at random (explores) with probability epsilon, otherwise it greedily chooses (exploits) the arm with the highest estimated reward.

Usage

```r
policy <- EpsilonGreedyPolicy(epsilon = 0.1)
```
EpsilonGreedyPolicy

Arguments

epsilon numeric; value in the closed interval (0,1] indicating the probability with which arms are selected at random (explored). Otherwise, EpsilonGreedyPolicy chooses the best arm (exploits) with a probability of $1 - \epsilon$

name character string specifying this policy. name is, among others, saved to the History log and displayed in summaries and plots.

Methods

new(epsilon = 0.1) Generates a new EpsilonGreedyPolicy object. Arguments are defined in the Argument section above.

set_parameters() each policy needs to assign the parameters it wants to keep track of to list self$theta_to_arms that has to be defined in set_parameters()'s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name

get_action(context) here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

set_reward(reward, context) in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.

References


See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTS Policy

Examples

```r
horizon <- 100L
simulations <- 100L
weights <- c(0.9, 0.1, 0.1)

policy <- EpsilonGreedyPolicy$new(epsilon = 0.1)
bandit <- BasicBernoulliBandit$new(weights = weights)
```
Exp3Policy

agent <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, simulations, do_parallel = FALSE)$run()

plot(history, type = "cumulative")

plot(history, type = "arms")

---

Exp3Policy  Policy: Exp3

Description

In Exp3Policy, "Exp3" stands for "Exponential-weight algorithm for Exploration and Exploitation". It makes use of a distribution over probabilities that is a mixture of a uniform distribution and a distribution which assigns to each action a probability mass exponential in the estimated cumulative reward for that action.

Usage

policy <- Exp3Policy(gamma = 0.1)

Arguments

gamma  double, value in the closed interval (0,1], controls the exploration - often referred to as the learning rate

name  character string specifying this policy. name is, among others, saved to the History log and displayed in summaries and plots.

Methods

new(gamma = 0.1) Generates a new Exp3Policy object. Arguments are defined in the Argument section above.

set_parameters() each policy needs to assign the parameters it wants to keep track of to list self$theta_to_arms that has to be defined in set_parameters()'s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name

get_action(context) here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

set_reward(reward, context) in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.
**FixedPolicy**

**Description**

FixedPolicy implements a "naive" policy which always chooses a prespecified arm.

**Usage**

```
policy <- FixedPolicy(fixed_arm = 1)
```

**Arguments**

- `fixed_arm` numeric; index of the arm that will be chosen for each time step.

**References**


**See Also**

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

**Examples**

```r
horizon <- 100L
simulations <- 100L
weights <- c(0.9, 0.1, 0.1)

policy <- Exp3Policy$new(gamma = 0.1)
bandit <- BasicBernoulliBandit$new(weights = weights)
agent <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, simulations, do_parallel = FALSE)$run()

plot(history, type = "cumulative")

plot(history, type = "arms")
```
Methods

new() Generates a new FixedPolicy object. Arguments are defined in the Argument section above.

set_parameters() each policy needs to assign the parameters it wants to keep track of to list self$theta_to_arms that has to be defined in set_parameters()'s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name

get_action(context) here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

set_reward(reward, context) in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

formatted_difftime  Format difftime objects

Description

Format difftime objects

Usage

formatted_difftime(x)

Arguments

x  difftime object

Value

string "days, h:mm:ss.ms"
get_arm_context

Return context vector of an arm

Description

Given d x k matrix or d dimensional vector X, returns a vector with arm’s context.

Usage

get_arm_context(context, arm, select_features = NULL, prepend_arm_vector = FALSE)

Arguments

context a context list containing a d x k Matrix or d dimensional context vector X, the number of features d and number of arms k.

arm index of arm.

select_features indices of to be returned features.

prepend_arm_vector prepend a one-hot-encoded arm vector to the returned context vector. That is, when k = 5 arms, and the to be returned arm vector is arm 3, prepend c(0,0,1,0,0)

Value

Vector that represents context related to an arm

get_full_context

Get full context matrix over all arms

Description

Given matrix or d dimensional vector X, number of arms k and number of features d returns a matrix with d x k context matrix

Usage

get_full_context(context, select_features = NULL, prepend_arm_matrix = FALSE)
GittinsBrezziLaiPolicy

Policy: Gittins Approximation algorithm for choosing arms in a MAB problem.

Description


Details

The algorithm provides an approximation of the Gittins index, by specifying a closed-form expression, which is a function of the discount factor, and the number of successes and failures associated with each arm.

Usage

```r
policy <- GittinsBrezziLaiPolicy$new(discount=0.95, prior=NULL)
```

Arguments

discount numeric; discount factor

prior numeric matrix; prior beliefs over Bernoulli parameters governing each arm. Beliefs are specified by Beta distribution with two parameters (alpha, beta) where alpha = number of success, beta = number of failures. Matrix is of arms times two (alpha / beta) dimensions

Methods

new(discount=0.95, prior=NULL) Generates and initializes a new Policy object.

get_action(t, context) arguments:

- t: integer, time step t.
GradientPolicy 35

- context: list, containing the current context$X$ (d x k context matrix), context$k$ (number of arms) and context$d$ (number of context features)

computes which arm to play based on the current values in named list theta and the current context. Returns a named list containing action$choice$, which holds the index of the arm to play.

set_reward(t, context, action, reward) arguments:
- t: integer, time step t.
- context: list, containing the current context$X$ (d x k context matrix), context$k$ (number of arms) and context$d$ (number of context features) (as set by bandit).
- action: list, containing action$choice$ (as set by policy).
- reward: list, containing reward$reward$ and, if available, reward$optimal$ (as set by bandit).

utilizes the above arguments to update and return the set of parameters in list theta.

set_parameters() Helper function, called during a Policy’s initialisation, assigns the values it finds in list self$theta_to_arms$ to each of the Policy’s k arms. The parameters defined here can then be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name.

References


Implementation follows https://github.com/ellarry/bandit-algorithms-simulated

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

---

GradientPolicy  Policy: Gradient

Description

GradientPolicy is a SoftMax type algorithm, based on Sutton & Barton (2018).

Usage

```r
policy <- GradientPolicy(alpha = 0.1)
```

Arguments

- `alpha = 0.1` double, temperature parameter alpha specifies how many arms we can explore. When alpha is high, all arms are explored equally, when alpha is low, arms offering higher rewards will be chosen.
Methods

new(epsilon = 0.1) Generates a new GradientPolicy object. Arguments are defined in the Argument section above.

set_parameters() each policy needs to assign the parameters it wants to keep track of to list self$theta_to_arms that has to be defined in set_parameters()'s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name

get_action(context) here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

set_reward(reward, context) in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.

References


See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
horizon <- 100L
simulations <- 100L
weights <- c(0.9, 0.1, 0.1)

policy <- GradientPolicy$new(alpha = 0.1)
bandit <- BasicBernoulliBandit$new(weights = weights)
agent <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, simulations, do_parallel = FALSE)$run()

plot(history, type = "cumulative")

plot(history, type = "arms")
```
**Description**

The R6 class `History` keeps a log of all Simulator interactions in its internal `data.table`. It also provides basic data summaries, and can save or load simulation log data files.

**Usage**

```r
History <- History$new(n = 1, save_context = FALSE, save_theta = FALSE)
```

**Arguments**

- `n` integer. The number of rows, to be preallocated during initialization.
- `save_context` logical. Save context matrix `X` when writing simulation data?
- `save_theta` logical. Save parameter lists theta when writing simulation data?

**Methods**

- `reset()` Resets a `History` instance to its original initialisation values.
- `insert(index, t, action, reward, agent_name, simulation_index, context_value = NA, theta_value = NA)` Saves one row of simulation data. Is generally not called directly, but from a Simulator instance.
- `save(filename = NA)` Writes the `History` log file in its default `data.table` format, with `filename` as the name of the file which the data is to be written to.
- `load = function(filename, interval = 0)` Reads a `History` log file in its default `data.table` format, with `filename` as the name of the file which the data are to be read from. If `interval` is larger than 0, every `interval` of data is read instead of the full data file. This can be of use with (a first) analysis of very large data files.
- `get_data_frame()` Returns the `History` log as a `data.frame`.
- `set_data_frame(df, auto_stats = TRUE)` Sets the `History` log with the data in `data.frame` `dt`. Recalculates cumulative statistics when `auto_stats` is `TRUE`.
- `get_data_table()` Returns the `History` log as a `data.table`.
- `set_data_table(dt, auto_stats = TRUE)` Sets the `History` log with the data in `data.table` `dt`. Recalculates cumulative statistics when `auto_stats` is `TRUE`.
- `clear_data_table()` Clear `History`'s internal `data.table` log.
- `save_csv(filename = NA)` Saves `History` data to csv file.
- `extract_theta(limit_agents, parameter, arm, tail = NULL)` Extracts theta parameter from theta list for `limit_agents`, where `parameter` sets the to be retrieved parameter or vector of parameters in theta, `arm` is the relevant integer index of the arm or vector of arms of interest, and the optional `tail` selects the last elements in the list. Returns a vector, matrix or array with the selected theta values.
print_data() Prints a summary of the History log.
update_statistics() Updates cumulative statistics.
get_agent_list() Retrieve list of agents in History.
get_agent_count() Retrieve number of agents in History.
get_simulation_count() Retrieve number of simulations in History.
get_arm_choice_percentage(limit_agents) Retrieve list of percentage arms chosen per agent for limit_agents.
get_meta_data() Retrieve History meta data.
set_meta_data(key, value, group = "sim", agent_name = NULL) Set History meta data.
get_cumulative_data(limit_agents = NULL, limit_cols = NULL, interval = 1, cum_average = FALSE) Retrieve cumulative statistics data.
get_cumulative_result(limit_agents = NULL, limit_cols = NULL, interval = 1, cum_average = FALSE) Retrieve cumulative statistics data point.
save_theta_json(filename = "theta.json") Save theta in JSON format to a file. Warning: the theta log, and therefore the file, can get very large very fast.
get_theta(limit_agent, to_numeric_matrix = FALSE) Retrieve an agent’s simplified data.table version of the theta log. If to_numeric is TRUE, the data.table will be converted to a numeric matrix.
data Active binding, read access to History’s internal data.table.
cumulative Active binding, read access to cumulative data by name through $ accessor.
meta Active binding, read access to meta data by name through $ accessor.

See Also
Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples
```
## Not run:

c <- EpsilonGreedyPolicy$new(epsilon = 0.1)
bandit <- BasicBernoulliBandit$new(weights = c(0.6, 0.1, 0.1))
agent <- Agent$new(policy, bandit, name = "E.G.", sparse = 0.5)
history <- Simulator$new(agents = agent,
                          horizon = 10,
                          simulations = 10)$run()

summary(history)

plot(history)
```
inc<-

```r
dt <- history$get_data_table()
df <- history$get_data_frame()
print(history$cumulative$E.G.$cum_regret_sd)
print(history$cumulative$E.G.$cum_regret)

