

# Package: sagemaker.mlframework (via r-universe)

October 9, 2026

**Type** Package

**Title** sagemaker machine learning developed by amazon

**Version** 0.2.0

**Description** `sagemaker` machine learning developed by amazon.

**Imports** http, lgr, jsonlite, methods, processx, fs, R6,  
sagemaker.core, sagemaker.common, sagemaker.mlcore, urltools,  
stats

**Suggests** crayon, testthat, mockthat, reticulate, RProtoBuf, readsparse

**Remotes** DyfanJones/sagemaker-r-core, DyfanJones/sagemaker-r-common,  
DyfanJones/sagemaker-r-mlcore

**License** Apache License (>= 2.0)

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**Collate** 'r\_utils.R' 'amazon\_factorization\_machines.R'  
'amazon\_ipinsights.R' 'amazon\_kmeans.R' 'amazon\_knn.R'  
'amazon\_lda.R' 'amazon\_linear\_learner.R' 'amazon\_ntm.R'  
'amazon\_object2vec.R' 'amazon\_pca.R' 'amazon\_randomcutforest.R'  
'automl\_candidate\_estimator.R' 'automl\_automl.R'  
'chainer\_default.R' 'chainer\_model.R' 'chainer\_estimator.R'  
'huggingface\_estimator.R' 'huggingface\_model.R'  
'huggingface\_processing.R' 'mxnet\_default.R' 'mxnet\_model.R'  
'mxnet\_estimator.R' 'mxnet\_processing.R' 'pytorch\_defaults.R'  
'pytorch\_model.R' 'pytorch\_estimator.R' 'pytorch\_processing.R'  
'tensorflow\_model.R' 'tensorflow\_defaults.R'  
'tensorflow\_estimator.R' 'rl\_estimator.R' 'sklearn\_model.R'  
'sklearn\_estimator.R' 'sklearn\_processing.R'  
'spark\_processing.R' 'sparkml\_model.R'  
'tensorflow\_processing.R' 'xgboost\_default.R' 'xgboost\_utils.R'  
'xgboost\_model.R' 'xgboost\_estimator.R' 'xgboost\_processing.R'  
'zzz.R'

**Config/pak/sysreqs** cmake make libuv1-dev libxml2-dev libssl-dev

**Repository** <https://dyfanjones.r-universe.dev>

**Date/Publication** 2022-03-08 18:32:12 UTC

**RemoteUrl** <https://github.com/DyfanJones/sagemaker-r-mlframework>

**RemoteRef** HEAD

**RemoteSha** e1988da8e0767e790b484229556dc8bbcd5017a

## Contents

|                                          |    |
|------------------------------------------|----|
| sagemaker.mlframework-package . . . . .  | 3  |
| .SparkProcessorBase . . . . .            | 4  |
| AutoML . . . . .                         | 7  |
| AutoMLInput . . . . .                    | 13 |
| AutoMLJob . . . . .                      | 14 |
| CandidateEstimator . . . . .             | 15 |
| CandidateStep . . . . .                  | 17 |
| Chainer . . . . .                        | 18 |
| ChainerModel . . . . .                   | 21 |
| ChainerPredictor . . . . .               | 23 |
| FactorizationMachines . . . . .          | 24 |
| FactorizationMachinesModel . . . . .     | 28 |
| FactorizationMachinesPredictor . . . . . | 29 |
| HuggingFace . . . . .                    | 30 |
| HuggingFaceModel . . . . .               | 33 |
| HuggingFacePredictor . . . . .           | 36 |
| HuggingFaceProcessor . . . . .           | 37 |
| IPInsights . . . . .                     | 39 |
| IPInsightsModel . . . . .                | 42 |
| IPInsightsPredictor . . . . .            | 43 |
| KMeans . . . . .                         | 44 |
| KMeansModel . . . . .                    | 47 |
| KMeansPredictor . . . . .                | 48 |
| KNN . . . . .                            | 49 |
| KNNModel . . . . .                       | 51 |
| KNNPredictor . . . . .                   | 52 |
| LDA . . . . .                            | 53 |
| LDAModel . . . . .                       | 56 |
| LDAPredictor . . . . .                   | 57 |
| LinearLearner . . . . .                  | 58 |
| LinearLearnerModel . . . . .             | 64 |
| LinearLearnerPredictor . . . . .         | 65 |
| MXNet . . . . .                          | 66 |
| MXNetModel . . . . .                     | 68 |
| MXNetPredictor . . . . .                 | 70 |
| MXNetProcessor . . . . .                 | 71 |
| NTM . . . . .                            | 73 |
| NTMModel . . . . .                       | 76 |

|                                    |     |
|------------------------------------|-----|
| NTMPredictor . . . . .             | 77  |
| Object2Vec . . . . .               | 78  |
| Object2VecModel . . . . .          | 82  |
| PCA . . . . .                      | 83  |
| PCAModel . . . . .                 | 86  |
| PCAPredictor . . . . .             | 87  |
| PySparkProcessor . . . . .         | 88  |
| PyTorch . . . . .                  | 91  |
| PyTorchModel . . . . .             | 94  |
| PyTorchPredictor . . . . .         | 97  |
| PyTorchProcessor . . . . .         | 98  |
| RandomCutForest . . . . .          | 100 |
| RandomCutForestModel . . . . .     | 102 |
| RandomCutForestPredictor . . . . . | 103 |
| RLEstimator . . . . .              | 104 |
| RLFramework . . . . .              | 107 |
| RLToolkit . . . . .                | 107 |
| SKLearn . . . . .                  | 108 |
| SKLearnModel . . . . .             | 110 |
| SKLearnPredictor . . . . .         | 112 |
| SKLearnProcessor . . . . .         | 113 |
| SparkJarProcessor . . . . .        | 115 |
| SparkMLModel . . . . .             | 118 |
| SparkMLPredictor . . . . .         | 119 |
| TensorFlow . . . . .               | 120 |
| TensorFlowModel . . . . .          | 124 |
| TensorFlowPredictor . . . . .      | 128 |
| TensorFlowProcessor . . . . .      | 130 |
| XGBoost . . . . .                  | 132 |
| XGBoostModel . . . . .             | 135 |
| XGBoostPredictor . . . . .         | 137 |
| XGBoostProcessor . . . . .         | 138 |

**Index****140**


---

sagemaker.mlframework-package

*r6 sagemaker: this is just a placeholder*


---

**Description**

‘sagemaker’ machine learning developed by amazon.

**Author(s)**

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

Other contributors:

- Amazon.com, Inc. [copyright holder]

---

|                     |                                                                        |
|---------------------|------------------------------------------------------------------------|
| .SparkProcessorBase | <i>Handles Amazon SageMaker processing tasks for jobs using Spark.</i> |
|---------------------|------------------------------------------------------------------------|

---

**Description**

Base class for either PySpark or SparkJars.

**Super classes**

`sagemaker.common:Processor` -> `sagemaker.common:ScriptProcessor` -> `.SparkProcessorBase`

**Methods****Public methods:**

- `.SparkProcessorBase$new()`
- `.SparkProcessorBase$get_run_args()`
- `.SparkProcessorBase$run()`
- `.SparkProcessorBase$start_history()`
- `.SparkProcessorBase$terminate_history_server()`
- `.SparkProcessorBase$clone()`

**Method** `new()`: Initialize a “\_SparkProcessorBase” instance. The \_SparkProcessorBase handles Amazon SageMaker processing tasks for jobs using SageMaker Spark.

*Usage:*

```
.SparkProcessorBase$new(
  role,
  instance_type,
  instance_count,
  framework_version = NULL,
  py_version = NULL,
  container_version = NULL,
  image_uri = NULL,
  volume_size_in_gb = 30,
  volume_kms_key = NULL,
  output_kms_key = NULL,
  max_runtime_in_seconds = NULL,
  base_job_name = NULL,
  sagemaker_session = NULL,
  env = NULL,
```

```

        tags = NULL,
        network_config = NULL
    )

```

*Arguments:*

**role** (str): An AWS IAM role name or ARN. The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

**instance\_type** (str): Type of EC2 instance to use for processing, for example, 'ml.c4.xlarge'.

**instance\_count** (int): The number of instances to run the Processing job with. Defaults to 1.

**framework\_version** (str): The version of SageMaker PySpark.

**py\_version** (str): The version of python.

**container\_version** (str): The version of spark container.

**image\_uri** (str): The container image to use for training.

**volume\_size\_in\_gb** (int): Size in GB of the EBS volume to use for storing data during processing (default: 30).

**volume\_kms\_key** (str): A KMS key for the processing volume.

**output\_kms\_key** (str): The KMS key id for all ProcessingOutputs.

**max\_runtime\_in\_seconds** (int): Timeout in seconds. After this amount of time Amazon SageMaker terminates the job regardless of its current status.

**base\_job\_name** (str): Prefix for processing name. If not specified, the processor generates a default job name, based on the training image name and current timestamp.

**sagemaker\_session** (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the processor creates one using the default AWS configuration chain.

**env** (dict): Environment variables to be passed to the processing job.

**tags** ([dict]): List of tags to be passed to the processing job. **network\_config** (sagemaker.network.NetworkConfig): A NetworkConfig object that configures network isolation, encryption of inter-container traffic, security group IDs, and subnets.

**network\_config** (sagemaker.network.NetworkConfig): A NetworkConfig object that configures network isolation, encryption of inter-container traffic, security group IDs, and subnets.

**Method** `get_run_args()`: For processors (:class:`~sagemaker.spark.processing.PySparkProcessor`, :class:`~sagemaker.spark.processing.SparkJar`) that have special `run()` arguments, this object contains the normalized arguments for passing to :class:`~sagemaker.workflow.steps.ProcessingStep`.

