Package: sagemaker.debugger (via r-universe)

August 23, 2024

Type Package

Title R6sagemaker debugger for sagemaker operations

Version 0.1.0

Description `R6sagemaker` debugger for sagemaker operations.

Imports lgr, R6, data.table, jsonlite, methods

License Apache License (>= 2.0)

Encoding UTF-8

RoxygenNote 7.1.2

Collate 'r_utils.R' 'actions_utils.R' 'actions_actions.R' 'debugger_rules_utils.R' 'debugger_rules_builtin_rules.R' 'debugger_rules_collections.R' 'debugger_rules_constants.R' 'debugger_rules_ruleGroups.R' 'profiler_rules_utils.R' 'profiler_rules_rules.R' 'zzz.R'

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Repository https://dyfanjones.r-universe.dev

RemoteUrl https://github.com/DyfanJones/sagemaker-r-debugger

RemoteRef HEAD

RemoteSha 2f10937153bf4cf70be72d1928ba7eb0ad4377ed

Contents

`sagemaker.debugger-package` *r6 sagemaker: this is just a placeholder*

Description

'R6sagemaker' debugger for sagemaker operations.

Author(s)

Maintainer: Dyfan Jones <dyfan.r.jones@gmail.com>

Other contributors:

- Amazon.com, Inc. [copyright holder]

`Action` *Debugger Action Class*

Description

Base class for action, which is to be invoked when a rule fires. Offers 'serialize' function to convert action parameters to a string dictionary.

Methods

Public methods:

- `Action$new()`
- `Action$serialize()`
- `Action$format()`
- `Action$clone()`

Method `new()`**:** This class is not meant to be initialized directly. Accepts dictionary of action parameters and drops keys whose values are 'None'.

Usage:

```
Action$new(...)
```

Arguments:

`...` : Dictionary of action parameters.

Method `serialize()`**:** Serialize the action parameters as a string dictionary.

Usage:

```
Action$serialize()
```

Returns: Action parameters serialized as a string dictionary.

Method `format()`**:** format class

Usage:

```
Action$format()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Action$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ActionList` *Debugger ActionList Action Class*

Description

Higher level object to maintain a list of actions to be invoked when a rule is fired.

Methods

Public methods:

- `ActionList$new()`
- `ActionList$update_training_job_prefix_if_not_specified()`
- `ActionList$serialize()`
- `ActionList$format()`
- `ActionList$clone()`

Method `new()`**:** Offers higher level 'serialize' function to handle serialization of actions as a string list of dictionaries.

Usage:

```
ActionList$new(...)
```

Arguments:

`...` : List of actions.

Method `update_training_job_prefix_if_not_specified()`**:** For any StopTraining actions in the action list, update the trainingjob prefix to be the training job name if the user has not already specified a custom training job prefix. This is meant to be called via the sagemaker SDK when 'estimator.fit' is called by the user. Validation is purposely excluded here so that any failures in validation of the training job name are intentionally caught in the sagemaker SDK and not here.

Usage:

```
ActionList$update_training_job_prefix_if_not_specified(training_job_name)
```

Arguments:

`training_job_name` : Name of the training job, passed in when 'estimator.fit' is called.

Method `serialize()`**:** Serialize the action parameters as a string dictionary.

Usage:

```
ActionList$serialize()
```

Returns: Action parameters serialized as a string dictionary.

Method `format()`**:** format class

Usage:

```
ActionList$format()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ActionList$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`BatchSize` *Debugger BatchSize class*

Description

This rule helps to detect if GPU is underulitized because of the batch size being too small. To detect this the rule analyzes the average GPU memory footprint, CPU and GPU utilization. If utilization on CPU, GPU and memory footprint is on average low , it may indicate that user can either run on a smaller instance type or that batch size could be increased. This analysis does not work for frameworks that heavily over-allocate memory. Increasing batch size could potentially lead to a processing/dataloading bottleneck, because more data needs to be pre-processed in each iteration.

Super class

`sagemaker.debugger::ProfilerRuleBase` -> `BatchSize`

Methods

Public methods:

- BatchSize$new()
- BatchSize$clone()

Method `new()`**:** Initialize BatchSize class

Usage:

```
BatchSize$new(
  cpu_threshold_p95 = 70,
  gpu_threshold_p95 = 70,
  gpu_memory_threshold_p95 = 70,
  patience = 1000,
  window = 500,
  scan_interval_us = 60 * 1000 * 1000
)
```

Arguments:

`cpu_threshold_p95` (numeric): defines the threshold for 95th quantile of CPU utilization.Default is 70%.

`gpu_threshold_p95` (numeric): defines the threshold for 95th quantile of GPU utilization.Default is 70%.