## End(Not run)
```

---

**inc<-- Increment**

**Description**

inc<-- increments x by value. Equivalent to x <- x + value.

**Usage**

inc(x) <- value

**Arguments**

- x: object to be incremented
- value: value by which x will be modified

**Examples**

```r
x <- 1:5
ing(x) <- 5
x
```

---

**ind**

*On-the-fly indicator function for use in formulae*

**Description**

On-the-fly indicator function for use in formulae

**Usage**

ind(cond)
Arguments

cond 

a logical condition to be evaluated

Value

a binary (0/1) coded variable indicating whether the condition is true

---

inv

Inverse from Choleski (or QR) Decomposition.

Description

Invert a symmetric, positive definite square matrix from its Choleski decomposition.

Usage

inv(M)

Arguments

M 

matrix

Examples

inv(cbind(1, 1:3, c(1,3,7)))

---

invgamma

The Inverse Gamma Distribution

Description

Density, distribution function, quantile function and random generation for the inverse gamma distribution.

Usage

dinvgamma(x, shape, rate = 1, scale = 1/rate, log = FALSE)

pinvgamma(q, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)

qinvgamma(p, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)

rinvgamma(n, shape, rate = 1, scale = 1/rate)
Arguments

- **x**, **q** vector of quantiles.
- **shape** inverse gamma shape parameter
- **rate** inverse gamma rate parameter
- **scale** alternative to rate; scale = 1/rate
- **log, log.p** logical; if TRUE, probabilities p are given as log(p).
- **lower.tail** logical; if TRUE (default), probabilities are P(X <= x) otherwise, P(X > x).
- **p** vector of probabilities.
- **n** number of observations. If length(n) > 1, the length is taken to be the number required.

Details

The inverse gamma distribution with parameters shape and rate has density

\[ f(x) = \frac{\text{rate}^\text{shape}}{\Gamma(\text{shape})} x^{-(1+\text{shape})} e^{-\text{rate}/x} \]

it is the inverse of the standard gamma parameterization in R.

The functions (d/p/q/r)invgamma simply wrap those of the standard (d/p/q/r)gamma R implementation, so look at, say, `dgamma` for details.

Examples

```r
s <- seq(0, 5, .01)
plot(s, dinvgamma(s, 7, 10), type = 'l')

f <- function(x) dinvgamma(x, 7, 10)
q <- 2
integrate(f, 0, q)
(p <- pinvgamma(q, 7, 10))
qinvgamma(p, 7, 10) # = q
mean(rinvgamma(1e5, 7, 10) <= q)
```

---

### invlogit

**Inverse Logit Function**

**Description**

Given a numeric object return the inverse logit of the values.

**Usage**

`invlogit(x)`

**Arguments**

- **x** A numeric object.
### LifPolicy

**Value**

An object of the same type as `x` containing the inverse logits of the input values.

---

### is_rstudio

**Check if in RStudio**

**Description**

Detects whether R is open in RStudio.

**Usage**

```r
is_rstudio()
```

**Value**

A logical value that indicates whether R is open in RStudio.

**Examples**

```r
is_rstudio()
```

---

### LifPolicy

**Policy: Continuum Bandit Policy with Lock-in Feedback**

**Description**

The continuum type Lock-in Feedback (LiF) policy is based on an approach used in physics and engineering, where, if a physical variable `y` depends on the value of a well controllable physical variable `x`, the search for `argmax x f(x)` can be solved via what is nowadays considered as standard electronics. This approach relies on the possibility of making the variable `x` oscillate at a fixed frequency and to look at the response of the dependent variable `y` at the very same frequency by means of a lock-in amplifier. The method is particularly suitable when `y` is immersed in a high noise level, where other more direct methods would fail. Furthermore, should the entire curve shift (or, in other words, if `argmax x f(x)` changes in time, also known as concept drift), the circuit will automatically adjust to the new situation and quickly reveal the new maximum position. This approach is widely used in a very large number of applications, both in industry and research, and is the basis for the Lock-in Feedback (LiF) method.
LinUCBDisjointOptimizedPolicy

Details

In this, Lock in feedback goes through the following steps, again and again:

- Oscillate a controllable independent variable X around a set value at a fixed pace.
- Apply the Lock-in amplifier algorithm to obtain values of the amplitude if the outcome variable Y at the pace you set at step 1.
- Is the amplitude of this variable zero? Congratulations, you have reached lock-in! That is, you have found the optimal value of Y at the current value of X. Still, this optimal value might shift over time, so move to step 1 and repeat the process to make sure we maintain lock-in.
- Is the amplitude less than, or greater than zero? Then move the set value around which we are oscillating our independent variable X up or down on the basis of the outcome.

Now move to step 1 and repeat.

Usage

b <- LifPolicy$new(intime, amplitude, learrate, omega, x0_start)

References


See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

LinUCBDisjointOptimizedPolicy

Policy: LinUCB with unique linear models

Description

Each time step, LinUCBDisjointPolicy runs a linear regression per arm that produces coefficients for each context feature d. Next, LinUCBDisjointPolicy observes the new context, and generates a predicted payoff or reward together with a confidence interval for each available arm. It then proceeds to choose the arm with the highest upper confidence bound.

Usage

policy <- LinUCBDisjointOptimizedPolicy(alpha = 1.0)
**LinUCBDisjointOptimizedPolicy**

**Arguments**

\[
\text{alpha} \quad \text{double, a positive real value R+; Hyper-parameter adjusting the balance between exploration and exploitation.}
\]

\[
\text{name} \quad \text{character string specifying this policy. name is, among others, saved to the History log and displayed in summaries and plots.}
\]

**Parameters**

\[
A \quad d*d \text{ identity matrix}
\]

\[
b \quad \text{a zero vector of length d}
\]

**Methods**

\[
\text{new(alpha = 1)} \quad \text{Generates a new LinUCBDisjointOptimizedPolicy object. Arguments are defined in the Argument section above.}
\]

\[
\text{set_parameters()} \quad \text{each policy needs to assign the parameters it wants to keep track of to list self$\theta_{to\_arms}$ that has to be defined in set_parameters()'s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index\_of\_arm]]$parameter\_name}
\]

\[
\text{get\_action(context)} \quad \text{here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.}
\]

\[
\text{set\_reward(reward, context)} \quad \text{in set\_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.}
\]

**References**


**See Also**

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy
Description

Each time step, LinUCBDisjointPolicy runs a linear regression per arm that produces coefficients for each context feature. Next, LinUCBDisjointPolicy observes the new context, and generates a predicted payoff or reward together with a confidence interval for each available arm. It then proceeds to choose the arm with the highest upper confidence bound.

Usage

```r
policy <- LinUCBDisjointPolicy(alpha = 1.0)
```

Arguments

- `alpha` double, a positive real value \( \mathbb{R}^+ \); Hyper-parameter adjusting the balance between exploration and exploitation.
- `name` character string specifying this policy. `name` is, among others, saved to the History log and displayed in summaries and plots.

Parameters

- `A` \( d \times d \) identity matrix
- `b` a zero vector of length \( d \)

Methods

- `new(alpha = 1)` Generates a new LinUCBDisjointPolicy object. Arguments are defined in the Argument section above.
- `set_parameters()` each policy needs to assign the parameters it wants to keep track of to list self\$theta_to_arms that has to be defined in set_parameters()\'s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]\$parameter_name
- `get_action(context)` here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.
- `set_reward(reward, context)` in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.

References

LinUCBGeneralPolicy

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

LinUCBGeneralPolicy  Policy: LinUCB with unique linear models

Description

Algorithm 1 LinUCB with unique linear models A Contextual-Bandit Approach to Personalized News Article Recommendation

Details

Lihong Li et al

Each time step t, LinUCBGeneralPolicy runs a linear regression per arm that produces coefficients for each context feature d. It then observes the new context, and generates a predicted payoff or reward together with a confidence interval for each available arm. It then proceeds to choose the arm with the highest upper confidence bound.

Usage

policy <- LinUCBGeneralPolicy(alpha = 1.0)

Arguments

alpha  double, a positive real value R+; Hyper-parameter adjusting the balance between exploration and exploitation.

name  character string specifying this policy. name is, among others, saved to the History log and displayed in summaries and plots.

Parameters

A  d*d identity matrix
b  a zero vector of length d

Methods

new(alpha = 1) Generates a new LinUCBGeneralPolicy object. Arguments are defined in the Argument section above.

set_parameters() each policy needs to assign the parameters it wants to keep track of to list self$theta_to_arms that has to be defined in set_parameters()’s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name
get_action(context) here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

set_reward(reward, context) in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.

References


See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

LinUCBHybridOptimizedPolicy

Policy: LinUCB with hybrid linear models

Description

Each time step t, LinUCBHybridOptimizedPolicy runs a linear regression per arm that produces coefficients for each context feature d. Next, it observes the new context, and generates a predicted payoff or reward together with a confidence interval for each available arm. It then proceeds to choose the arm with the highest upper confidence bound.

Usage

policy <- LinUCBHybridOptimizedPolicy(alpha = 1.0)

Arguments

alpha double, a positive real value R+: Hyper-parameter adjusting the balance between exploration and exploitation.

name character string specifying this policy. name is, among others, saved to the History log and displayed in summaries and plots.

Parameters

A d*d identity matrix
b a zero vector of length d
Methods

new(alpha = 1) Generates a new LinUCBHybridOptimizedPolicy object. Arguments are defined in the Argument section above.

set_parameters() each policy needs to assign the parameters it wants to keep track of to list self$theta_to_arms that has to be defined in set_parameters()’s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name

get_action(context) here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

set_reward(reward, context) in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.

References


See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

LinUCBHybridPolicy Policy: LinUCB with hybrid linear models

Description

Each time step t, LinUCBHybridOptimizedPolicy runs a linear regression per arm that produces coefficients for each context feature d. Next, it observes the new context, and generates a predicted payoff or reward together with a confidence interval for each available arm. It then proceeds to choose the arm with the highest upper confidence bound.

Usage

policy <- LinUCBHybridPolicy(alpha = 1.0)

Arguments

alpha  double, a positive real value R+; Hyper-parameter adjusting the balance between exploration and exploitation.

name  character string specifying this policy. name is, among others, saved to the History log and displayed in summaries and plots.
mvrnorm

Parameters

- A: d*d identity matrix
- b: a zero vector of length d

Methods

- `new(alpha = 1)` Generates a new LinUCBHybridPolicy object. Arguments are defined in the Argument section above.
- `set_parameters()` each policy needs to assign the parameters it wants to keep track of to list `self$theta_to_arms` that has to be defined in `set_parameters()`'s body. The parameters defined here can later be accessed by arm index in the following way: `theta[[index_of_arm]]$parameter_name`
- `get_action(context)` here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.
- `set_reward(reward, context)` in `set_reward(reward, context)`, a policy updates its parameter values based on the reward received, and, potentially, the current context.

References


See Also

- Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
- Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
- Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Description

Simulate from a Multivariate Normal Distribution

Produces one or more samples from the specified multivariate normal distribution.

Usage

`mvrnorm(n, mu, sigma)`

Arguments

- `n`: the number of samples required.
- `mu`: a vector giving the means of the variables.
- `sigma`: a positive-definite symmetric matrix specifying the covariance matrix of the variables.
Value

If \( n = 1 \) a vector of the same length as \( \mu \), otherwise an \( n \) by \( \text{length}(\mu) \) matrix with one sample in each row.

---

**OfflineBootstrappedReplayBandit**

*Bandit: Offline Bootstrapped Replay*

**Description**

Policy for the evaluation of policies with offline data through replay with bootstrapping.

**Details**

The key assumption of the method is that the original logging policy chose i.i.d. arms uniformly at random.

Take care: if the original logging policy does not change over trials, data may be used more efficiently via propensity scoring (Langford et al., 2008; Strehl et al., 2011) and related techniques like doubly robust estimation (Dudik et al., 2011).

**Usage**