*Usage:*

```

SparkProcessorBase$get_run_args(
    code,
    inputs = NULL,
    outputs = NULL,
    arguments = NULL
)

```

*Arguments:*

**code** (str): This can be an S3 URI or a local path to a file with the framework script to run.

`inputs` (list[:class:~sagemaker.processing.ProcessingInput]): Input files for the processing job. These must be provided as :class:~sagemaker.processing.ProcessingInput objects (default: None).

`outputs` (list[:class:~sagemaker.processing.ProcessingOutput]): Outputs for the processing job. These can be specified as either path strings or :class:~sagemaker.processing.ProcessingOutput objects (default: None).

`arguments` (list[str]): A list of string arguments to be passed to a processing job (default: None).

*Returns:* Returns a RunArgs object.

**Method** `run()`: Runs a processing job.

*Usage:*

```
.SparkProcessorBase$.run(
  submit_app,
  inputs = NULL,
  outputs = NULL,
  arguments = NULL,
  wait = TRUE,
  logs = TRUE,
  job_name = NULL,
  experiment_config = NULL,
  kms_key = NULL
)
```

*Arguments:*

`submit_app` (str): .py or .jar file to submit to Spark as the primary application

`inputs` (list[:class:~sagemaker.processing.ProcessingInput]): Input files for the processing job. These must be provided as :class:~sagemaker.processing.ProcessingInput objects (default: None).

`outputs` (list[:class:~sagemaker.processing.ProcessingOutput]): Outputs for the processing job. These can be specified as either path strings or :class:~sagemaker.processing.ProcessingOutput objects (default: None).

`arguments` (list[str]): A list of string arguments to be passed to a processing job (default: None).

`wait` (bool): Whether the call should wait until the job completes (default: True).

`logs` (bool): Whether to show the logs produced by the job. Only meaningful when wait is True (default: True).

`job_name` (str): Processing job name. If not specified, the processor generates a default job name, based on the base job name and current timestamp.

`experiment_config` (dict[str, str]): Experiment management configuration. Dictionary contains three optional keys: 'ExperimentName', 'TrialName', and 'TrialComponentDisplayName'.

`kms_key` (str): The ARN of the KMS key that is used to encrypt the user code file (default: None).

**Method** `start_history()`: Starts a Spark history server.

*Usage:*

```
.SparkProcessorBase$start_history(spark_event_logs_s3_uri = NULL)
```

*Arguments:*

spark\_event\_logs\_s3\_uri (str): S3 URI where Spark events are stored.

**Method** terminate\_history\_server(): Terminates the Spark history server.

*Usage:*

```
.SparkProcessorBase$terminate_history_server()
```

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

*Usage:*

```
.SparkProcessorBase$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

AutoML

*AutoML Class*


---

## Description

A class for creating and interacting with SageMaker AutoML jobs.

## Methods

### Public methods:

- [AutoML\\$new\(\)](#)
- [AutoML\\$fit\(\)](#)
- [AutoML\\$attach\(\)](#)
- [AutoML\\$describe\\_auto\\_ml\\_job\(\)](#)
- [AutoML\\$best\\_candidate\(\)](#)
- [AutoML\\$list\\_candidates\(\)](#)
- [AutoML\\$create\\_model\(\)](#)
- [AutoML\\$deploy\(\)](#)
- [AutoML\\$validate\\_and\\_update\\_inference\\_response\(\)](#)
- [AutoML\\$format\(\)](#)
- [AutoML\\$clone\(\)](#)

**Method** new(): Initialize AutoML class Place holder doc string

*Usage:*

```

AutoML$new(
  role,
  target_attribute_name,
  output_kms_key = NULL,
  output_path = NULL,
  base_job_name = NULL,
  compression_type = NULL,
  sagemaker_session = NULL,
  volume_kms_key = NULL,
  encrypt_inter_container_traffic = FALSE,
  vpc_config = NULL,
  problem_type = NULL,
  max_candidates = NULL,
  max_runtime_per_training_job_in_seconds = NULL,
  total_job_runtime_in_seconds = NULL,
  job_objective = NULL,
  generate_candidate_definitions_only = FALSE,
  tags = NULL
)

```

*Arguments:*

```

role :
target_attribute_name :
output_kms_key :
output_path :
base_job_name :
compression_type :
sagemaker_session :
volume_kms_key :
encrypt_inter_container_traffic :
vpc_config :
problem_type :
max_candidates :
max_runtime_per_training_job_in_seconds :
total_job_runtime_in_seconds :
job_objective :
generate_candidate_definitions_only :
tags :

```

**Method** `fit()`: Create an AutoML Job with the input dataset.

*Usage:*

```
AutoML$fit(inputs = NULL, wait = TRUE, logs = TRUE, job_name = NULL)
```

*Arguments:*

`inputs` (str or list[str] or AutoMLInput): Local path or S3 Uri where the training data is stored.  
 Or an AutoMLInput object. If a local path is provided, the dataset will be uploaded to an S3 location.

`wait` (bool): Whether the call should wait until the job completes (default: True).

`logs` (bool): Whether to show the logs produced by the job. Only meaningful when `wait` is True (default: True). If “wait” is False, “logs” will be set to False as well.

`job_name` (str): Training job name. If not specified, the estimator generates a default job name, based on the training image name and current timestamp.

**Method** `attach()`: Attach to an existing AutoML job. Creates and returns a AutoML bound to an existing automl job.

*Usage:*

```
AutoML$attach(auto_ml_job_name, sagemaker_session = NULL)
```

*Arguments:*

`auto_ml_job_name` (str): AutoML job name

`sagemaker_session` (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, the one originally associated with the “AutoML” instance is used.

*Returns:* `sagemaker.automl.AutoML`: A “AutoML” instance with the attached automl job.

**Method** `describe_auto_ml_job()`: Returns the job description of an AutoML job for the given job name.

*Usage:*

```
AutoML$describe_auto_ml_job(job_name = NULL)
```

*Arguments:*

`job_name` (str): The name of the AutoML job to describe. If None, will use object’s `latest_auto_ml_job_name`.

*Returns:* dict: A dictionary response with the AutoML Job description.

**Method** `best_candidate()`: Returns the best candidate of an AutoML job for a given name.

*Usage:*

```
AutoML$best_candidate(job_name = NULL)
```

*Arguments:*

`job_name` (str): The name of the AutoML job. If None, will use object’s `current_auto_ml_job_name`.

*Returns:* dict: A dictionary with information of the best candidate.

**Method** `list_candidates()`: Returns the list of candidates of an AutoML job for a given name.

*Usage:*

```
AutoML$list_candidates(
  job_name = NULL,
  status_equals = NULL,
  candidate_name = NULL,
  candidate_arn = NULL,
  sort_order = NULL,
  sort_by = NULL,
  max_results = NULL
)
```

*Arguments:*

`job_name` (str): The name of the AutoML job. If None, will use object's `.current_job` name.

`status_equals` (str): Filter the result with candidate status, values could be "Completed", "InProgress", "Failed", "Stopped", "Stopping"

`candidate_name` (str): The name of a specified candidate to list. Default to None.

`candidate_arn` (str): The Arn of a specified candidate to list. Default to None.

`sort_order` (str): The order that the candidates will be listed in result. Default to None.

`sort_by` (str): The value that the candidates will be sorted by. Default to None.

`max_results` (int): The number of candidates will be listed in results, between 1 to 100. Default to None. If None, will return all the candidates.

*Returns:* list: A list of dictionaries with candidates information.

**Method** `create_model()`: Creates a model from a given candidate or the best candidate from the job.

*Usage:*

```
AutoML$create_model(
  name,
  sagemaker_session = NULL,
  candidate = NULL,
  vpc_config = NULL,
  enable_network_isolation = FALSE,
  model_kms_key = NULL,
  predictor_cls = NULL,
  inference_response_keys = NULL
)
```

*Arguments:*

`name` (str): The pipeline model name.

`sagemaker_session` (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, the one originally associated with the "AutoML" instance is used.:

`candidate` (CandidateEstimator or dict): a CandidateEstimator used for deploying to a SageMaker Inference Pipeline. If None, the best candidate will be used. If the candidate input is a dict, a CandidateEstimator will be created from it.

`vpc_config` (dict): Specifies a VPC that your training jobs and hosted models have access to. Contents include "SecurityGroupIds" and "Subnets".

`enable_network_isolation` (bool): Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. Default: False

`model_kms_key` (str): KMS key ARN used to encrypt the repacked model archive file if the model is repacked

`predictor_cls` (callable[string, sagemaker.session.Session]): A function to call to create a predictor (default: None). If specified, "deploy()" returns the result of invoking this function on the created endpoint name.

`inference_response_keys` (list): List of keys for response content. The order of the keys will dictate the content order in the response.

*Returns:* PipelineModel object.

**Method** `deploy()`: Deploy a candidate to a SageMaker Inference Pipeline.

*Usage:*