`gpu_memory_threshold_p95` (numeric): defines the threshold for 95th quantile of GPU memory utilization.Default is 70%.

`patience` (numeric): defines how many data points to capture before Rule runs the first evluation. Default 100

`window` (numeric): window size for computing quantiles.

`scan_interval_us` (numeric): interval with which timeline files are scanned. Default is 60000000 us.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
BatchSize$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CPUBottleneck` *Debugger CPUBottleneck class*

Description

This rule helps to detect if GPU is underutilized due to CPU bottlenecks. Rule returns True if number of CPU bottlenecks exceeds a predefined threshold.

Super class

`sagemaker.debugger::ProfilerRuleBase` -> `CPUBottleneck`

Methods

Public methods:

- `CPUBottleneck$new()`
- `CPUBottleneck$clone()`

Method `new()`**:** Initialize CPUBottleneck class

Usage:

```
CPUBottleneck$new(
  threshold = 50,
  gpu_threshold = 10,
  cpu_threshold = 90,
  patience = 1000,
  scan_interval_us = 60 * 1000 * 1000
)
```

Arguments:

`threshold` : defines the threshold beyond which Rule should return True. Default is 50 percent. So if there is a bottleneck more than 50% of the time during the training Rule will return True.

`gpu_threshold` : threshold that defines when GPU is considered being under-utilized. Default is 10%

`cpu_threshold` : threshold that defines high CPU utilization. Default is above 90%

`patience` : How many values to record before checking for CPU bottlenecks. During training initialization, GPU is likely at 0 percent, so Rule should not check for under utilization immediately. Default 1000.

`scan_interval_us` : interval with which timeline files are scanned. Default is 60000000 us.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CPUBottleneck$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Dataloader` *Debugger Dataloader class*

Description

This rule helps to detect how many dataloader processes are running in parallel and whether the total number is equal the number of available CPU cores.

Super class

sagemaker.debugger::ProfilerRuleBase -> Dataloader

Methods

Public methods:

- Dataloader$new()
- Dataloader$clone()

Method new(): Initialize Dataloader class

Usage:

```
Dataloader$new(
  min_threshold = 70,
  max_threshold = 200,
  scan_interval_us = 6e+07
)
```

Arguments:

min_threshold : how many cores should be at least used by dataloading processes. Default 70%

max_threshold : how many cores should be at maximum used by dataloading processes. Default 200%

scan_interval_us : interval with which timeline files are scanned. Default is 60000000 us.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
Dataloader$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Email *Debugger Email Action class*

Description

Action for sending an email to the provided email address when the rule is fired. Note that a policy must be created in the AWS account to allow the sagemaker role to send an email to the user.

Super class

sagemaker.debugger::Action -> Email

Methods

Public methods:

- Email$new()
- Email$clone()

Method `new()`**:** Initialize Email action class.

Usage:

```
Email$new(email_address)
```

Arguments:

`email_address` : Email address to send the email notification to.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Email$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`GPUMemoryIncrease` *Debugger GPUMemoryIncrease class*

Description

This rule helps to detect large increase in memory usage on GPUs. The rule computes the moving average of continous datapoints and compares it against the moving average of previous iteration.

Super class

`sagemaker.debugger::ProfilerRuleBase` -> `GPUMemoryIncrease`

Methods

Public methods:

- GPUMemoryIncrease$new()
- GPUMemoryIncrease$clone()

Method `new()`**:** Initialize GPUMemoryIncrease class

Usage:

```
GPUMemoryIncrease$new(
  increase = 5,
  patience = 1000,
  window = 10,
  scan_interval_us = 60 * 1000 * 1000
)
```

Arguments:

`increase` : defines the threshold for absolute memory increase.Default is 5%. So if moving average increase from 5% to 6%, the rule will fire.

`patience` : defines how many continous datapoints to capture before Rule runs the first evluation. Default is 1000

`window` : window size for computing moving average of continous datapoints

`scan_interval_us` : interval with which timeline files are scanned. Default is 60000000 us.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GPUMemoryIncrease$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`IOBottleneck` *Debugger IOBottleneck class*

Description

This rule helps to detect if GPU is underutilized due to IO bottlenecks. Rule returns True if number of IO bottlenecks exceeds a predefined threshold.