```r
bandit <- OfflineBootstrappedReplayBandit(formula, 
  data, k = NULL, d = NULL, 
  unique = NULL, shared = NULL, 
  randomize = TRUE, replacement = TRUE, 
  jitter = TRUE, arm_multiply = TRUE)
```

**Arguments**

- `formula` formula (required). Format: `y.context ~ z.choice | x1.context + x2.xcontext + ...`
  By default, adds an intercept to the context model. Exclude the intercept, by adding "0" or ".-1" to the list of contextual features, as in: `y.context ~ z.choice | x1.context + x2.xcontext -1`
- `data` data.table or data.frame; offline data source (required)
- `k` integer; number of arms (optional). Optionally used to reformat the formula defined `x.context` vector as a \( k \times d \) matrix. When making use of such matrix formatted contexts, you need to define custom intercept(s) when and where needed in data.table or data.frame.
- `d` integer; number of contextual features (optional) Optionally used to reformat the formula defined `x.context` vector as a \( k \times d \) matrix. When making use of such matrix formatted contexts, you need to define custom intercept(s) when and where needed in data.table or data.frame.
- `randomize` logical; randomize rows of data stream per simulation (optional, default: TRUE)
- `replacement` logical; sample with replacement (optional, default: TRUE)
- `replacement` logical; add jitter to contextual features (optional, default: TRUE)
- `arm_multiply` logical; multiply the horizon by the number of arms (optional, default: TRUE)
- `unique` integer vector; index of disjoint features (optional)
- `shared` integer vector; index of shared features (optional)
## OfflineBootstrappedReplayBandit

### Methods

```r
classOfflineBootstrappedReplayBandit = new(formula = formula, data = data, k = NULL, d = NULL, unique = NULL, shared = NULL, randomize = TRUE, replacement = TRUE, jitter = TRUE, arm_multiply = TRUE, 
   generates and instantializes a new OfflineBootstrappedReplayBandit instance.
```

- `get_context(t)` argument:
  - `t`: integer, time step `t`.
  - Returns a named list containing the current `d x k` dimensional matrix `context$X`, the number of arms `context$k` and the number of features `context$d`.

- `get_reward(t, context, action)` arguments:
  - `t`: integer, time step `t`.
  - `context`: list, containing the current `context$X` (d x k context matrix), `context$k` (number of arms) and `context$d` (number of context features) (as set by bandit).
  - `action`: list, containing `action$choice` (as set by policy).
  - Returns a named list containing `reward$reward` and, where computable, `reward$optimal` (used by "oracle" policies and to calculate regret).

- `post_initialization()` Randomize offline data by shuffling the offline data.table before the start of each individual simulation when self$randomize is TRUE (default)

### References


### See Also

- Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
- Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineBootstrappedReplayBandit
- Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

### Examples

```r
## Not run:

library(contextual)
library(data.table)

# Import personalization data-set
url <- "http://d1ie9w1kzugsxr.cloudfront.net/data_cmab_basic/dataset.txt"
datafile <- fread(url)
simulations <- 1
horizon <- nrow(datafile)
bandit <- OfflineReplayEvaluatorBandit$new(formula = V2 ~ V1 | . ~ V1, data = datafile)

# Define agents.
```
agents <- list(Agent$new(LinUCBDisjointOptimizedPolicy$new(0.01), bandit, "alpha = 0.01"),
    Agent$new(LinUCBDisjointOptimizedPolicy$new(0.05), bandit, "alpha = 0.05"),
    Agent$new(LinUCBDisjointOptimizedPolicy$new(0.1), bandit, "alpha = 0.1"),
    Agent$new(LinUCBDisjointOptimizedPolicy$new(1.0), bandit, "alpha = 1.0"))

# Initialize the simulation.
simulation <- Simulator$new(agents = agents, simulations = simulations, horizon = horizon,
    do_parallel = FALSE, save_context = TRUE)

# Run the simulation.
sim <- simulation$run()

# plot the results
plot(sim, type = "cumulative", regret = FALSE, rate = TRUE,
    legend_position = "bottomright", ylim = c(0,1))

## End(Not run)

---

**OfflineDirectMethodBandit**

*Bandit: Offline Direct Methods*

**Description**

Policy for the evaluation of policies with offline data with modeled rewards per arm.

**Usage**

```r
bandit <- OfflineDirectMethodBandit(formula,
    data, k = NULL, d = NULL,
    unique = NULL, shared = NULL,
    randomize = TRUE)
```

**Arguments**

- `formula` formula (required). Format: `y.context ~ z.choice | x1.context + x2.xontext + ... | r1.reward + r2.reward`
  
  Here, `r1.reward` to `r.k.reward` represent regression based precalculated rewards per arm. Adds an intercept to the context model by default. Exclude the intercept, by adding "0" or "-1" to the list of contextual features, as in: `y.context ~ z.choice | x1.context + x2.xontext -1`

- `data` data.table or data.frame; offline data source (required)

- `k` integer; number of arms (optional). Optionally used to reformat the formula defined `x.context` vector as a `k x d` matrix. When making use of such matrix formatted contexts, you need to define custom intercept(s) when and where needed in data.table or data.frame.

- `d` integer; number of contextual features (optional). Optionally used to reformat the formula defined `x.context` vector as a `k x d` matrix. When making use of such matrix formatted contexts, you need to define custom intercept(s) when and where needed in data.table or data.frame.
**OfflineDirectMethodBandit**

randomize logical; randomize rows of data stream per simulation (optional, default: TRUE)
replacement logical; sample with replacement (optional, default: FALSE)
replacement logical; add jitter to contextual features (optional, default: FALSE)
unique integer vector; index of disjoint features (optional)
shared integer vector; index of shared features (optional)

**Methods**

`new(formula, data, k = NULL, d = NULL, unique = NULL, shared = NULL, randomize = TRUE)`
generates and instantiates a new `OfflineDirectMethodBandit` instance.

`get_context(t)` argument:
- t: integer, time step t.
  returns a named list containing the current d x k dimensional matrix `context$X`, the number of arms `context$k` and the number of features `context$d`.

`get_reward(t, context, action)` arguments:
- t: integer, time step t.
- context: list, containing the current `context$X` (d x k context matrix), `context$k` (number of arms) and `context$d` (number of context features) (as set by bandit).
- action: list, containing `action$choice` (as set by policy).
  returns a named list containing `reward$reward` and, where computable, `reward$optimal` (used by "oracle" policies and to calculate regret).

`post_initialization()` Randomize offline data by shuffling the offline data.table before the start of each individual simulation when self$randomize is TRUE (default)

**References**


**See Also**

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineDirectMethodBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

**Examples**

```r
## Not run:

library(contextual)
library(data.table)

# Import myocardial infection dataset
url <- "http://dl1e9w1kzugsxr.cloudfront.net/data_propensity/myocardial_propensity.csv"
data <- fread(url)
```
54

```
simulations <- 50
horizon <- nrow(data)

# arms always start at 1
data$trt <- data$trt + 1

# turn death into alive, making it a reward
data$alive <- abs(data$death - 1)

# Run regression per arm, predict outcomes, and save results, a column per arm
f <- alive ~ age + male + risk + severity
model_f <- function(arm) glm(f, data=data[trt==arm],
                        family=binomial(link="logit"),
                        y=FALSE, model=FALSE)
arms <- sort(unique(data$trt))
model_arms <- lapply(arms, FUN = model_f)
predict_arm <- function(model) predict(model, data, type = "response")
r_data <- lapply(model_arms, FUN = predict_arm)
r_data <- do.call(cbind, r_data)
colnames(r_data) <- paste0("R", (1:max(arms)))

# Bind data and model predictions
data <- cbind(data, r_data)

# Define Bandit
f <- alive ~ trt | age + male + risk + severity | R1 + R2 # y ~ z | x | r
bandit <- OfflineDirectMethodBandit$new(formula = f, data = data)