```
AutoML$deploy(
  initial_instance_count,
  instance_type,
  serializer = NULL,
  deserializer = NULL,
  candidate = NULL,
  sagemaker_session = NULL,
  name = NULL,
  endpoint_name = NULL,
  tags = NULL,
  wait = TRUE,
  vpc_config = NULL,
  enable_network_isolation = FALSE,
  model_kms_key = NULL,
  predictor_cls = NULL,
  inference_response_keys = NULL
)
```

*Arguments:*

`initial_instance_count` (int): The initial number of instances to run in the “Endpoint” created from this “Model”.

`instance_type` (str): The EC2 instance type to deploy this Model to. For example, ‘ml.p2.xlarge’.

`serializer` (:class:~sagemaker.serializers.BaseSerializer): A serializer object, used to encode data for an inference endpoint (default: None). If “serializer” is not None, then “serializer” will override the default serializer. The default serializer is set by the “predictor\_cls”.

`deserializer` (:class:~sagemaker.deserializers.BaseDeserializer): A deserializer object, used to decode data from an inference

`candidate` (CandidateEstimator or dict): a CandidateEstimator used for deploying to a SageMaker Inference Pipeline. If None, the best candidate will be used. If the candidate input is a dict, a CandidateEstimator will be created from it.

`sagemaker_session` (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, the one originally associated with the “AutoML” instance is used.

`name` (str): The pipeline model name. If None, a default model name will be selected on each “deploy”.

`endpoint_name` (str): The name of the endpoint to create (default: None). If not specified, a unique endpoint name will be created.

`tags` (List[dict[str, str]]): The list of tags to attach to this specific endpoint.

`wait` (bool): Whether the call should wait until the deployment of model completes (default: True).

`vpc_config` (dict): Specifies a VPC that your training jobs and hosted models have access to. Contents include “SecurityGroupIds” and “Subnets”.

`enable_network_isolation` (bool): Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. Default: False

`model_kms_key` (str): KMS key ARN used to encrypt the repacked model archive file if the model is repacked

`predictor_cls` (callable[string, sagemaker.session.Session]): A function to call to create a predictor (default: None). If specified, “`deploy()`” returns the result of invoking this function on the created endpoint name.

`inference_response_keys` (list): List of keys for response content. The order of the keys will dictate the content order in the response.

`endpoint` (default: None). If “`deserializer`” is not None, then “`deserializer`” will override the default deserializer. The default deserializer is set by the “`predictor_cls`”.

*Returns:* callable[string, sagemaker.session.Session] or “None”: If “`predictor_cls`” is specified, the invocation of “`self.predictor_cls`” on the created endpoint name. Otherwise, “None”.

**Method** `validate_and_update_inference_response()`: Validates the requested inference keys and updates response content. On validation, also updates the inference containers to emit appropriate response content in the inference response.

*Usage:*

```
AutoML$validate_and_update_inference_response(
    inference_containers,
    inference_response_keys
)
```

*Arguments:*

`inference_containers` (list): list of inference containers

`inference_response_keys` (list): list of inference response keys

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

*Usage:*

```
AutoML$format()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

AutoMLInput*Accepts parameters that specify an S3 input for an auto ml job*

---

**Description**

Provides a method to turn those parameters into a dictionary.

**Methods****Public methods:**

- [AutoMLInput\\$new\(\)](#)
- [AutoMLInput\\$to\\_request\\_list\(\)](#)
- [AutoMLInput\\$format\(\)](#)
- [AutoMLInput\\$clone\(\)](#)

**Method** `new()`: Convert an S3 Uri or a list of S3 Uri to an AutoMLInput object.

*Usage:*

```
AutoMLInput$new(inputs, target_attribute_name, compression = NULL)
```

*Arguments:*

`inputs` (str, list[str]): a string or a list of string that points to (a) S3 location(s) where input data is stored.

`target_attribute_name` (str): the target attribute name for regression or classification.

`compression` (str): if training data is compressed, the compression type. The default value is None.

**Method** `to_request_list()`: Generates a request dictionary using the parameters provided to the class.

*Usage:*

```
AutoMLInput$to_request_list()
```

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

*Usage:*

```
AutoMLInput$format()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

AutoMLJob

*AutoMLJob class***Description**

A class for interacting with CreateAutoMLJob API.

**Super class**

`sagemaker.common::Job` -> AutoMLJob

**Methods****Public methods:**

- `AutoMLJob$new()`
- `AutoMLJob$start_new()`
- `AutoMLJob$describe()`
- `AutoMLJob$wait()`
- `AutoMLJob$format()`
- `AutoMLJob$clone()`

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

*Usage:*

`AutoMLJob$new(sagemaker_session, job_name = NULL, inputs = NULL)`

*Arguments:*

`sagemaker_session` (`sagemaker.session.Session`): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, the one originally associated with the “AutoMLJob” instance is used.

`job_name` :

`inputs` (`str`, `list[str]`): Parameters used when called :meth:‘~sagemaker.automl.AutoML.fit’.

**Method** `start_new()`: Create a new Amazon SageMaker AutoML job from `auto_ml`.

*Usage:*

`AutoMLJob$start_new(auto_ml, inputs)`

*Arguments:*

`auto_ml` (`sagemaker.automl.AutoML`): AutoML object created by the user.

`inputs` (`str`, `list[str]`): Parameters used when called :meth:‘~sagemaker.automl.AutoML.fit’.

*Returns:* `sagemaker.automl.AutoMLJob`: Constructed object that captures all information about the started AutoML job.

**Method** `describe()`: Prints out a response from the DescribeAutoMLJob API call.

*Usage:*

`AutoMLJob$describe()`

**Method** wait(): Wait for the AutoML job to finish.

*Usage:*

```
AutoMLJob$wait(logs = TRUE)
```

*Arguments:*

logs (bool): indicate whether to output logs.

**Method** format(): format class

*Usage:*

```
AutoMLJob$format()
```

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

---

|                    |                                 |
|--------------------|---------------------------------|
| CandidateEstimator | <i>CandidateEstimator Class</i> |
|--------------------|---------------------------------|

---

## Description

A class for SageMaker AutoML Job Candidate

## Methods

### Public methods:

- [CandidateEstimator\\$new\(\)](#)
- [CandidateEstimator\\$get\\_steps\(\)](#)
- [CandidateEstimator\\$fit\(\)](#)
- [CandidateEstimator\\$format\(\)](#)
- [CandidateEstimator\\$clone\(\)](#)

**Method** new(): Constructor of CandidateEstimator.

*Usage:*

```
CandidateEstimator$new(candidate, sagemaker_session = NULL)
```

*Arguments:*

candidate (dict): a dictionary of candidate returned by AutoML.list\_candidates() or AutoML.best\_candidate().  
 sagemaker\_session (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, one is created using the default AWS configuration chain.

**Method** get\_steps(): Get the step job of a candidate so that users can construct estimators/transformers

*Usage:*

```
CandidateEstimator$get_steps()
```

*Returns:* list: a list of dictionaries that provide information about each step job's name, type, inputs and description

**Method** `fit()`: Rerun a candidate's step jobs with new input datasets or security config.

*Usage:*

```
CandidateEstimator$fit(
  inputs,
  candidate_name = NULL,
  volume_kms_key = NULL,
  encrypt_inter_container_traffic = FALSE,
  vpc_config = NULL,
  wait = TRUE,
  logs = TRUE
)
```

*Arguments:*

`inputs` (str or list[str]): Local path or S3 Uri where the training data is stored. If a local path is provided, the dataset will be uploaded to an S3 location.

`candidate_name` (str): name of the candidate to be rerun, if None, candidate's original name will be used.

`volume_kms_key` (str): The KMS key id to encrypt data on the storage volume attached to the ML compute instance(s).

`encrypt_inter_container_traffic` (bool): To encrypt all communications between ML compute instances in distributed training. Default: False.

`vpc_config` (dict): Specifies a VPC that jobs and hosted models have access to. Control access to and from training and model containers by configuring the VPC

`wait` (bool): Whether the call should wait until all jobs completes (default: True).

`logs` (bool): Whether to show the logs produced by the job. Only meaningful when wait is True (default: True).

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

*Usage:*

```
CandidateEstimator$format()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

CandidateStepCandidateStep Class

---

**Description**

A class that maintains an AutoML Candidate step's name, inputs, type, and description.

**Public fields**

name Name of the candidate step -> (str)

inputs Inputs of the candidate step -> (dict)

type Type of the candidate step, Training or Transform -> (str)

description Description of candidate step job -> (dict)

**Methods****Public methods:**

- [CandidateStep\\$new\(\)](#)
- [CandidateStep\\$format\(\)](#)
- [CandidateStep\\$clone\(\)](#)

**Method** new(): Initialize CandidateStep Class

*Usage:*

CandidateStep\$new(name, inputs, step\_type, description)

*Arguments:*

name (str): Name of the candidate step

inputs (dict): Inputs of the candidate step

step\_type (str): Type of the candidate step, Training or Transform

description (dict): Description of candidate step job

**Method** format(): format class

*Usage:*

CandidateStep\$format()

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

*Usage:*

CandidateStep\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

Chainer

*Chainer Class***Description**

Handle end-to-end training and deployment of custom Chainer code.

**Super classes**

`sagemaker.mlcore::EstimatorBase` -> `sagemaker.mlcore::Framework` -> Chainer

**Public fields**

`.use_mpi` Entry point is run as an MPI script.  
`.num_processes` Total number of processes to run the entry point with  
`.process_slots_per_host` The number of processes that can run on each instance.  
`.additional_mpi_options` String of options to the 'mpirun' command used to run the entry point.  
`.module` mimic python module

**Methods****Public methods:**

- `Chainer$new()`
- `Chainer$hyperparameters()`
- `Chainer$create_model()`
- `Chainer$clone()`

**Method new():** This “Estimator“ executes an Chainer script in a managed Chainer execution environment, within a SageMaker Training Job. The managed Chainer environment is an Amazon-built Docker container that executes functions defined in the supplied “entry\_point“ Python script. Training is started by calling :meth:`~sagemaker.amazon.estimator.Framework.fit` on this Estimator. After training is complete, calling :meth:`~sagemaker.amazon.estimator.Framework.deploy` creates a hosted SageMaker endpoint and returns an :class:`~sagemaker.amazon.chainer.model.ChainerPredictor` instance that can be used to perform inference against the hosted model. Technical documentation on preparing Chainer scripts for SageMaker training and using the Chainer Estimator is available on the project home-page: <https://github.com/aws/sagemaker-python-sdk>

*Usage:*

```
Chainer$new(
    entry_point,
    use_mpi = NULL,
    num_processes = NULL,
    process_slots_per_host = NULL,
    additional_mpi_options = NULL,
    source_dir = NULL,
```

```

hyperparameters = NULL,
framework_version = NULL,
py_version = NULL,
image_uri = NULL,
...
)

```

*Arguments:*

**entry\_point** (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to training. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

**use\_mpi** (bool): If true, entry point is run as an MPI script. By default, the Chainer Framework runs the entry point with ‘mpirun’ if more than one instance is used.