Super class

`sagemaker.debugger::ProfilerRuleBase` -> `IOBottleneck`

Methods

Public methods:

- `IOBottleneck$new()`
- `IOBottleneck$clone()`

Method `new()`**:** Initialize IOBottleneck class

Usage:

```
IOBottleneck$new(
  threshold = 50,
  gpu_threshold = 10,
  io_threshold = 50,
  patience = 1000,
  scan_interval_us = 60 * 1000 * 1000
)
```

Arguments:

`threshold` : defines the threshold when Rule should return True. Default is 50 percent. So if there is a bottleneck more than 50% of the time during the training Rule will return True.

gpu_threshold : threshold that defines when GPU is considered being under-utilized. Default is 70%

io_threshold : threshold that defines high IO wait time. Default is above 50%

patience : How many values to record before checking for IO bottlenecks. During training initilization, GPU is likely at 0 percent, so Rule should not check for underutilization immediatly. Default 1000.

scan_interval_us : interval with which timeline files are scanned. Default is 60000000 us.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
IOBottleneck$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

`LoadBalancing` *Debugger LoadBalancing class*

Description

This rule helps to detect issues in workload balancing between multiple GPUs. It computes a histogram of utilization per GPU and measures the distance between those histograms. If the histogram exceeds a pre-defined threshold then rule triggers.

Super class

`sagemaker.debugger::ProfilerRuleBase` -> `LoadBalancing`

Methods

Public methods:

- `LoadBalancing$new()`
- `LoadBalancing$clone()`

Method `new()`**:** Initialize LoadBalancing class

Usage:

```
LoadBalancing$new(
  threshold = 0.5,
  patience = 1000,
  scan_interval_us = 60 * 1000 * 1000
)
```

Arguments:

threshold : difference between 2 histograms 0.5

patience : how many values to record before checking for loadbalancing issues

scan_interval_us : interval with which timeline files are scanned. Default is 60000000 us.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LoadBalancing$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`LowGPUUtilization` *Debugger LowGPUUtilization class*

Description

This rule helps to detect if GPU utilization is low or suffers from fluctuations. This is checked for each single GPU on each worker node. Rule returns True if 95th quantile is below threshold_p95 which indicates under-utilization. Rule returns true if 95th quantile is above threshold_p95 and 5th quantile is below threshold_p5 which indicates fluctuations.

Super class

`sagemaker.debugger::ProfilerRuleBase` -> `LowGPUUtilization`

Methods

Public methods:

- `LowGPUUtilization$new()`
- `LowGPUUtilization$clone()`

Method `new()`**:** Initialize LowGPUUtilization class

Usage:

```
LowGPUUtilization$new(
  threshold_p95 = 70,
  threshold_p5 = 10,
  window = 500,
  patience = 1000,
  scan_interval_us = 60 * 1000 * 1000
)
```

Arguments:

`threshold_p95` : threshold for 95th quantile below which GPU is considered to be underutilized. Default is 70 percent.

`threshold_p5` : threshold for 5th quantile. Default is 10 percent.

`window` : number of past datapoints which are used to compute the quantiles.

`patience` : How many values to record before checking for underutilization/fluctuations. During training initilization, GPU is likely at 0 percent, so Rule should not check for underutilization immediately. Default 1000.

`scan_interval_us` : interval with which timeline files are scanned. Default is 60000000 us.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LowGPUUtilization$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`MaxInitializationTime` *Debugger MaxInitializationTime class*

Description

This rule helps to detect if the training intialization is taking too much time. The rule waits until first step is available.

Super class

`sagemaker.debugger::ProfilerRuleBase` -> `MaxInitializationTime`

Methods

Public methods:

- `MaxInitializationTime$new()`
- `MaxInitializationTime$clone()`

Method `new()`**:** Initialize MaxInitializationTime class

Usage:

```
MaxInitializationTime$new(threshold = 20, scan_interval_us = 60 * 1000 * 1000)
```

Arguments:

`threshold` : defines the threshold in minutes to wait for first step to become available. Default is 20 minutes.

`scan_interval_us` : interval with which timeline files are scanned. Default is 60000000 us.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MaxInitializationTime$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`OverallSystemUsage` *Debugger OverallSystemUsage class*

Description

This rule measures overall system usage per worker node. The rule currently only aggregates values per node and computes their percentiles. The rule does currently not take any threshold parameters into account nor can it trigger. The reason behind that is that other rules already cover cases such as under utilization and they do it at a more fine-grained level e.g. per GPU. We may change this in the future.