# Define agents.
agents <- list(Agent$new(LinUCBDisjointOptimizedPolicy$new(0.2), bandit, "LinUCB"),
               Agent$new(FixedPolicy$new(1), bandit, "Arm1"),
               Agent$new(FixedPolicy$new(2), bandit, "Arm2"))

# Initialize the simulation.
simulation <- Simulator$new(agents = agents, simulations = simulations, horizon = horizon)

# Run the simulation.
sim <- simulation$run()

# plot the results
plot(sim, type = "cumulative", regret = FALSE, rate = TRUE, legend_position = "bottomright")
plot(sim, type = "arms", limit_agents = "LinUCB", legend_position = "topright")

## End(Not run)
```
Description

Bandit for the doubly robust evaluation of policies with offline data.

Usage

```r
bandit <- OfflineDoublyRobustBandit(formula, 
  data, k = NULL, d = NULL, 
  unique = NULL, shared = NULL, 
  inverted = FALSE, randomize = TRUE)
```

Arguments

- `formula` (required). Format: `y.context ~ z.choice | x1.context + x2.context + ... | r1.reward + r2.reward`
  - Here, `r1.reward` to `r_k.reward` represent regression based precalculated rewards per arm. When leaving out `p.propensity`, Doubly Robust Bandit uses marginal prob per arm for propensities:
  - Adds an intercept to the context model by default. Exclude the intercept, by adding "0" or "-1" to the list of contextual features, as in: `y.context ~ z.choice | x1.context + x2.context -1`
- `data` (data.table or data.frame; offline data source (required)
- `k` (integer; number of arms (optional). Optionally used to reformat the formula defined `x.context` vector as a `k x d` matrix. When making use of such matrix formatted contexts, you need to define custom intercept(s) when and where needed in data.table or data.frame.
- `d` (integer; number of contextual features (optional) Optionally used to reformat the formula defined `x.context` vector as a `k x d` matrix. When making use of such matrix formatted contexts, you need to define custom intercept(s) when and where needed in data.table or data.frame.
- `randomize` (logical; randomize rows of data stream per simulation (optional, default: TRUE)
- `replacement` (logical; sample with replacement (optional, default: FALSE)
- `jitter` (logical; add jitter to contextual features (optional, default: FALSE)
- `unique` (integer vector; index of disjoint features (optional)
- `shared` (integer vector; index of shared features (optional)
- `inverted` (logical; have the propensities been inverted (1/p) or not (p)?
- `threshold` (float (0,1); Lower threshold or Tau on propensity score values. Smaller Tau makes for less biased estimates with more variance, and vice versa. For more information, see paper by Strehl at all (2010). Values between 0.01 and 0.05 are known to work well.

Methods

- `new(formula, data, k = NULL, d = NULL, unique = NULL, shared = NULL, randomize = TRUE)` generates and instantializes a new `OfflineDoublyRobustBandit` instance.
- `get_context(t)` argument:
OfflineDoublyRobustBandit

- t: integer, time step t.
  returns a named list containing the current d x k dimensional matrix context$X, the number of arms context$k and the number of features context$d.

get_reward(t, context, action) arguments:
- t: integer, time step t.
- context: list, containing the current context$X (d x k context matrix), context$k (number of arms) and context$d (number of context features) (as set by bandit).
- action: list, containing action$choice (as set by policy).

returns a named list containing reward$reward and, where computable, reward$optimal (used by "oracle" policies and to calculate regret).

post_initialization() Randomize offline data by shuffling the offline data.table before the start of each individual simulation when self$randomize is TRUE (default)

References


See Also
Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineDoublyRobustBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples
## Not run:

library(contextual)
library(data.table)

# Import myocardial infection dataset

url <- "http://dlie9w1kzugsxr.cloudfront.net/data_propensity/myocardial_propensity.csv"
data <- fread(url)

simulations <- 300
horizon <- nrow(data)

# arms always start at 1
data$trt <- data$trt + 1

# turn death into alive, making it a reward
data$alive <- abs(data$death - 1)

# Run regression per arm, predict outcomes, and save results, a column per arm
f <- alive ~ age + risk + severity
model_f <- function(arm) glm(f, data=data[trt==arm],
    family=binomial(link="logit"),
    y=FALSE, model=FALSE)
arms <- sort(unique(data$trt))
model_arms <- lapply(arms, FUN = model_f)
predict_arm <- function(model) predict(model, data, type = "response")
r_data <- lapply(model_arms, FUN = predict_arm)
r_data <- do.call(cbind, r_data)
colnames(r_data) <- paste0("r", 1:max(arms))

# Bind data and model predictions
data <- cbind(data, r_data)

m <- glm(I(trt-1) ~ age + risk + severity, data=data, family=binomial(link="logit"))
data$p <- predict(m, type = "response")

f <- alive ~ trt | age + risk + severity | r1 + r2 | p
bandit <- OfflineDoublyRobustBandit$new(formula = f, data = data)

# Define agents.
agents <- list(Agent$new(LinUCBDisjointOptimizedPolicy$new(0.2), bandit, "LinUCB"),
    Agent$new(FixedPolicy$new(1), bandit, "Arm1"),
    Agent$new(FixedPolicy$new(2), bandit, "Arm2"))

# Initialize the simulation.
simulation <- Simulator$new(agents = agents, simulations = simulations, horizon = horizon)

# Run the simulation.
sim <- simulation$run()

# plot the results
plot(sim, type = "cumulative", regret = FALSE, rate = TRUE, legend_position = "bottomright")
plot(sim, type = "arms", limit_agents = "LinUCB")

## End(Not run)
Description

Alternative interface for replay style bandit.

Details

TODO: Needs to be documented more fully.

Usage

```r
bandit <- OfflineLookupReplayEvaluatorBandit(offline_data, k, shared_lookup = NULL, unique_lookup = NULL, unique_col = NULL, unique = NULL, shared = NULL, randomize = TRUE)
```

Arguments

- `offline_data` data.table; offline data source (required)
- `k` integer; number of arms (required)
- `d` integer; number of contextual features (required)
- `randomize` logical; randomize rows of data stream per simulation (optional, default: TRUE)
- `unique` integer vector; index of disjoint features (optional)
- `shared` integer vector; index of shared features (optional)

Methods

- `new(offline_data, k, shared_lookup = NULL, unique_lookup = NULL, unique_col = NULL, unique = NULL, shared = NULL)` generates and instantializes a new `OfflineLookupReplayEvaluatorBandit` instance.

get_context(t) argument:

- `t`: integer, time step t.

  returns a named list containing the current $d \times k$ dimensional matrix `context$X`, the number of arms `context$k` and the number of features `context$d`.

get_reward(t, context, action) arguments:

- `t`: integer, time step t.
- `context`: list, containing the current `context$X` ($d \times k$ context matrix), `context$k` (number of arms) and `context$d` (number of context features) (as set by `bandit`).
- `action`: list, containing `action$choice` (as set by policy).

  returns a named list containing `reward$reward` and, where computable, `reward$optimal` (used by "oracle" policies and to calculate regret).

post_initialization() Randomize offline data by shuffling the offline data.table before the start of each individual simulation when `self$randomize` is TRUE (default)

References

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineLookupReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
## Not run:
library(contextual)
library(data.table)
library(splitstackshape)
library(RCurl)

# Import MovieLens ml-10M

# Info: https://d1ie9wlzkugsxr.cloudfront.net/data_movielens/ml-10M/README.html
movies_dat <- "http://d1ie9wlzkugsxr.cloudfront.net/data_movielens/ml-10M/movies.dat"
ratings_dat <- "http://d1ie9wlzkugsxr.cloudfront.net/data_movielens/ml-10M/ratings.dat"

movies_dat <- readLines(movies_dat)
movies_dat <- gsub("::","-",movies_dat)
movies_dat <- paste0(movies_dat, collapse = "\n")
movies_dat <- fread(movies_dat, sep = "-", quote="")
setnames(movies_dat, c("V1", "V2", "V3"), c("MovieID", "Name", "Type"))
movies_dat <- splitstackshape::cSplit_e(movies_dat, "Type", sep = "|", type = "character",
            fill = 0, drop = TRUE)
movies_dat[[3]] <- NULL

ratings_dat <- RCurl::getURL(ratings_dat)
ratings_dat <- readLines(textConnection(ratings_dat))
ratings_dat <- gsub("::","-",ratings_dat)
ratings_dat <- paste0(ratings_dat, collapse = "\n")
ratings_dat <- fread(ratings_dat, sep = "-", quote="")
setnames(ratings_dat, c("V1", "V2", "V3", "V4"), c("UserID", "MovieID", "Rating", "Timestamp"))

all_movies <- ratings_dat[movies_dat, on=c("MovieID" = "MovieID")]

all_movies <- na.omit(all_movies,cols=c("MovieID", "UserID"))

rm(movies_dat,ratings_dat)

all_movies[, UserID := as.numeric(as.factor(UserID))]

count_movies <- all_movies[,.(MovieCount = .N), by = MovieID]
top_50 <- as.vector(count_movies[order(-MovieCount)][1:50]$MovieID)
not_50 <- as.vector(count_movies[order(-MovieCount)][51:nrow(count_movies)]$MovieID)
top_50_movies <- all_movies[MovieID %in% top_50]
```
# Create feature lookup tables - to speed up, MovieID and UserID are
# ordered and lined up with the (dt/matrix) default index.