**num\_processes** (int): Total number of processes to run the entry point with. By default, the Chainer Framework runs one process per GPU (on GPU instances), or one process per host (on CPU instances).

**process\_slots\_per\_host** (int): The number of processes that can run on each instance. By default, this is set to the number of GPUs on the instance (on GPU instances), or one (on CPU instances).

**additional\_mpi\_options** (str): String of options to the ‘mpirun’ command used to run the entry point. For example, ‘-X NCCL\_DEBUG=WARN’ will pass that option string to the mpirun command.

**source\_dir** (str): Path (absolute or relative) to a directory with any other training source code dependencies aside from the entry point file (default: None). Structure within this directory are preserved when training on Amazon SageMaker.

**hyperparameters** (dict): Hyperparameters that will be used for training (default: None). The hyperparameters are made accessible as a dict[str, str] to the training code on SageMaker. For convenience, this accepts other types for keys and values, but “str()” will be called to convert them before training.

**framework\_version** (str): Chainer version you want to use for executing your model training code. Defaults to “None”. Required unless “image\_uri” is provided. List of supported versions: <https://github.com/aws/sagemaker-python-sdk#chainer-sagemaker-estimators>.

**py\_version** (str): Python version you want to use for executing your model training code. Defaults to “None”. Required unless “image\_uri” is provided.

**image\_uri** (str): If specified, the estimator will use this image for training and hosting, instead of selecting the appropriate SageMaker official image based on framework\_version and py\_version. It can be an ECR url or dockerhub image and tag. Examples \* “123412341234.dkr.ecr.us-west-2.amazonaws.com/my-custom-image:1.0” \* “custom-image:latest” If “framework\_version” or “py\_version” are “None”, then “image\_uri” is required. If also “None”, then a “ValueError” will be raised.

... : Additional kwargs passed to the :class:~sagemaker.estimator.Framework’ constructor.

**Method** `hyperparameters()`: Return hyperparameters used by your custom Chainer code during training.

*Usage:*

```
Chainer$hyperparameters()
```

**Method** `create_model()`: Create a SageMaker “ChainerModel” object that can be deployed to an “Endpoint”.

*Usage:*

```
Chainer$create_model(
  model_server_workers = NULL,
  role = NULL,
  vpc_config_override = "VPC_CONFIG_DEFAULT",
  entry_point = NULL,
  source_dir = NULL,
  dependencies = NULL,
  ...
)
```

*Arguments:*

`model_server_workers` (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

`role` (str): The “ExecutionRoleArn” IAM Role ARN for the “Model”, which is also used during transform jobs. If not specified, the role from the Estimator will be used.

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* ‘Subnets’ (list[str]): List of subnet ids. \* ‘SecurityGroupIds’ (list[str]): List of security group ids.

`entry_point` (str): Path (absolute or relative) to the local Python source file which should be executed as the entry point to training. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”. If not specified, the training entry point is used.

`source_dir` (str): Path (absolute or relative) to a directory with any other serving source code dependencies aside from the entry point file. If not specified, the model source directory from training is used.

`dependencies` (list[str]): A list of paths to directories (absolute or relative) with any additional libraries that will be exported to the container. If not specified, the dependencies from training are used. This is not supported with "local code" in Local Mode.

... : Additional kwargs passed to the ChainerModel constructor.

*Returns:* `sagemaker.chainer.model.ChainerModel`: A SageMaker “ChainerModel” object. See `:func:~sagemaker.chainer.model.ChainerModel` for full details.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|              |                           |
|--------------|---------------------------|
| ChainerModel | <i>ChainerModel Class</i> |
|--------------|---------------------------|

---

## Description

An Chainer SageMaker “Model” that can be deployed to a SageMaker “Endpoint”.

## Super classes

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> sagemaker.mlcore::FrameworkModel
-> ChainerModel
```

## Methods

### Public methods:

- `ChainerModel$new()`
- `ChainerModel$prepare_container_def()`
- `ChainerModel$serving_image_uri()`
- `ChainerModel$clone()`

**Method** `new()`: Initialize an ChainerModel.

*Usage:*

```
ChainerModel$new(
  model_data,
  role,
  entry_point,
  image_uri = NULL,
  framework_version = NULL,
  py_version = NULL,
  predictor_cls = ChainerPredictor,
  model_server_workers = NULL,
  ...
)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`entry_point` (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to model hosting. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

`image_uri` (str): A Docker image URI (default: None). If not specified, a default image for Chainer will be used. If “framework\_version” or “py\_version” are “None”, then “image\_uri” is required. If also “None”, then a “ValueError” will be raised.

`framework_version` (str): Chainer version you want to use for executing your model training code. Defaults to "None". Required unless "image\_uri" is provided.

`py_version` (str): Python version you want to use for executing your model training code. Defaults to "None". Required unless "image\_uri" is provided.

`predictor_cls` (callable[str, sagemaker.session.Session]): A function to call to create a predictor with an endpoint name and SageMaker "Session". If specified, "deploy()" returns the result of invoking this function on the created endpoint name.

`model_server_workers` (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

... : Keyword arguments passed to the :class:`~sagemaker.model.FrameworkModel` initializer.

**Method** `prepare_container_def()`: Return a container definition with framework configuration set in model environment variables.

*Usage:*

```
ChainerModel.prepare_container_def(
    instance_type = NULL,
    accelerator_type = NULL
)
```

*Arguments:*

`instance_type` (str): The EC2 instance type to deploy this Model to. For example, 'ml.p2.xlarge'.

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the model. For example, 'ml.eia1.medium'.

*Returns:* dict[str, str]: A container definition object usable with the CreateModel API.

**Method** `serving_image_uri()`: Create a URI for the serving image.

*Usage:*

```
ChainerModel.serving_image_uri(
    region_name,
    instance_type,
    accelerator_type = NULL
)
```

*Arguments:*

`region_name` (str): AWS region where the image is uploaded.

`instance_type` (str): SageMaker instance type. Used to determine device type (cpu/gpu/family-specific optimized).

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the model. For example, 'ml.eia1.medium'.

*Returns:* str: The appropriate image URI based on the given parameters.

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

*Usage:*

```
ChainerModel.clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

|                  |                                                             |
|------------------|-------------------------------------------------------------|
| ChainerPredictor | <i>A Predictor for inference against Chainer Endpoints.</i> |
|------------------|-------------------------------------------------------------|

---

## Description

This is able to serialize Python lists, dictionaries, and numpy arrays to multidimensional tensors for Chainer inference.

## Super classes

`sagemaker.mlcore::PredictorBase` -> `sagemaker.mlcore::Predictor` -> ChainerPredictor

## Methods

### Public methods:

- `ChainerPredictor$new()`
- `ChainerPredictor$clone()`

**Method** `new()`: Initialize an “ChainerPredictor”.

*Usage:*

```
ChainerPredictor$new(
  endpoint_name,
  sagemaker_session = NULL,
  serializer = NumpySerializer$new(),
  deserializer = NumpyDeserializer$new()
)
```

*Arguments:*

`endpoint_name` (str): The name of the endpoint to perform inference on.

`sagemaker_session` (`sagemaker.session.Session`): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`serializer` (`sagemaker.serializers.BaseSerializer`): Optional. Default serializes input data to .npy format. Handles lists and numpy arrays.

`deserializer` (`sagemaker.deserializers.BaseDeserializer`): Optional. Default parses the response from .npy format to numpy array.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

FactorizationMachines *A supervised learning algorithm used in classification and regression.*

---

### Description

Factorization Machines combine the advantages of Support Vector Machines with factorization models. It is an extension of a linear model that is designed to capture interactions between features within high dimensional sparse datasets economically.

### Super classes

```
sagemaker.mlcore::EstimatorBase -> sagemaker.mlcore::AmazonAlgorithmEstimatorBase
-> FactorizationMachines
```

### Public fields

```
repo_name  sagemaker repo name for framework
repo_version version of framework
.module    mimic python module
```

### Active bindings

```
num_factors  Dimensionality of factorization.
predictor_type Type of predictor 'binary_classifier' or 'regressor'.
epochs       Number of training epochs to run.
clip_gradient Clip the gradient by projecting onto the box [-clip_gradient, +clip_gradient]
eps          Small value to avoid division by 0.
rescale_grad If set, multiplies the gradient with rescale_grad before updating
bias_lr      Non-negative learning rate for the bias term.
linear_lr    Non-negative learning rate for linear terms.
factors_lr   Non-negative learning rate for factorization terms.
bias_wd      Non-negative weight decay for the bias term.
linear_wd    Non-negative weight decay for linear terms.
factors_wd   Non-negative weight decay for factorization terms.
bias_init_method Initialization method for the bias term: 'normal', 'uniform' or 'constant'.
bias_init_scale Non-negative range for initialization of the bias term that takes effect when
               bias_init_method parameter is 'uniform'
bias_init_sigma Non-negative standard deviation for initialization of the bias term that takes
               effect when bias_init_method parameter is 'normal'.
bias_init_value Initial value of the bias term that takes effect when bias_init_method parameter
               is 'constant'.
linear_init_method Initialization method for linear term: 'normal', 'uniform' or 'constant'.
```

`linear_init_scale` on-negative range for initialization of linear terms that takes effect when `linear_init_method` parameter is 'uniform'.

`linear_init_sigma` Non-negative standard deviation for initialization of linear terms that takes effect when `linear_init_method` parameter is 'normal'.

`linear_init_value` Initial value of linear terms that takes effect when `linear_init_method` parameter is 'constant'.

`factors_init_method` Initialization method for factorization term: 'normal', 'uniform' or 'constant'.

`factors_init_scale` Non-negative range for initialization of factorization terms that takes effect when `factors_init_method` parameter is 'uniform'.

`factors_init_sigma` Non-negative standard deviation for initialization of factorization terms that takes effect when `factors_init_method` parameter is 'normal'.

`factors_init_value` Initial value of factorization terms that takes effect when `factors_init_method` parameter is constant'.

## Methods

### Public methods:

- `FactorizationMachines$new()`
- `FactorizationMachines$create_model()`
- `FactorizationMachines$clone()`

**Method** `new()`: Factorization Machines is :class:'Estimator' for general-purpose supervised learning. Amazon SageMaker Factorization Machines is a general-purpose supervised learning algorithm that you can use for both classification and regression tasks. It is an extension of a linear model that is designed to parsimoniously capture interactions between features within high dimensional sparse datasets. This Estimator may be fit via calls to :meth:'~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.fit'. It requires Amazon :class:'~sagemaker.amazon.record\_pb2.Record' protobuf serialized data to be stored in S3. There is an utility :meth:'~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.record\_set' that can be used to upload data to S3 and creates :class:'~sagemaker.amazon.amazon\_estimator.RecordSet' to be passed to the 'fit' call. To learn more about the Amazon protobuf Record class and how to prepare bulk data in this format, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/training.html> After this Estimator is fit, model data is stored in S3. The model may be deployed to an Amazon SageMaker Endpoint by invoking :meth:'~sagemaker.amazon.estimator.EstimatorBase.deploy'. As well as deploying an Endpoint, `deploy` returns a :class:'~sagemaker.amazon.pca.FactorizationMachinesPredictor' object that can be used for inference calls using the trained model hosted in the SageMaker Endpoint. FactorizationMachines Estimators can be configured by setting hyperparameters. The available hyperparameters for FactorizationMachines are documented below. For further information on the AWS FactorizationMachines algorithm, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/fact-machines.html>

### Usage:

```
FactorizationMachines$new(
  role,
  instance_count,
  instance_type,
  num_factors,
```

```

    predictor_type,
    epochs = NULL,
    clip_gradient = NULL,
    eps = NULL,
    rescale_grad = NULL,
    bias_lr = NULL,
    linear_lr = NULL,
    factors_lr = NULL,
    bias_wd = NULL,
    linear_wd = NULL,
    factors_wd = NULL,
    bias_init_method = NULL,
    bias_init_scale = NULL,
    bias_init_sigma = NULL,
    bias_init_value = NULL,
    linear_init_method = NULL,
    linear_init_scale = NULL,
    linear_init_sigma = NULL,
    linear_init_value = NULL,
    factors_init_method = NULL,
    factors_init_scale = NULL,
    factors_init_sigma = NULL,
    factors_init_value = NULL,
    ...
)

```

*Arguments:*

**role (str):** An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if accessing AWS resource.

**instance\_count (int):** Number of Amazon EC2 instances to use for training.

**instance\_type (str):** Type of EC2 instance to use for training, for example, 'ml.c4.xlarge'.

**num\_factors (int):** Dimensionality of factorization.

**predictor\_type (str):** Type of predictor 'binary\_classifier' or 'regressor'.

**epochs (int):** Number of training epochs to run.

**clip\_gradient (float):** Optimizer parameter. Clip the gradient by projecting onto the box [-clip\_gradient, +clip\_gradient]

**eps (float):** Optimizer parameter. Small value to avoid division by 0.

**rescale\_grad (float):** Optimizer parameter. If set, multiplies the gradient with rescale\_grad before updating. Often choose to be 1.0/batch\_size.

**bias\_lr (float):** Non-negative learning rate for the bias term.

**linear\_lr (float):** Non-negative learning rate for linear terms.

**factors\_lr (float):** Non-negative learning rate for factorization terms.

**bias\_wd (float):** Non-negative weight decay for the bias term.

**linear\_wd (float):** Non-negative weight decay for linear terms.

**factors\_wd (float):** Non-negative weight decay for factorization terms.

`bias_init_method` (string): Initialization method for the bias term: 'normal', 'uniform' or 'constant'.

`bias_init_scale` (float): Non-negative range for initialization of the bias term that takes effect when `bias_init_method` parameter is 'uniform'.

`bias_init_sigma` (float): Non-negative standard deviation for initialization of the bias term that takes effect when `bias_init_method` parameter is 'normal'.

`bias_init_value` (float): Initial value of the bias term that takes effect when `bias_init_method` parameter is 'constant'.

`linear_init_method` (string): Initialization method for linear term: 'normal', 'uniform' or 'constant'.

`linear_init_scale` (float): Non-negative range for initialization of linear terms that takes effect when `linear_init_method` parameter is 'uniform'.

`linear_init_sigma` (float): Non-negative standard deviation for initialization of linear terms that takes effect when `linear_init_method` parameter is 'normal'.

`linear_init_value` (float): Initial value of linear terms that takes effect when `linear_init_method` parameter is 'constant'.

`factors_init_method` (string): Initialization method for factorization term: 'normal', 'uniform' or 'constant'.

`factors_init_scale` (float): Non-negative range for initialization of factorization terms that takes effect when `factors_init_method` parameter is 'uniform'.

`factors_init_sigma` (float): Non-negative standard deviation for initialization of factorization terms that takes effect when `factors_init_method` parameter is 'normal'.

`factors_init_value` (float): Initial value of factorization terms that takes effect when `factors_init_method` parameter is 'constant'.

... : base class keyword argument values. You can find additional parameters for initializing this class at :class:'~sagemaker.estimator.amazon\_estimator.AmazonAlgorithmEstimatorBase' and :class:'~sagemaker.estimator.EstimatorBase'.

**Method** `create_model()`: Return a :class:'~sagemaker.amazon.FactorizationMachinesModel' referencing the latest s3 model data produced by this Estimator.

*Usage:*

```
FactorizationMachines$create_model(
  vpc_config_override = "VPC_CONFIG_DEFAULT",
  ...
)
```

*Arguments:*

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* 'Subnets' (list[str]): List of subnet ids. \* 'SecurityGroupIds' (list[str]): List of security group ids.

... : Additional kwargs passed to the FactorizationMachinesModel constructor.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

FactorizationMachinesModel

*Amazon FactorizationMachinesModel Class*


---

## Description

Reference S3 model data created by FactorizationMachines estimator. Calling :meth:~sagemaker.model.Model.deploy creates an Endpoint and returns :class:FactorizationMachinesPredictor.

## Super classes

sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> FactorizationMachinesModel

## Methods

### Public methods:

- [FactorizationMachinesModel\\$new\(\)](#)
- [FactorizationMachinesModel\\$clone\(\)](#)

**Method** new(): Initialize FactorizationMachinesModel class

*Usage:*

```
FactorizationMachinesModel$new(model_data, role, sagemaker_session = NULL, ...)
```

*Arguments:*

model\_data (str): The S3 location of a SageMaker model data “.tar.gz“ file.

role (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

sagemaker\_session (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

... : Keyword arguments passed to the “FrameworkModel“ initializer.

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

---

FactorizationMachinesPredictor

*Performs binary-classification or regression prediction from input vectors.*

---

## Description

The implementation of :meth:`~sagemaker.predictor.Predictor.predict` in this ‘Predictor’ requires a numpy “ndarray” as input. The array should contain the same number of columns as the feature-dimension of the data used to fit the model this Predictor performs inference on. :meth:`~predict()` returns a list of :class:`~sagemaker.amazon.record\_pb2.Record` objects, one for each row in the input “ndarray”. The prediction is stored in the “score” key of the “Record.label” field. Please refer to the formats details described: <https://docs.aws.amazon.com/sagemaker/latest/dg/fm-in-formats.html>

## Super classes

`sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> FactorizationMachinesPredictor`

## Methods

### Public methods:

- `FactorizationMachinesPredictor$new()`
- `FactorizationMachinesPredictor$clone()`

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

*Usage:*

```
FactorizationMachinesPredictor$new(endpoint_name, sagemaker_session = NULL)
```

*Arguments:*

`endpoint_name` (str): Name of the Amazon SageMaker endpoint to which requests are sent.  
`sagemaker_session` (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: NULL). If not specified, one is created using the default AWS configuration chain.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|             |                                    |
|-------------|------------------------------------|
| HuggingFace | <i>HuggingFace estimator class</i> |
|-------------|------------------------------------|

---

**Description**

Handle training of custom HuggingFace code.

**Super classes**

`sagemaker.mlcore::EstimatorBase` -> `sagemaker.mlcore::Framework` -> HuggingFace

**Public fields**

`.module` mimic python module

**Methods****Public methods:**

- `HuggingFace$new()`
- `HuggingFace$hyperparameters()`
- `HuggingFace$create_model()`
- `HuggingFace$clone()`

**Method** `new()`: This “Estimator” executes a HuggingFace script in a managed execution environment. The managed HuggingFace environment is an Amazon-built Docker container that executes functions defined in the supplied “entry\_point” Python script within a SageMaker Training Job. Training is started by calling `:meth:~sagemaker.amazon.estimator.Framework.fit` on this Estimator.

*Usage:*