Super class

`sagemaker.debugger::ProfilerRuleBase` -> `OverallSystemUsage`

Methods

Public methods:

- `OverallSystemUsage$new()`
- `OverallSystemUsage$clone()`

Method `new()`**:** Initialize OverallSystemUsage class

Usage:

```
OverallSystemUsage$new(scan_interval_us = 60 * 1000 * 1000)
```

Arguments:

`scan_interval_us` : interval with which timeline files are scanned. Default is 60000000 us.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
OverallSystemUsage$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ProfilerReport` *Debugger ProfilerReport class*

Description

This rule will create a profiler report after invoking all of the rules. The parameters used in any of these rules can be customized by following this naming scheme: <rule_name>_<parameter_name> : value Validation is also done here to ensure that:

- The key names follow the above format
- rule_name corresponds to a valid rule name.
- parameter_name corresponds to a valid parameter of this rule.
- The parameter for this rule's parameter is valid.

Super class

sagemaker.debugger::ProfilerRuleBase -> ProfilerReport

Methods

Public methods:

- ProfilerReport$new()
- ProfilerReport$clone()

Method new(): Initialize ProfilerReport class

Usage:

```
ProfilerReport$new(...)
```

Arguments:

... : Dictionary mapping rule + parameter name to value.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
ProfilerReport$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

rule_config *List of Debugger Built-in Rules*

Description

Use the Debugger built-in rules provided by Amazon SageMaker Debugger and analyze tensors emitted while training your models. The Debugger built-in rules monitor various common conditions that are critical for the success of a training job. You can call the built-in rules using Amazon SageMaker Python SDK or the low-level SageMaker API operations. Depending on deep learning frameworks of your choice, there are four scopes of validity for the built-in rules as shown in the following table. https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-built-in-rules.html

Usage

```
vanishing_gradient()

similar_across_runs()

weight_update_ratio()

all_zero()
```

```
exploding_tensor()

unchanged_tensor()

loss_not_decreasing()

check_input_images()

dead_relu()

confusion()

tree_depth()

class_imbalance()

overfit()

tensor_variance()

overtraining()

poor_weight_initialization()

saturated_activation()

nlp_sequence_ratio()

stalled_training_rule()

feature_importance_overweight()

create_xgboost_report()
```

Value

list to be used in Amazon SageMaker Debugger

SMS *Debugger SMS Action Class*

Description

Action for sending an SMS to the provided phone number when the rule is fired. Note that a policy must be created in the AWS account to allow the sagemaker role to send an SMS to the user.

Super class

sagemaker.debugger::Action -> SMS

Methods

Public methods:

- SMS$new()
- SMS$clone()

Method new(): Initialize SMS action class

Usage:

```
SMS$new(phone_number)
```

Arguments:

phone_number : Valid phone number that follows the the E.164 format. See https://docs.aws.amazon.com/sns/latest/dg/sms_publish-to-phone.html for more info.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
SMS$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

StepOutlier *Debugger StepOutlier class*

Description

This rule helps to detect outlier in step durations. Rule returns True if duration is larger than stddev * standard deviation.

Super class

sagemaker.debugger::ProfilerRuleBase -> StepOutlier

Methods

Public methods:

- StepOutlier$new()
- StepOutlier$clone()

Method new(): Initialize StepOutlier class

Usage:

```
StepOutlier$new(
  stddev = 3,
  mode = NULL,
  n_outliers = 10,
  scan_interval_us = 60 * 1000 * 1000
)
```

Arguments:

`stddev` : factor by which to multiply the standard deviation. Default is 3

`mode` : select mode under which steps have been saved and on which Rule should run on. Per default rule will run on steps from EVAL and TRAIN phase.

`n_outliers` : How many outliers to ignore before rule returns True. Default 10.

`scan_interval_us` : interval with which timeline files are scanned. Default is 60000000 us.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
StepOutlier$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`StopTraining` *Debugger StopTraining Action class*

Description

Action for stopping the training job when a rule is fired.

Super class

`sagemaker.debugger::Action` -> `StopTraining`

Methods

Public methods:

- `StopTraining$new()`
- `StopTraining$update_training_job_prefix_if_not_specified()`
- `StopTraining$clone()`

Method `new()`**:** Note that a policy must be created in the AWS account to allow the sagemaker role to stop the training job.

Usage:

```
StopTraining$new(training_job_prefix = NULL)
```

Arguments:

`training_job_prefix` : The prefix of the training job to stop if the rule is fired. This must only refer to one active training job, otherwise no training job will be stopped.

Method `update_training_job_prefix_if_not_specified()`**:** Update the training job prefix to be the training job name if the user has not already specified a custom training job prefix. This is only meant to be called via the sagemaker SDK when 'estimator.fit' is called by the user. Validation is purposely excluded here so that any failures in validation of the training job name are intentionally caught in the sagemaker SDK and not here.

Usage:

```
StopTraining$update_training_job_prefix_if_not_specified(training_job_name)
```

Arguments:

`training_job_name` : Name of the training job, passed in when 'estimator.fit' is called.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
StopTraining$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index