# Arm features

# MovieID of top 50 ordered from 1 to N:
top_50_movies[, MovieID := as.numeric(as.factor(MovieID))]
arm_features <- top_50_movies[head(.SD, 1), by = MovieID][, c(1, 6:24)]
setorder(arm_features, MovieID)

# User features

# Count of categories for non-top-50 movies normalized per user
user_features <- all_movies[MovieID %in% not_50]
user_features[, c("MovieID", "Rating", "Timestamp", "Name") := NULL]
user_features <- user_features[, lapply(.SD, sum, na.rm = TRUE), by = UserID]
user_features[, total := rowSums(.SD, na.rm = TRUE), .SDcols = 2:20]
user_features[, 2:20 := lapply(.SD, function(x) x/total), .SDcols = 2:20]
user_features$total <- NULL

# Add users that were not in the set of non-top-50 movies (4 in 10m dataset)
all_users <- as.data.frame(unique(all_movies$UserID))
user_features <- user_features[all_users, on = c(UserID = "V1")]
user_features[is.na(user_features)] <- 0
setorder(user_features, UserID)

rm(all_movies, not_50, top_50, count_movies)

# Contextual format

top_50_movies[, t := .1]
top_50_movies[, sim := .1]
top_50_movies[, agent := "Offline"]
top_50_movies[, choice := MovieID]
top_50_movies[, reward := ifelse(Rating <= 4, 0, 1)]
setorder(top_50_movies, Timestamp, Name)

# Run simulation

simulations <- 1
horizon <- nrow(top_50_movies)

bandit <- OfflineLookupReplayEvaluatorBandit$new(top_50_movies, 
  k = 50,
  unique_col = "UserID",
  shared_lookup = arm_features,
  unique_lookup = user_features)
agents <- list(Agent$new(ThompsonSamplingPolicy$new(), bandit, "Thompson"),
  Agent$new(UCB1Policy$new(), bandit, "UCB1"),
OfflinePropensityWeightingBandit

Bandit: Offline Propensity Weighted Replay

Description
Policy for the evaluation of policies with offline data through replay with propensity weighting.

Usage
```r
bandit <- OfflinePropensityWeightingBandit(formula,
                                         data, k = NULL, d = NULL,
                                         unique = NULL, shared = NULL,
                                         randomize = TRUE, replacement = TRUE,
                                         jitter = TRUE, arm_multiply = TRUE,
                                         inverted = FALSE)
```

Arguments
- `formula` formula (required). Format: `y.context ~ z.choice | x1.context + x2.xontext + ... | p.propensity` When leaving out `p.propensity`, Doubly Robust Bandit uses marginal prob per arm for propensities: By default, adds an intercept to the context model. Exclude the intercept, by adding "0" or "-1" to the list of contextual features, as in: `y.context ~ z.choice | x1.context + x2.xontext -1 | p.propensity`
- `data` data.table or data.frame; offline data source (required)
- `k` integer; number of arms (optional). Optionally used to reformat the formula defined x.context vector as a k x d matrix. When making use of such matrix formatted contexts, you need to define custom intercept(s) when and where needed in data.table or data.frame.
__OfflinePropensityWeightingBandit__

*d* integer; number of contextual features (optional) Optionally used to reformat the formula defined

x.context vector as a k x d matrix. When making use of such matrix formatted contexts, you
need to define custom intercept(s) when and where needed in data.table or data.frame.

**randomize** logical; randomize rows of data stream per simulation (optional, default: TRUE)

**replacement** logical; sample with replacement (optional, default: TRUE)

**jitter** logical; add jitter to contextual features (optional, default: TRUE)

**arm_multiply** logical; multiply the horizon by the number of arms (optional, default: TRUE)

**inverted** logical; have the propensity scores been weighted (optional, default: FALSE)

**threshold** float (0,1); Lower threshold or Tau on propensity score values. Smaller Tau makes for
less biased estimates with more variance, and vice versa. For more information, see paper by
Strehl at all (2010). Values between 0.01 and 0.05 are known to work well.

**unique** integer vector; index of disjoint features (optional)

**shared** integer vector; index of shared features (optional)

**Methods**

`new(formula, data, k = NULL, d = NULL, unique = NULL, shared = NULL, randomize = TRUE, replacement = TRUE)`

generates and instantializes a new `OfflinePropensityWeightingBandit` instance.

`get_context(t)` argument:

* `t`: integer, time step t.

returns a named list containing the current d x k dimensional matrix context$X, the number
of arms context$k and the number of features context$d.

`get_reward(t, context, action)` arguments:

* `t`: integer, time step t.

* `context`: list, containing the current context$X (d x k context matrix), context$k (num-
ber of arms) and context$d (number of context features) (as set by bandit).

* `action`: list, containing action$choice (as set by policy).

returns a named list containing reward$reward and, where computable, reward$optimal
(used by "oracle" policies and to calculate regret).

`post_initialization()` Randomize offline data by shuffling the offline data.table before the start
of each individual simulation when self$randomize is TRUE (default)

**References**


Strehl, Alex, et al. "Learning from logged implicit exploration data." Advances in Neural Informa-

**See Also**

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflinePropensityWeightingBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy
OfflineReplayEvaluatorBandit

Examples

```r
## Not run:

library(contextual)
library(data.table)

# Import myocardial infection dataset

url <- "http://dlie9wilzugsxr.cloudfront.net/data_propensity/myocardial_propensity.csv"
data <- fread(url)
simulations <- 3000
horizon <- nrow(data)

# arms always start at 1
data$trt <- data$trt + 1

# turn death into alive, making it a reward
data$alive <- abs(data$death - 1)

# calculate propensity weights

m <- glm(I(trt-1) ~ age + risk + severity, data=data, family=binomial(link="logit"))
data$p <- predict(m, type = "response")

# run bandit - if you leave out p, Propensity Bandit uses marginal prob per arm for propensities:
# table(private$z)/length(private$z)

f <- alive ~ trt | age + risk + severity | p

bandit <- OfflinePropensityWeightingBandit$new(formula = f, data = data)

# Define agents.
agents <- list(Agent$new(LinUCBDisjointOptimizedPolicy$new(0.2), bandit, "LinUCB"))

# Initialize the simulation.
simulation <- Simulator$new(agents = agents, simulations = simulations, horizon = horizon)

# Run the simulation.
sim <- simulation$run()

# plot the results
plot(sim, type = "cumulative", regret = FALSE, rate = TRUE, legend_position = "bottomright")

## End(Not run)
```

OfflineReplayEvaluatorBandit

Bandit: Offline Replay
OfflineReplayEvaluatorBandit

Description

Policy for the evaluation of policies with offline data through replay.

Details

The key assumption of the method is that the original logging policy chose i.i.d. arms uniformly at random.

Take care: if the original logging policy does not change over trials, data may be used more efficiently via propensity scoring (Langford et al., 2008; Strehl et al., 2011) and related techniques like doubly robust estimation (Dudik et al., 2011).

Usage

```r
bandit <- OfflineReplayEvaluatorBandit(formula, data, k = NULL, d = NULL, unique = NULL, shared = NULL, randomize = TRUE, replacement = FALSE, jitter = FALSE)
```

Arguments

- `formula` formula (required). Format: `y.context ~ z.choice | x1.context + x2.context + ...` By default, adds an intercept to the context model. Exclude the intercept, by adding "0" or "-1" to the list of contextual features, as in: `y.context ~ z.choice | x1.context + x2.context -1`
- `data` data.frame or data.table; offline data source (required)
- `k` integer; number of arms (optional). Optionally used to reformat the formula defined `x.context` vector as a `k x d` matrix. When making use of such matrix formatted contexts, you need to define custom intercept(s) when and where needed in data.table or data.frame.
- `d` integer; number of contextual features (optional) Optionally used to reformat the formula defined `x.context` vector as a `k x d` matrix. When making use of such matrix formatted contexts, you need to define custom intercept(s) when and where needed in data.table or data.frame.
- `randomize` logical; randomize rows of data stream per simulation (optional, default: TRUE)
- `replacement` logical; sample with replacement (optional, default: FALSE)
- `jitter` logical; add jitter to contextual features (optional, default: FALSE)
- `unique` integer vector; index of disjoint features (optional)
- `shared` integer vector; index of shared features (optional)

Methods

- `new(formula, data, k = NULL, d = NULL, unique = NULL, shared = NULL, randomize = TRUE, replacement = FALSE)` generates and instantializes a new OfflineReplayEvaluatorBandit instance.
- `get_context(t)` argument:
  - `t`: integer, time step \( t \).
  - returns a named list containing the current \( d \times k \) dimensional matrix `context$X`, the number of arms `context$k` and the number of features `context$d`.
get_reward(t, context, action) arguments:
  • t: integer, time step t.
  • context: list, containing the current context $X(d \times k$ context matrix), context $k$ (number of arms) and context $d$ (number of context features) (as set by bandit).
  • action: list, containing action $\text{choice}$ (as set by policy).

returns a named list containing reward $\text{reward}$ and, where computable, reward $\text{optimal}$ (used by "oracle" policies and to calculate regret).

post_initialization() Randomize offline data by shuffling the offline data.table before the start of each individual simulation when self$randomize is TRUE (default)

References

See Also
Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLInTS Policy

Examples
## Not run:
url <- "http://dlie9wlkzugsxr.cloudfront.net/data_irecsys_CARSKit/Movie_DePaulMovie/ratings.csv"
data <- fread(url, stringsAsFactors=TRUE)

# Convert data
data <- contextual::one_hot(data, cols = c("Time", "Location", "Companion"),
  sparseNAs = TRUE)
data[, itemid := as.numeric(itemid)]
data[, rating := ifelse(rating <= 3, 0, 1)]

# Set simulation parameters.
simulations <- 10 # here, "simulations" represents the number of bootstrap samples
horizon <- nrow(data)

# Initiate Replay bandit with 10 arms and 100 context dimensions
log_S <- data
formula <- formula("rating ~ itemid | Time_Weekday + Time_Weekend + Location_Cinema +
  Location_Home + Companion_A lone + Companion_Family + Companion_Partner")
bandit <- OfflineReplayEvaluatorBandit$new(formula = formula, data = data)

# Define agents.
agents <-
ones_in_zeroes

A vector of zeroes and ones

Description

A vector of zeroes and ones

Usage

ones_in_zeroes(vector_length, index_of_one)

Arguments

vector_length   How long will the vector be?
index_of_one    Where to insert the one?

Value

Vector of zeroes with one(s) at given index position(s)
**one_hot**  

*One Hot Encoding of data.table columns*

**Description**

One-Hot-Encode unordered factor columns of a data.table mltools. From ben519’s “mltools” package.

**Usage**

```r
one_hot(dt, cols = "auto", sparsifyNAs = FALSE, naCols = FALSE, 
        dropCols = TRUE, dropUnusedLevels = FALSE)
```

**Arguments**

- `dt` A data.table
- `cols` Which column(s) should be one-hot-encoded? DEFAULT = "auto" encodes all unordered factor columns.
- `sparsifyNAs` Should NAs be converted to 0s?
- `naCols` Should columns be generated to indicate the present of NAs? Will only apply to factor columns with at least one NA
- `dropCols` Should the resulting data.table exclude the original columns which are one-hot-encoded?
- `dropUnusedLevels` Should columns of all 0s be generated for unused factor levels?