```
HuggingFace$new(
  py_version,
  entry_point,
  transformers_version = NULL,
  tensorflow_version = NULL,
  pytorch_version = NULL,
  source_dir = NULL,
  hyperparameters = NULL,
  image_uri = NULL,
  distribution = NULL,
  compiler_config = NULL,
  ...
)
```

*Arguments:*

`py_version` (str): Python version you want to use for executing your model training code.

Defaults to “None“. Required unless “image\_uri“ is provided. List of supported versions:

<https://github.com/aws/sagemaker-python-sdk#huggingface-sagemaker-estimators>

`entry_point` (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to training. If `"source_dir"` is specified, then `"entry_point"` must point to a file located at the root of `"source_dir"`.

`transformers_version` (str): Transformers version you want to use for executing your model training code. Defaults to `"None"`. Required unless `"image_uri"` is provided. The current supported version is `"4.6.1"`.

`tensorflow_version` (str): TensorFlow version you want to use for executing your model training code. Defaults to `"None"`. Required unless `"pytorch_version"` is provided. List of supported versions: <https://github.com/aws/sagemaker-python-sdk#huggingface-sagemaker-estimators>.

`pytorch_version` (str): PyTorch version you want to use for executing your model training code. Defaults to `"None"`. Required unless `"tensorflow_version"` is provided. List of supported versions: <https://github.com/aws/sagemaker-python-sdk#huggingface-sagemaker-estimators>.

`source_dir` (str): Path (absolute, relative or an S3 URI) to a directory with any other training source code dependencies aside from the entry point file (default: `None`). If `"source_dir"` is an S3 URI, it must point to a tar.gz file. Structure within this directory are preserved when training on Amazon SageMaker.

`hyperparameters` (dict): Hyperparameters that will be used for training (default: `None`). The hyperparameters are made accessible as a dict[str, str] to the training code on SageMaker. For convenience, this accepts other types for keys and values, but `"str()"` will be called to convert them before training.

`image_uri` (str): If specified, the estimator will use this image for training and hosting, instead of selecting the appropriate SageMaker official image based on `framework_version` and `py_version`. It can be an ECR url or dockerhub image and tag. Examples: \* `"123412341234.dkr.ecr.us-west-2.amazonaws.com/my-custom-image:1.0"` \* `"custom-image:latest"` If `"framework_version"` or `"py_version"` are `"None"`, then `"image_uri"` is required. If also `"None"`, then a `"ValueError"` will be raised.

`distribution` (dict): A dictionary with information on how to run distributed training (default: `None`). Currently, the following are supported: distributed training with parameter servers, SageMaker Distributed (SMD) Data and Model Parallelism, and MPI. SMD Model Parallelism can only be used with MPI. To enable parameter server use the following setup:

```
.. code:: python
"parameter_server": "enabled": True
To enable MPI: .. code:: python
"mpi": "enabled": True
To enable SMDistributed Data Parallel or Model Parallel: .. code:: python
"smdistributed": "dataparallel": "enabled": True , "modelparallel": "enabled": True, "parameters":
```

`compiler_config` (:class:'sagemaker.mlcore::TrainingCompilerConfig'): Configures SageMaker Training Compiler to accelerate training.

... : Additional kwargs passed to the :class:'~sagemaker.estimator.Framework' constructor.

**Method** `hyperparameters()`: Return hyperparameters used by your custom PyTorch code during model training.

*Usage:*

```
HuggingFace$hyperparameters()
```

**Method** `create_model()`: Create a model to deploy. The serializer, deserializer, `content_type`, and `accept` arguments are only used to define a default Predictor. They are ignored if an explicit

predictor class is passed in. Other arguments are passed through to the Model class. Creating model with HuggingFace training job is not supported.

*Usage:*

```
HuggingFace$create_model(
  model_server_workers = NULL,
  role = NULL,
  vpc_config_override = "VPC_CONFIG_DEFAULT",
  entry_point = NULL,
  source_dir = NULL,
  dependencies = NULL,
  ...
)
```

*Arguments:*

`model_server_workers` (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

`role` (str): The "ExecutionRoleArn" IAM Role ARN for the "Model", which is also used during transform jobs. If not specified, the role from the Estimator will be used.

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* 'Subnets' (list[str]): List of subnet ids. \* 'SecurityGroupIds' (list[str]): List of security group ids.

`entry_point` (str): Path (absolute or relative) to the local Python source file which should be executed as the entry point to training. If "source\_dir" is specified, then "entry\_point" must point to a file located at the root of "source\_dir". If 'git\_config' is provided, 'entry\_point' should be a relative location to the Python source file in the Git repo.

`source_dir` (str): Path (absolute, relative or an S3 URI) to a directory with any other training source code dependencies aside from the entry point file (default: None). If "source\_dir" is an S3 URI, it must point to a tar.gz file. Structure within this directory are preserved when training on Amazon SageMaker. If 'git\_config' is provided, 'source\_dir' should be a relative location to a directory in the Git repo.

`dependencies` (list[str]): A list of paths to directories (absolute or relative) with any additional libraries that will be exported to the container (default: []). The library folders will be copied to SageMaker in the same folder where the entrypoint is copied. If 'git\_config' is provided, 'dependencies' should be a list of relative locations to directories with any additional libraries needed in the Git repo.

... : Additional parameters passed to :class:`~sagemaker.model.Model` .. tip:: You can find additional parameters for using this method at :class:`~sagemaker.model.Model`.

*Returns:* (sagemaker.model.Model) a Model ready for deployment.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                  |                               |
|------------------|-------------------------------|
| HuggingFaceModel | <i>HuggingFaceModel Class</i> |
|------------------|-------------------------------|

---

## Description

A Hugging Face SageMaker “Model” that can be deployed to a SageMaker “Endpoint”.

## Super classes

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> sagemaker.mlcore::FrameworkModel
-> HuggingFaceModel
```

## Methods

### Public methods:

- `HuggingFaceModel$new()`
- `HuggingFaceModel$register()`
- `HuggingFaceModel$prepare_container_def()`
- `HuggingFaceModel$serving_image_uri()`
- `HuggingFaceModel$clone()`

**Method** `new()`: Initialize a `HuggingFaceModel`.

#### Usage:

```
HuggingFaceModel$new(
  role,
  model_data = NULL,
  entry_point = NULL,
  transformers_version = NULL,
  tensorflow_version = NULL,
  pytorch_version = NULL,
  py_version = NULL,
  image_uri = NULL,
  predictor_cls = HuggingFacePredictor,
  model_server_workers = NULL,
  ...
)
```

#### Arguments:

`role` (str): An AWS IAM role specified with either the name or full ARN. The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`model_data` (str): The Amazon S3 location of a SageMaker model data “.tar.gz” file.

`entry_point` (str): The absolute or relative path to the Python source file that should be executed as the entry point to model hosting. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”. Defaults to None.

`transformers_version` (str): Transformers version you want to use for executing your model training code. Defaults to None. Required unless “image\_uri” is provided.

`tensorflow_version` (str): TensorFlow version you want to use for executing your inference code. Defaults to “None”. Required unless “pytorch\_version” is provided. List of supported versions: <https://github.com/aws/sagemaker-python-sdk#huggingface-sagemaker-estimators>.

`pytorch_version` (str): PyTorch version you want to use for executing your inference code. Defaults to “None”. Required unless “tensorflow\_version” is provided. List of supported versions: <https://github.com/aws/sagemaker-python-sdk#huggingface-sagemaker-estimators>.

`py_version` (str): Python version you want to use for executing your model training code. Defaults to “None”. Required unless “image\_uri” is provided.

`image_uri` (str): A Docker image URI. Defaults to None. If not specified, a default image for PyTorch will be used. If “framework\_version” or “py\_version” are “None”, then “image\_uri” is required. If also “None”, then a “ValueError” will be raised.

`predictor_cls` (callable[str, sagemaker.session.Session]): A function to call to create a predictor with an endpoint name and SageMaker “Session”. If specified, “deploy()” returns the result of invoking this function on the created endpoint name.

`model_server_workers` (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

... : Keyword arguments passed to the superclass :class:`~sagemaker.model.FrameworkModel` and, subsequently, its superclass :class:`~sagemaker.model.Model`.,

**Method** `register()`: Creates a model package for creating SageMaker models or listing on Marketplace.

*Usage:*

```
HuggingFaceModel.register(
    content_types,
    response_types,
    inference_instances,
    transform_instances,
    model_package_name = NULL,
    model_package_group_name = NULL,
    image_uri = NULL,
    model_metrics = NULL,
    metadata_properties = NULL,
    marketplace_cert = FALSE,
    approval_status = NULL,
    description = NULL,
    drift_check_baselines = NULL
)
```

*Arguments:*

`content_types` (list): The supported MIME types for the input data.

`response_types` (list): The supported MIME types for the output data.

`inference_instances` (list): A list of the instance types that are used to generate inferences in real-time.

`transform_instances` (list): A list of the instance types on which a transformation job can be run or on which an endpoint can be deployed.

`model_package_name` (str): Model Package name, exclusive to 'model\_package\_group\_name', using 'model\_package\_name' makes the Model Package un-versioned. Defaults to "None".

`model_package_group_name` (str): Model Package Group name, exclusive to 'model\_package\_name', using 'model\_package\_group\_name' makes the Model Package versioned. Defaults to "None".

`image_uri` (str): Inference image URI for the container. Model class' self.image will be used if it is None. Defaults to "None".

`model_metrics` (ModelMetrics): ModelMetrics object. Defaults to "None".

`metadata_properties` (MetadataProperties): MetadataProperties object. Defaults to "None".

`marketplace_cert` (bool): A boolean value indicating if the Model Package is certified for AWS Marketplace. Defaults to "False".

`approval_status` (str): Model Approval Status, values can be "Approved", "Rejected", or "PendingManualApproval". Defaults to "PendingManualApproval".

`description` (str): Model Package description. Defaults to "None".

`drift_check_baselines` (DriftCheckBaselines): DriftCheckBaselines object (default: None)

*Returns:* A 'sagemaker.model.ModelPackage' instance.

**Method** `prepare_container_def()`: A container definition with framework configuration set in model environment variables.

*Usage:*

```
HuggingFaceModel$prepare_container_def(
  instance_type = NULL,
  accelerator_type = NULL
)
```

*Arguments:*

`instance_type` (str): The EC2 instance type to deploy this Model to. For example, 'ml.p2.xlarge'.

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the model.

*Returns:* dict[str, str]: A container definition object usable with the CreateModel API.

**Method** `serving_image_uri()`: Create a URI for the serving image.

*Usage:*

```
HuggingFaceModel$serving_image_uri(
  region_name,
  instance_type,
  accelerator_type = NULL
)
```

*Arguments:*

`region_name` (str): AWS region where the image is uploaded.

`instance_type` (str): SageMaker instance type. Used to determine device type (cpu/gpu/family-specific optimized).

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the model.

*Returns:* str: The appropriate image URI based on the given parameters.

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

---

HuggingFacePredictor    *A Predictor for inference against Hugging Face Endpoints.*

---

## Description

This is able to serialize Python lists, dictionaries, and numpy arrays to multidimensional tensors for Hugging Face inference.

## Super classes

```
sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> HuggingFacePredictor
```

## Methods

### Public methods:

- [HuggingFacePredictor\\$new\(\)](#)
- [HuggingFacePredictor\\$clone\(\)](#)

**Method** new(): Initialize an “HuggingFacePredictor”.

*Usage:*

```
HuggingFacePredictor$new(
  endpoint_name,
  sagemaker_session = NULL,
  serializer = JsonSerializer$new(),
  deserializer = JSONDeserializer$new()
)
```

*Arguments:*

endpoint\_name (str): The name of the endpoint to perform inference on.

sagemaker\_session (sagemaker.session.Session): Session object that manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

serializer (sagemaker.serializers.BaseSerializer): Optional. Default serializes input data to .npy format. Handles lists and numpy arrays.

deserializer (sagemaker.deserializers.BaseDeserializer): Optional. Default parses the response from .npy format to numpy array.

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

---

HuggingFaceProcessor    *HuggingFaceProcessor class*

---

## Description

Handles Amazon SageMaker processing tasks for jobs using HuggingFace containers.

## Super classes

```
sagemaker.common::Processor -> sagemaker.common::ScriptProcessor -> sagemaker.common::FrameworkProcess
-> HuggingFaceProcessor
```

## Public fields

estimator\_cls Estimator object

## Methods

### Public methods:

- [HuggingFaceProcessor\\$new\(\)](#)
- [HuggingFaceProcessor\\$clone\(\)](#)

**Method new():** This processor executes a Python script in a HuggingFace execution environment. Unless “image\_uri” is specified, the environment is an Amazon-built Docker container that executes functions defined in the supplied “code” Python script. The arguments have the same meaning as in “FrameworkProcessor”, with the following exceptions.

*Usage:*

```
HuggingFaceProcessor$new(
  role,
  instance_count,
  instance_type,
  transformers_version = NULL,
  tensorflow_version = NULL,
  pytorch_version = NULL,
  py_version = "py36",
  image_uri = NULL,
  command = NULL,
  volume_size_in_gb = 30,
  volume_kms_key = NULL,
  output_kms_key = NULL,
  code_location = NULL,
  max_runtime_in_seconds = NULL,
```

```

    base_job_name = NULL,
    sagemaker_session = NULL,
    env = NULL,
    tags = NULL,
    network_config = NULL
)

```

*Arguments:*

`role` (str): An AWS IAM role name or ARN. Amazon SageMaker Processing uses this role to access AWS resources, such as data stored in Amazon S3.

`instance_count` (int): The number of instances to run a processing job with.

`instance_type` (str): The type of EC2 instance to use for processing, for example, 'ml.c4.xlarge'.

`transformers_version` (str): Transformers version you want to use for executing your model training code. Defaults to "None". Required unless "image\_uri" is provided. The current supported version is "4.4.2".

`tensorflow_version` (str): TensorFlow version you want to use for executing your model training code. Defaults to "None". Required unless "pytorch\_version" is provided. The current supported version is "1.6.0".

`pytorch_version` (str): PyTorch version you want to use for executing your model training code. Defaults to "None". Required unless "tensorflow\_version" is provided. The current supported version is "2.4.1".

`py_version` (str): Python version you want to use for executing your model training code. Defaults to "None". Required unless "image\_uri" is provided. If using PyTorch, the current supported version is "py36". If using TensorFlow, the current supported version is "py37".

`image_uri` (str): The URI of the Docker image to use for the processing jobs (default: None).

`command` ([str]): The command to run, along with any command-line flags to \*precede\* the "code script". Example: ["python3", "-v"]. If not provided, ["python"] will be chosen (default: None).

`volume_size_in_gb` (int): Size in GB of the EBS volume to use for storing data during processing (default: 30).

`volume_kms_key` (str): A KMS key for the processing volume (default: None).

`output_kms_key` (str): The KMS key ID for processing job outputs (default: None).

`code_location` (str): The S3 prefix URI where custom code will be uploaded (default: None). The code file uploaded to S3 is 'code\_location/job-name/source/sourcedir.tar.gz'. If not specified, the default "code location" is 's3://sagemaker-default-bucket'

`max_runtime_in_seconds` (int): Timeout in seconds (default: None). After this amount of time, Amazon SageMaker terminates the job, regardless of its current status. If 'max\_runtime\_in\_seconds' is not specified, the default value is 24 hours.

`base_job_name` (str): Prefix for processing name. If not specified, the processor generates a default job name, based on the processing image name and current timestamp (default: None).

`sagemaker_session` (:class:`~sagemaker.session.Session`): Session object which manages interactions with Amazon SageMaker and any other AWS services needed. If not specified, the processor creates one using the default AWS configuration chain (default: None).

`env` (dict[str, str]): Environment variables to be passed to the processing jobs (default: None).

`tags` (list[dict]): List of tags to be passed to the processing job (default: None). For more, see [https://docs.aws.amazon.com/sagemaker/latest/dg/API\\_Tag.html](https://docs.aws.amazon.com/sagemaker/latest/dg/API_Tag.html).

`network_config` (:class:~sagemaker.network.NetworkConfig): A :class:~sagemaker.network.NetworkConfig object that configures network isolation, encryption of inter-container traffic, security group IDs, and subnets (default: None).

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

*Usage:*

`HuggingFaceProcessor$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

---

|            |                                                                                              |
|------------|----------------------------------------------------------------------------------------------|
| IPInsights | <i>An unsupervised learning algorithm that learns the usage patterns for IPv4 addresses.</i> |
|------------|----------------------------------------------------------------------------------------------|

---

## Description

It is designed to capture associations between IPv4 addresses and various entities, such as user IDs or account numbers.

## Super classes

`sagemaker.mlcore::EstimatorBase` -> `sagemaker.mlcore::AmazonAlgorithmEstimatorBase`  
-> IPInsights

## Public fields

`repo_name` sagemaker repo name for framework

`repo_version` version of framework

`MINI_BATCH_SIZE` The size of each mini-batch to use when training. If None, a default value will be used.

`.module` mimic python module

## Active bindings

`num_entity_vectors` The number of embeddings to train for entities accessing online resources

`vector_dim` The size of the embedding vectors for both entity and IP addresses

`batch_metrics_publish_interval` The period at which to publish metrics

`epochs` Maximum number of passes over the training data.

`learning_rate` Learning rate for the optimizer.

`num_ip_encoder_layers` The number of fully-connected layers to encode IP address embedding.

`random_negative_sampling_rate` The ratio of random negative samples to draw during training.

`shuffled_negative_sampling_rate` The ratio of shuffled negative samples to draw during training.

`weight_decay` Weight decay coefficient. Adds L2 regularization

## Methods

### Public methods:

- `IPInsights$new()`
- `IPInsights$create_model()`
- `IPInsights$.prepare_for_training()`
- `IPInsights$clone()`

**Method new():** This estimator is for IP Insights, an unsupervised algorithm that learns usage patterns of IP addresses. This Estimator may be fit via calls to :meth: '~sagemaker.amazon.amazon\_estimator.AmazonAlgorithm'. It requires CSV data to be stored in S3. After this Estimator is fit, model data is stored in S3. The model may be deployed to an Amazon SageMaker Endpoint by invoking :meth: '~sagemaker.amazon.estimator.EstimatorBase.deploy'. As well as deploying an Endpoint, deploy returns a :class: '~sagemaker.amazon.IPInsightPredictor' object that can be used for inference calls using the trained model hosted in the SageMaker Endpoint. IPInsights Estimators can be configured by setting hyperparameters. The available hyperparameters are documented below. For further information on the AWS IPInsights algorithm, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/ip-insights-hyperparameters.html>

#### Usage:

```
IPInsights$new(
    role,
    instance_count,
    instance_type,
    num_entity_vectors,
    vector_dim,
    batch_metrics_publish_interval = NULL,
    epochs = NULL,
    learning_rate = NULL,
    num_ip_encoder_layers = NULL,
    random_negative_sampling_rate = NULL,
    shuffled_negative_sampling_rate = NULL,
    weight_decay = NULL,
    ...
)
```

#### Arguments:

**role (str):** An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if accessing AWS resource.

**instance\_count (int):** Number of Amazon EC2 instances to use for training.

**instance\_type (str):** Type of EC2 instance to use for training, for example, 'ml.m5.xlarge'.

**num\_entity\_vectors (int):** Required. The number of embeddings to train for entities accessing online resources. We recommend 2x the total number of unique entity IDs.

**vector\_dim (int):** Required. The size of the embedding vectors for both entity and IP addresses.

**batch\_metrics\_publish\_interval (int):** Optional. The period at which to publish metrics (batches).

`epochs` (int): Optional. Maximum number of passes over the training data.

`learning_rate` (float): Optional. Learning rate for the optimizer.

`num_ip_encoder_layers` (int): Optional. The number of fully-connected layers to encode IP address embedding.

`random_negative_sampling_rate` (int): Optional. The ratio of random negative samples to draw during training. Random negative samples are randomly drawn IPv4 addresses.

`shuffled_negative_sampling_rate` (int): Optional. The ratio of shuffled negative samples to draw during training. Shuffled negative samples are IP addresses picked from within a batch.

`weight_decay` (float): Optional. Weight decay coefficient. Adds L2 regularization.

... : base class keyword argument values.

**Method** `create_model()`: Create a model for the latest s3 model produced by this estimator.

*Usage:*

```
IPInsights$.create_model(vpc_config_override = "VPC_CONFIG_DEFAULT", ...)
```

*Arguments:*

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* `'Subnets'` (list[str]): List of subnet ids. \* `'SecurityGroupIds'` (list[str]): List of security group ids.