**Details**

One-hot-encoding converts an unordered categorical vector (i.e. a factor) to multiple binarized vectors where each binary vector of 1s and 0s indicates the presence of a class (i.e. level) of the original vector.

**Examples**

```r
library(data.table)

dt <- data.table(
  ID = 1:4,
  color = factor(c("red", NA, "blue", "blue"), levels=c("blue", "green", "red"))
)

one_hot(dt)
one_hot(dt, sparsifyNAs=TRUE)
one_hot(dt, naCols=TRUE)
one_hot(dt, dropCols=FALSE)
one_hot(dt, dropUnusedLevels=TRUE)
```
Policy: Oracle

Description

OraclePolicy is also known as a "cheating" or "godlike" policy, as it knows the reward probabilities at all times, and will always play the optimal arm. It is often used as a baseline to compare other policies to.

Usage

policy <- OraclePolicy()

Arguments

name character string specifying this policy. name is, among others, saved to the History log and displayed in summaries and plots.

Methods

new() Generates a new OraclePolicy object. Arguments are defined in the Argument section above.

set_parameters() each policy needs to assign the parameters it wants to keep track of to list self$theta_to_arms that has to be defined in set_parameters()'s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name

get_action(context) here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

set_reward(reward, context) in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.

References


See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy
Description

Generates plots from History data.

Details

Usually not instantiated directly but invoked by calling the generic plot(h), where h is an History class instance.

Usage

Plot <- Plot$new()

Methods

cumulative(history,...) Plots cumulative regret or reward (depending on parameter regret=TRUE/FALSE) over time.
average(history,...) Plots average regret or reward (depending on parameter regret=TRUE/FALSE) over time.
arms(history,...) Plot the percentage of simulations per time step each arm was chosen over time. If multiple agents have been run, plots only the first agent.

Plot method arguments

type (character, "cumulative") Can be either "cumulative" (default), "average", or "arms". Sets the plot method when Plot() is called through R’s generic plot() function. Methods are described in the Methods section above.
regret (logical, TRUE) Plot policy regret (default, TRUE) or reward (FALSE)?
rate (logical, TRUE) If rate is TRUE, the rate of regret or reward is plotted.
limit_agents (list, NULL) Limit plotted agents to the agents in the list.
limit_context (character vector, NULL) Only plots data where context feature name(s) in vector equal to one.
no_par (logical, FALSE) If no_par is TRUE, Plot() does not set or adjust plotting parameters itself. This makes it possible to set custom plotting parameters through R’s par() function.
legend (logical, TRUE) Shows the legend when TRUE (default).
legend_title (character, NULL) Sets a custom legend title.
legend_labels (character list, NULL) Sets legend labels to custom values as specified in list.
legend_border (logical, NULL) When TRUE, the legend is borderless.
legend_position (character, "topleft") a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center". This places the legend on the inside of the plot frame at the given location.

xlim (c(integer, integer), NULL) Sets x-axis limits.

ylim (c(integer, integer), NULL) Sets y-axis limits.

log (character, "") A character string which contains "x" if the x axis is to be logarithmic, "y" if the y axis is to be logarithmic and "xy" or "yx" if both axes are to be logarithmic.

use_colors (logical, TRUE) If use_colors is FALSE, plots will be in grayscale. Otherwise, plots will make use of a color palette (default).

disp (character, NULL) When disp (for "dispersion measure") is set to either 'var', 'sd' or 'ci', the variance, standard deviation, or 95

plot_only_disp (logical, FALSE) When TRUE and disp is either 'var', 'sd' or 'ci', plot only dispersion measure.

traces (logical, FALSE) Plot traces of independent simulations (default is FALSE).

traces_max (integer, 100) The number of trace lines.

traces_alpha (numeric, 0.3) Opacity of the trace lines. Default is 0.3 - that is, an opacity of 30

smooth (logical, FALSE) Smooth the plot (default is FALSE)

interval (integer, NULL) Plot only every t

cum_average (logical, FALSE) Calculates moving average from cum_reward or cum_regret with step size interval.

color_step (integer, 1) When > 1, the plot cycles through nr_agents/color_step colors.

lty_step (integer, 1) When > 1, the plot cycles through nr_agents/lty_step line types.

lwd (integer, 1) Line width.

xlab (character, NULL) a title for the x axis

ylab (character, NULL) a title for the y axis

trunc_over_agents (logical, TRUE) Truncate the chart to the agent with the fewest time steps t.

trunc_per_agent (logical, TRUE) Truncate every agent’s plot to the number of time steps that have been fully simulated. That is, time steps for which the number of simulations equals the number defined in Simulator’s simulations parameter.

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy
plot.history

Examples

## Not run:

```r
bandit <- ContextualPrecachingBandit$new(weights = c(0.9, 0.1, 0.1))

agents <- list(
  Agent$new(RandomPolicy$new(), bandit),
  Agent$new(OraclePolicy$new(), bandit),
  Agent$new(ThompsonSamplingPolicy$new(1.0, 1.0), bandit),
  Agent$new(Exp3Policy$new(0.1), bandit),
  Agent$new(GittinsBrezziLaiPolicy$new(), bandit),
  Agent$new(UCB1Policy$new(), bandit))

history <- Simulator$new(agents, horizon = 100, simulations = 1000)$run()

par(mfrow = c(3, 2), mar = c(1, 4, 2, 1), cex=1.3)

plot(history, type = "cumulative", use_colors = FALSE, no_par = TRUE, legend_border = FALSE,
     limit_agents = c("GittinsBrezziLai", "UCB1", "ThompsonSampling"))

plot(history, type = "cumulative", regret = FALSE, legend = FALSE,
     limit_agents = c("UCB1"), traces = TRUE, no_par = TRUE)

plot(history, type = "cumulative", regret = FALSE, rate = TRUE, disp = "sd",
     limit_agents = c("Exp3", "ThompsonSampling"),
     legend_position = "bottomright", no_par = TRUE)

plot(history, type = "cumulative", rate = TRUE, plot_only_disp = TRUE,
     disp = "var", smooth = TRUE, limit_agents = c("UCB1", "GittinsBrezziLai"),
     legend_position = "bottomleft", no_par = TRUE)

plot(history, type = "average", disp = "ci", regret = FALSE, interval = 10,
     smooth = TRUE, legend_position = "bottomright", no_par = TRUE, legend = FALSE)

plot(history, limit_agents = c("ThompsonSampling"), type = "arms",
     interval = 20, no_par = TRUE)
```

## End(Not run)

---

**plot.history**  
*Plot Method for Contextual History*

**Description**

plot.history, a method for the plot generic. It is designed for a quick look at History data.
Usage

```r
## S3 method for class 'History'
plot(x, ...)
```

Arguments

- `x`: A `History` object.
- `...`: Further plotting parameters.

See Also

Core contextual classes: `Simulator`, `Agent`, `History`, `Plot`
Bandit classes: `Bandit`, `BasicBernoulliBandit`, `OfflineReplayEvaluatorBandit`, `ContextualLogitBandit`

---

Policy: Superclass

Description

Parent or superclass of all `{contextual}` Policy subclasses.

Details

On every \( t = 1, \ldots, T \), a policy receives a \( d \)-dimensional feature vector or \( d \times k \)-dimensional matrix \( \text{context}^X \), the current number of Bandit arms in \( \text{context}^k \), and the current number of contextual features in \( \text{context}^d \).

To make sure a policy supports both contextual feature vectors and matrices in \( \text{context}^X \), it is suggested any contextual policy makes use of `contextual`'s `get_arm_context(context, arm)` utility function to obtain the current context for a particular arm, and `get_full_context(context)` where a policy makes direct use of a \( d \times k \)-dimensional context matrix.

It has to compute which of the \( k \) Bandit arms to pull by taking into account this contextual information plus the policy’s current parameter values stored in the named list \( \text{theta} \). On selecting an arm, the policy then returns its index as `action$choice`.

[Diagram showing the interaction between Simulator, Agent, Policy, and Bandit]

On pulling a Bandit arm the policy receives a Bandit reward through `reward$reward`. In combination with the current `context$X` and `action$choice`, this reward can then be used to update to the policy’s parameters as stored in list `theta`. 
* Note: in context-free scenario’s, context\$X can be omitted.

**Usage**

```r
policy <- Policy$new()
```

**Methods**

- `new()` Generates and initializes a new Policy object.
- `get_action(t, context)`
  - Arguments:
    - `t`: integer, time step t.
    - `context`: list, containing the current context\$X (d x k context matrix), context\$k (number of arms) and context\$d (number of context features)
  - Computes which arm to play based on the current values in named list theta and the current context. Returns a named list containing action\$choice, which holds the index of the arm to play.
- `set_reward(t, context, action, reward)`
  - Arguments:
    - `t`: integer, time step t.
    - `context`: list, containing the current context\$X (d x k context matrix), context\$k (number of arms) and context\$d (number of context features) (as set by bandit).
    - `action`: list, containing action\$choice (as set by policy).
    - `reward`: list, containing reward\$reward and, if available, reward\$optimal (as set by bandit).
  - Utilizes the above arguments to update and return the set of parameters in list theta.
- `set_parameters()` Helper function, called during a Policy’s initialisation, assigns the values it finds in list self\$theta_to_arms to each of the Policy’s k arms. The parameters defined here can then be accessed by arm index in the following way: theta[[index_of_arm]]\$parameter_name.

**See Also**

- Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
- Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
- Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy
print.history

Print Method for Contextual History

Description

print.history, a method for the print generic. It is designed for a quick look at History data.

Usage

```r
## S3 method for class 'History'
print(x, ...)
```

Arguments

- `x`: A History object.
- `...`: Further plotting parameters.

See Also

Core contextual classes: Simulator, Agent, History, Plot
Bandit classes: Bandit, BasicBernoulliBandit, OfflineReplayEvaluatorBandit, ContextualLogitBandit

prob_winner

Binomial Win Probability

Description

Function to compute probability that each arm is the winner, given simulated posterior results.

Usage

```r
prob_winner(post)
```

Arguments

- `post`: Simulated results from the posterior, as provided by sim_post()

Value

Probabilities each arm is the winner.

Author(s)

Thomas Lotze and Markus Loecher
**RandomPolicy**

**Examples**

```r
dx <- c(10, 20, 30, 50)
n <- c(100, 102, 120, 130)
betapost <- sim_post(x, n)
pw <- prob_winner(betapost)
```

**Description**

RandomPolicy always explores, choosing arms uniformly at random. In that respect, RandomPolicy is the mirror image of a pure greedy policy, which would always seek to exploit.

**Usage**

```r
policy <- RandomPolicy(name = "RandomPolicy")
```

**Arguments**

- `name`: character string specifying this policy. `name` is, among others, saved to the History log and displayed in summaries and plots.

**Methods**

- `new()`: Generates a new RandomPolicy object. Arguments are defined in the Argument section above.

- `set_parameters()`: Each policy needs to assign the parameters it wants to keep track of to list `self$theta_to_arms` that has to be defined in `set_parameters()`'s body. The parameters defined here can later be accessed by arm index in the following way: `theta[[index_of_arm]]$parameter_name`

- `get_action(context)`: Here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

- `set_reward(reward, context)`: In `set_reward(reward, context)`, a policy updates its parameter values based on the reward received, and, potentially, the current context.

**References**

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
horizon <- 100L
simulations <- 100L
weights <- c(0.9, 0.1, 0.1)

policy <- RandomPolicy$new()
bandit <- BasicBernoulliBandit$new(weights = weights)
age nt <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, simulations, do_parallel = FALSE)$run()

plot(history, type = "arms")
```

---

**sample_one_of**

*Sample one element from vector or list*

**Description**

Takes one sample from a vector or list. Does not throw an error for zero length lists.

**Usage**

`sample_one_of(x)`

**Arguments**

- `x`  
  A vector of one or more elements from which to choose

**Value**

One value, drawn from `x`. 
Description

Checks to see if the user is in RStudio. If so, then it changes the device to a popup window.

Usage

```
set_external(ext = TRUE, width = 10, height = 6)
```

Arguments

- `ext`: A logical indicating whether to plot in a popup or within the RStudio UI.
- `width`: Width in pixels of the popup window
- `height`: Height in pixels of the popup window

Details

Depending on the operating system, the default drivers attempted to be used are:

- OS X: quartz()
- Linux: x11()
- Windows: windows()

Note, this setting is not permanent. Thus, the behavioral change will last until the end of the session. Also, the active graphing environment will be killed. As a result, any graphs that are open will be deleted.

Examples

```
## Not run:

# Turn on external graphs
external_graphs()

# Turn off external graphs
external_graphs(F)

## End(Not run)
```
**Sherman-Morrisson inverse**

**Description**

Sherman-Morrisson inverse

**Usage**

```plaintext
sherman_morrisson(inv, x)
```

**Arguments**

- `inv`: to be updated inverse matrix
- `x`: column vector to update inv with

---

**Simulator**

**Description**

The entry point of any contextual simulation.

**Details**

A Simulator takes, at a minimum, one or more Agent instances, a horizon (the length of an individual simulation, \( t = 1, \ldots, T \)) and the number of simulations (How many times to repeat each simulation over \( t = 1, \ldots, T \), with a new seed on each repeat*).

It then runs all simulations (in parallel by default), keeping a log of all Policy and Bandit interactions in a History instance.

* Note: to be able to fairly evaluate and compare each agent’s performance, and to make sure that simulations are replicable, for each separate agent, seeds are set equally and deterministically for each agent over all horizon x simulations time steps.
Usage

```r
simulator <- Simulator$new(agents,
    horizon = 100L,
    simulations = 100L,
    save_interval = 1,
    save_context = FALSE,
    save_theta = FALSE,
    do_parallel = TRUE,
    worker_max = NULL,
    t_over_sims = FALSE,
    set_seed = 0,
    progress_file = FALSE,
    include_packages = NULL)
```

Arguments

- **agents** An Agent instance or a list of Agent instances.
- **horizon** integer. The number of pulls or time steps to run each agent, where \( t = 1, \ldots, T \).
- **simulations** integer. How many times to repeat each agent’s simulation over \( t = 1, \ldots, T \), with a new seed on each repeat (itself deterministically derived from `set_seed`).
- **save_interval** integer. Write data to history only every `save_interval` time steps. Default is 1.
- **save_context** logical. Save the context matrices \( X \) to the History log during a simulation?
- **save_theta** logical. Save the parameter list `theta` to the History log during a simulation?
- **do_parallel** logical. Run `simulator` processes in parallel?
- **worker_max** integer. Specifies how many parallel workers are to be used. If unspecified, the amount of workers defaults to `max(workers_available) - 1`.
- **t_over_sims** logical. Of use to, among others, offline Bandits. If `t_over_sims` is set to TRUE, the current `simulator` iterates over all rows in a data set for each repeated simulation. If FALSE, it splits the data into `simulations` parts, and a different subset of the data for each repeat of an agent’s simulation.
- **set_seed** integer. Sets the seed of R’s random number generator for the current `simulator`.
- **progress_file** logical. If TRUE, `simulator` writes `workers_progress.log`, `agents_progress.log` and `parallel.log` files to the current working directory, allowing you to keep track of respectively workers, agents, and potential errors when running a `simulator` in parallel mode.
- **log_interval** integer. Sets the log write interval. Default every 1000 time steps.
- **include_packages** list. List of packages that (one of) the policies depend on. If a Policy requires an R package to be loaded, this option can be used to load that package on each of the workers. Ignored if `do_parallel` is FALSE.
- **chunk_multiplier** integer. By default, simulations are equally divided over available workers, and every worker saves its simulation results to a local history file which is then aggregated. Depending on workload, network bandwidth, memory size and other variables it can sometimes be useful to break these workloads into smaller chunks. This can be done by setting the chunk_multiplier to some integer value, where the number of chunks will total `chunk_multiplier x number_of_workers`.
Methods

reset() Resets a Simulator instance to its original initialisation values.
run() Runs a Simulator instance.
history Active binding, read access to Simulator’s History instance.

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
## Not run:
policy <- EpsilonGreedyPolicy$new(epsilon = 0.1)
bandit <- BasicBernoulliBandit$new(weights = c(0.6, 0.1, 0.1))
agent <- Agent$new(policy, bandit, name = "E.G.", sparse = 0.5)

history <- Simulator$new(agents = agent,
                           horizon = 10,
                           simulations = 10)$run()

summary(history)

plot(history)

dt <- history$get_data_table()

df <- history$get_data_frame()

print(history$cumulative$E.G.$cum_regret_sd)

print(history$cumulative$E.G.$cum_regret)

## End(Not run)
```

---

**sim_post**  
*Binomial Posterior Simulator*

Description

Simulates the posterior distribution of the Bayesian probabilities for each arm being the best binomial bandit.
Usage

\[
\text{sim\_post}(x, n, \alpha = 1, \beta = 1, \text{ndraws} = 5000)
\]

Arguments

- \(x\) Vector of the number of successes per arm.
- \(n\) Vector of the number of trials per arm.
- \(\alpha\) Shape parameter \(\alpha\) for the prior beta distribution.
- \(\beta\) Shape parameter \(\beta\) for the prior beta distribution.
- \(\text{ndraws}\) Number of random draws from the posterior.

Value

Matrix of bayesian probabilities for each arm being the best binomial bandit

Author(s)

Thomas Lotze and Markus Loecher

Examples

\[
\begin{align*}
x & \leftarrow c(10, 20, 30, 50) \\
n & \leftarrow c(100, 102, 120, 130) \\
sp & \leftarrow \text{sim\_post}(x, n)
\end{align*}
\]

---

**SoftmaxPolicy**

Policy: Softmax

Description

SoftmaxPolicy is very similar to Exp3Policy, but selects an arm based on the probability from the Boltmann distribution. It makes use of a temperature parameter \(\tau\), which specifies how many arms we can explore. When \(\tau\) is high, all arms are explored equally, when \(\tau\) is low, arms offering higher rewards will be chosen.

Usage

\[
policy \leftarrow \text{SoftmaxPolicy}(\tau = 0.1)
\]

Arguments

- \(\tau = 0.1\) double, temperature parameter \(\tau\) specifies how many arms we can explore. When \(\tau\) is high, all arms are explored equally, when \(\tau\) is low, arms offering higher rewards will be chosen.
SoftmaxPolicy

Methods

new(epsilon = 0.1) Generates a new SoftmaxPolicy object. Arguments are defined in the Argument section above.

set_parameters() each policy needs to assign the parameters it wants to keep track of to list self$theta_to_arms that has to be defined in set_parameters()'s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name

get_action(context) here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

set_reward(reward, context) in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.

References


See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

```r
horizon <- 100L
simulations <- 100L
weights <- c(0.9, 0.1, 0.1)

policy <- SoftmaxPolicy$new(tau = 0.1)
bandit <- BasicBernoulliBandit$new(weights = weights)
agent <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, simulations, do_parallel = FALSE)$run()

plot(history, type = "cumulative")

plot(history, type = "arms")
```
**summary.history**

**Summary Method for Contextual History**

**Description**

summary.history, a method for the summary generic. It is designed for a quick summary of History data.

**Usage**

```r
## S3 method for class 'History'
summary(object, ...)
```

**Arguments**

- `object` A History object.
- `...` Further summary parameters.

**See Also**

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

---

**sum_of**

**Sum of list**

**Description**

Returns the sum of the values of the elements of a list x.

**Usage**

```r
sum_of(x)
```

**Arguments**

- `x` List

**Details**

If there is a tie, and equal_is_random is TRUE, the index of one of the tied maxima is returned at random. Otherwise, the value with the lowest index is returned.
Thompson Sampling Policy

**Examples**

```r
theta = list(par_one = list(1,2,3), par_two = list(2,3,4))
sum_of(theta$par_one)
```

**Description**

Thompson Sampling Policy works by maintaining a prior on the mean rewards of its arms. In this, it follows a beta-binomial model with parameters alpha and beta, sampling values for each arm from its prior and picking the arm with the highest value. When an arm is pulled and a Bernoulli reward is observed, it modifies the prior based on the reward. This procedure is repeated for the next arm pull.

**Usage**

```r
policy <- ThompsonSamplingPolicy(alpha = 1, beta = 1)
```

**Arguments**

- `alpha` integer, a natural number N>0 - first parameter of the Beta distribution
- `beta` integer, a natural number N>0 - second parameter of the Beta distribution

**Methods**

- `new(alpha = 1, beta = 1)` Generates a new ThompsonSamplingPolicy object. Arguments are defined in the Argument section above.
- `set_parameters()` each policy needs to assign the parameters it wants to keep track of to list `self$theta_to_arms` that has to be defined in `set_parameters()`’s body. The parameters defined here can later be accessed by arm index in the following way: `theta[[index_of_arm]]$parameter_name`
- `get_action(context)` here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.
- `set_reward(reward, context)` in `set_reward(reward, context)`, a policy updates its parameter values based on the reward received, and, potentially, the current context.
**UCB1Policy**

**Description**

UCB policy for bounded bandits with a Chernoff-Hoeffding Bound

**Details**

**UCB1Policy** constructs an optimistic estimate in the form of an Upper Confidence Bound to create an estimate of the expected payoff of each action, and picks the action with the highest estimate. If the guess is wrong, the optimistic guess quickly decreases, till another action has the higher estimate.

**Usage**

``` r
policy <- UCB1Policy()
```

---

**References**


**See Also**

Core contextual classes: **Bandit, Policy, Simulator, Agent, History, Plot**

Bandit subclass examples: **BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit**

Policy subclass examples: **EpsilonGreedyPolicy, ContextualLinTSPolicy**

**Examples**

``` r
horizon <- 100L
simulations <- 100L
weights <- c(0.9, 0.1, 0.1)

policy <- ThompsonSamplingPolicy$new(alpha = 1, beta = 1)
bandit <- BasicBernoulliBandit$new(weights = weights)
agent <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, simulations, do_parallel = FALSE)$run()

plot(history, type = "cumulative")
```
Methods

new() Generates a new UCB1Policy object.

set_parameters() each policy needs to assign the parameters it wants to keep track of to list
self$theta_to_arms that has to be defined in set_parameters()'s body. The parameters
declared here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name

get_action(context) here, a policy decides which arm to choose, based on the current values of
its parameters and, potentially, the current context.

set_reward(reward, context) in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.

References

applied mathematics, 6(1), 4-22.

See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot

Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit

Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy

Examples

## Not run:

```r
horizon <- 100L
simulations <- 100L
weights <- c(0.9, 0.1, 0.1)
policy <- UCB1Policy$new()
bandit <- BasicBernoulliBandit$new(weights = weights)
agent <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, simulations, do_parallel = FALSE)$run()

plot(history, type = "cumulative")

plot(history, type = "arms")

## End(Not run)
```
UCB2Policy

Policy: UCB2

Description

UCB policy for bounded bandits with plays divided in epochs.

Details

UCB2Policy constructs an optimistic estimate in the form of an Upper Confidence Bound to create an estimate of the expected payoff of each action, and picks the action with the highest estimate. If the guess is wrong, the optimistic guess quickly decreases, till another action has the higher estimate.

Usage

policy <- UCB2Policy(alpha = 0.1)

Arguments

alpha numeric; Tuning parameter in the interval (0,1)

Methods

new(alpha = 0.1) Generates a new UCB2Policy object.

set_parameters() each policy needs to assign the parameters it wants to keep track of to list self$theta_to_arms that has to be defined in set_parameters()’s body. The parameters defined here can later be accessed by arm index in the following way: theta[[index_of_arm]]$parameter_name

get_action(context) here, a policy decides which arm to choose, based on the current values of its parameters and, potentially, the current context.

set_reward(reward, context) in set_reward(reward, context), a policy updates its parameter values based on the reward received, and, potentially, the current context.

References


See Also

Core contextual classes: Bandit, Policy, Simulator, Agent, History, Plot
Bandit subclass examples: BasicBernoulliBandit, ContextualLogitBandit, OfflineReplayEvaluatorBandit
Policy subclass examples: EpsilonGreedyPolicy, ContextualLinTSPolicy
value_remaining

Examples

```r
horizon <- 100L
simulations <- 100L
weights <- c(0.9, 0.1, 0.1)

policy <- UCB2Policy$new()
bandit <- BasicBernoulliBandit$new(weights = weights)
agent <- Agent$new(policy, bandit)

history <- Simulator$new(agent, horizon, simulations, do_parallel = FALSE)$run()

plot(history, type = "cumulative")

plot(history, type = "arms")
```

---

**value_remaining**

*Potential Value Remaining*

**Description**

Compute "value_remaining" in arms not currently best in binomial bandits

**Usage**

```r
value_remaining(x, n, alpha = 1, beta = 1, ndraws = 10000)
```

**Arguments**

- **x**: Vector of the number of successes per arm.
- **n**: Vector of the number of trials per arm.
- **alpha**: Shape parameter alpha for the prior beta distribution.
- **beta**: Shape parameter beta for the prior beta distribution.
- **ndraws**: Number of random draws from the posterior.

**Value**

Value_remaining distribution; the distribution of improvement amounts that another arm might have over the current best arm.

**Author(s)**

Thomas Lotze and Markus Loecher
Examples

```r
x <- c(10, 20, 30, 80)
n <- c(100, 102, 120, 240)
vr <- value_remaining(x, n)
hist(vr)

# "potential value" remaining in the experiment
potential_value <- quantile(vr, 0.95)
```

---

**var_welford**

*Welford’s variance*

---

**Description**

Welford described a method for ‘robust’ one-pass computation of the standard deviation. By ‘robust’, we mean robust to round-off caused by a large shift in the mean.

**Usage**

```r
var_welford(z)
```

**Arguments**

- `z` vector

**Value**

variance

---

**which_max_list**

*Get maximum value in list*

---

**Description**

Returns the index of the maximum value in list `x`.

**Usage**

```r
which_max_list(x, equal_is_random = TRUE)
```

**Arguments**

- `x` vector of values
- `equal_is_random` boolean
Details

If there is a tie and `equal_is_random` is TRUE, the index of one of the tied maxima is returned at random.

If `equal_is_random` is FALSE, the maximum with the lowest index number is returned.

Examples

```r
theta = list(par_one = list(1,2,3), par_two = list(2,3,4))
which_max_list(theta$par_one)
```

---

**which_max_tied**  
*Get maximum value randomly breaking ties*

Description

Returns the index of the maximum value in vector `vec`.

Usage

```r
which_max_tied(x, equal_is_random = TRUE)
```

Arguments

- `x`  
  vector of values

- `equal_is_random`  
  boolean

Details

If there is a tie, and `equal_is_random` is TRUE, the index of one of the tied maxima is returned at random. Otherwise, the value with the lowest index is returned.
Index

agent (Agent), 3
arms (Plot), 69
average (Plot), 69
bandit (Bandit), 5
BasicGaussianBandit, 8
BootstrapTSPolicy, 9
check_history_data (Plot), 69
clear_data_table (History), 37
clipr, 10
ContextualBernoulliBandit, 10, 11
ContextualBinaryBandit, 12, 12
ContextualEpochGreedyPolicy, 13
ContextualEpsilonGreedyPolicy, 13
ContextualHybridBandit, 14
ContextualLinearBandit, 16
ContextualLogitBTSPolicy, 20
ContextualPrecachingBandit, 21
ContextualTSProbitPolicy, 22
ContextualWheelBandit, 23
ContinuumBandit, 24
cumulative (History), 37
data_table_factors_to_numeric, 26
decc<, 26
dgamma, 41
dinvgamma (invgamma), 40
do_plot (Plot), 69
do_step (Agent), 3
EpsilonFirstPolicy, 27
Exp3Policy, 30, 81
FixedPolicy, 30
formatted_difftime, 32
generate_bandit_data (Bandit), 5
g_action (Policy), 72
g_get_arm_context, 33
go_get_context (Bandit), 5
go_get_data_frame (History), 37
go_get_data_table (History), 37
go_get_full_context, 33
get_t (Agent), 3
gg_color_hue (Plot), 69
gittinsbrezzilai (GittinsBrezziLaiPolicy), 34
GittinsBrezziLaiPolicy, 34
GradientPolicy, 35
inc<-, 39
ind, 39
initialize_theta (Policy), 72
inv, 40
invgamma, 40
invlogit, 41
is_rstudio, 42
LifPolicy, 42
LinUCBDisjointOptimizedPolicy, 43
LinUCBDisjointPolicy, 45
LinUCBGeneralPolicy, 46
LinUCBHybridOptimizedPolicy, 47
LinUCBHybridPolicy, 48
load (History), 37
mvrnorm, 49
OfflineBootstrappedReplayBandit, 50, 51
OfflineDirectMethodBandit, 52, 53
OfflineDoublyRobustBandit, 55, 56
OfflineLookupReplayEvaluatorBandit, 57, 59
OfflinePropensityWeightingBandit, 61, 62
OfflineReplayEvaluatorBandit, 4, 6–8, 10–15, 17, 18, 20, 22, 24, 25, 28, 29, 31, 32, 35, 36, 38, 43, 44, 46–49, 63, 65, 68, 70, 72–74, 76, 80, 82, 83, 85–87
one_hot, 67
ones_in_zeroes, 66
optimal (Plot), 69
OraclePolicy, 68

invgamma (invgamma), 40
plot, 4
post initialization (Bandit), 5
print . History (print . history), 72
print . history, 74
print . data (History), 37
prob_winner, 74
qinvgamma (invgamma), 40
RandomPolicy, 75
rinvgamma (invgamma), 40
run (Simulator), 78
sample_one_of, 76
save (History), 37
set_data_frame (History), 37
set_data_table (History), 37
set_external, 77
setParameters (Policy), 72
set_reward (Policy), 72
set_t (Agent), 3
sherman_morrisson, 78
sim_post, 80
simulator (Simulator), 78
SoftmaxPolicy, 81
sum of, 83
summary (History) (summary . history), 83
summary . history, 83
theta (Policy), 72
ThompsonSamplingPolicy, 84
UCB1Policy, 85
UCB2Policy, 87
value_remaining, 88
var_welford, 89
which_max_list, 89
which_max_tied, 90