... : Additional kwargs passed to the IPInsightsModel constructor.

*Returns:* :class:`~sagemaker.amazon.IPInsightsModel`: references the latest s3 model data produced by this estimator.

**Method** `.prepare_for_training()`: Set hyperparameters needed for training. This method will also validate “source\_dir”.

*Usage:*

```
IPInsights$.prepare_for_training(
    records,
    mini_batch_size = NULL,
    job_name = NULL
)
```

*Arguments:*

`records` (RecordSet) – The records to train this Estimator on.

`mini_batch_size` (int or None) – The size of each mini-batch to use when training. If None, a default value will be used.

`job_name` (str): Name of the training job to be created. If not specified, one is generated, using the base name given to the constructor if applicable.

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

*Usage:*

```
IPInsights$.clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

IPInsightsModel

Reference IPInsights s3 model data.

---

## Description

Calling `:meth:`~sagemaker.model.Model.deploy`` creates an Endpoint and returns a Predictor that calculates anomaly scores for data points.

## Super classes

`sagemaker.mlcore::ModelBase` -> `sagemaker.mlcore::Model` -> IPInsightsModel

## Methods

### Public methods:

- `IPInsightsModel$new()`
- `IPInsightsModel$clone()`

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

*Usage:*

```
IPInsightsModel$new(model_data, role, sagemaker_session = NULL, ...)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`sagemaker_session` (`sagemaker.session.Session`): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`...` : Keyword arguments passed to the “FrameworkModel” initializer.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                     |                                                                                              |
|---------------------|----------------------------------------------------------------------------------------------|
| IPInsightsPredictor | <i>Returns dot product of entity and IP address embeddings as a score for compatibility.</i> |
|---------------------|----------------------------------------------------------------------------------------------|

---

## Description

The implementation of :meth:`~sagemaker.predictor.Predictor.predict` in this ‘Predictor’ requires a numpy “ndarray” as input. The array should contain two columns. The first column should contain the entity ID. The second column should contain the IPv4 address in dot notation.

## Super classes

`sagemaker.mlcore::PredictorBase` -> `sagemaker.mlcore::Predictor` -> IPInsightsPredictor

## Methods

### Public methods:

- `IPInsightsPredictor$new()`
- `IPInsightsPredictor$clone()`

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

*Usage:*

```
IPInsightsPredictor$new(endpoint_name, sagemaker_session = NULL)
```

*Arguments:*

`endpoint_name` (str): Name of the Amazon SageMaker endpoint to which requests are sent.  
`sagemaker_session` (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, one is created using the default AWS configuration chain.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|        |                                                                                                 |
|--------|-------------------------------------------------------------------------------------------------|
| KMeans | <i>An unsupervised learning algorithm that attempts to find discrete groupings within data.</i> |
|--------|-------------------------------------------------------------------------------------------------|

---

### Description

As the result of KMeans, members of a group are as similar as possible to one another and as different as possible from members of other groups. You define the attributes that you want the algorithm to use to determine similarity.

### Super classes

```
sagemaker.mlcore::EstimatorBase -> sagemaker.mlcore::AmazonAlgorithmEstimatorBase
-> KMeans
```

### Public fields

```
repo_name  sagemaker repo name for framework
repo_version version of framework
.module    mimic python module
```

### Active bindings

```
k      The number of clusters to produce.
init_method  How to initialize cluster locations.
max_iterations  Maximum iterations for Lloyds EM procedure in the local kmeans used in final-
               ize stage.
tol      Tolerance for change in ssd for early stopping in local kmeans.
num_trials  Local version is run multiple times and the one with the best loss is chosen.
local_init_method  Initialization method for local version.
half_life_time_size  The points can have a decayed weight.
epochs     Number of passes done over the training data.
center_factor  The algorithm will create “num_clusters * extra_center_factor“ as it runs.
eval_metrics  JSON list of metrics types to be used for reporting the score for the model.
```

### Methods

#### Public methods:

- `KMeans$new()`
- `KMeans$create_model()`
- `KMeans$.prepare_for_training()`
- `KMeans$hyperparameters()`
- `KMeans$clone()`

**Method new():** A k-means clustering :class: '~sagemaker.amazon.AmazonAlgorithmEstimatorBase'. Finds k clusters of data in an unlabeled dataset. This Estimator may be fit via calls to :meth: '~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.fit'. The former allows a KMeans model to be fit on a 2-dimensional numpy array. The latter requires Amazon :class: '~sagemaker.amazon.record\_pb2.Record' protobuf serialized data to be stored in S3. To learn more about the Amazon protobuf Record class and how to prepare bulk data in this format, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/cdf-training.html>. After this Estimator is fit, model data is stored in S3. The model may be deployed to an Amazon SageMaker Endpoint by invoking :meth: '~sagemaker.amazon.estimator.EstimatorBase.deploy'. As well as deploying an Endpoint, "deploy" returns a :class: '~sagemaker.amazon.kmeans.KMeansPredictor' object that can be used to k-means cluster assignments, using the trained k-means model hosted in the SageMaker Endpoint. KMeans Estimators can be configured by setting hyperparameters. The available hyperparameters for KMeans are documented below. For further information on the AWS KMeans algorithm, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/kmeans.html>.

*Usage:*

```
KMeans$new(
  role,
  instance_count,
  instance_type,
  k,
  init_method = NULL,
  max_iterations = NULL,
  tol = NULL,
  num_trials = NULL,
  local_init_method = NULL,
  half_life_time_size = NULL,
  epochs = NULL,
  center_factor = NULL,
  eval_metrics = NULL,
  ...
)
```

*Arguments:*

**role** (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if accessing AWS resource.

**instance\_count** (int): Number of Amazon EC2 instances to use for training.

**instance\_type** (str): Type of EC2 instance to use for training, for example, 'ml.c4.xlarge'.

**k** (int): The number of clusters to produce.

**init\_method** (str): How to initialize cluster locations. One of 'random' or 'kmeans++'.

**max\_iterations** (int): Maximum iterations for Lloyds EM procedure in the local kmeans used in finalize stage.

**tol** (float): Tolerance for change in ssd for early stopping in local kmeans.

**num\_trials** (int): Local version is run multiple times and the one with the best loss is chosen. This determines how many times.

`local_init_method` (str): Initialization method for local version. One of 'random', 'kmeans++'

`half_life_time_size` (int): The points can have a decayed weight. When a point is observed its weight, with regard to the computation of the cluster mean is 1. This weight will decay exponentially as we observe more points. The exponent coefficient is chosen such that after observing "half\_life\_time\_size" points after the mentioned point, its weight will become 1/2. If set to 0, there will be no decay.

`epochs` (int): Number of passes done over the training data.

`center_factor` (int): The algorithm will create "num\_clusters \* extra\_center\_factor" as it runs and reduce the number of centers to "k" when finalizing

`eval_metrics` (list): JSON list of metrics types to be used for reporting the score for the model. Allowed values are "msd" Means Square Error, "ssd": Sum of square distance. If test data is provided, the score shall be reported in terms of all requested metrics.

... : base class keyword argument values.

**Method** `create_model()`: Return a :class:`~sagemaker.amazon.kmeans.KMeansModel` referencing the latest s3 model data produced by this Estimator.

*Usage:*

```
KMeans$.create_model(vpc_config_override = "VPC_CONFIG_DEFAULT", ...)
```

*Arguments:*

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* 'Subnets' (list[str]): List of subnet ids. \* 'SecurityGroupIds' (list[str]): List of security group ids.

... : Additional kwargs passed to the KMeansModel constructor.

**Method** `.prepare_for_training()`: Set hyperparameters needed for training. This method will also validate "source\_dir".

*Usage:*

```
KMeans$.prepare_for_training(records, mini_batch_size = 5000, job_name = NULL)
```

*Arguments:*

`records` (RecordSet) – The records to train this Estimator on.

`mini_batch_size` (int or None) – The size of each mini-batch to use when training. If None, a default value will be used.

`job_name` (str): Name of the training job to be created. If not specified, one is generated, using the base name given to the constructor if applicable.

**Method** `hyperparameters()`: Return the SageMaker hyperparameters for training this KMeans Estimator

*Usage:*

```
KMeans$.hyperparameters()
```

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

*Usage:*

```
KMeans$.clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

KMeansModel

*Reference KMeans s3 model data.***Description**

Calling :meth:‘~sagemaker.model.Model.deploy‘ creates an Endpoint and return a Predictor to performs k-means cluster assignment.

**Super classes**

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> KMeansModel
```

**Methods****Public methods:**

- `KMeansModel$new()`
- `KMeansModel$clone()`

**Method** `new()`: Initialize KMeansPredictor Class

*Usage:*

```
KMeansModel$new(model_data, role, sagemaker_session = NULL, ...)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz“ file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`sagemaker_session` (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`...` : Keyword arguments passed to the “FrameworkModel“ initializer.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                 |                                                                          |
|-----------------|--------------------------------------------------------------------------|
| KMeansPredictor | <i>Assigns input vectors to their closest cluster in a KMeans model.</i> |
|-----------------|--------------------------------------------------------------------------|

---

## Description

The implementation of :meth:`~sagemaker.predictor.Predictor.predict` in this ‘Predictor’ requires a numpy “ndarray” as input. The array should contain the same number of columns as the feature-dimension of the data used to fit the model this Predictor performs inference on. “predict()” returns a list of :class:`~sagemaker.amazon.record\_pb2.Record` objects, one for each row in the input “ndarray”. The nearest cluster is stored in the “closest\_cluster” key of the “Record.label” field.

## Super classes

```
sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> KMeansPredictor
```

## Methods

### Public methods:

- `KMeansPredictor$new()`
- `KMeansPredictor$clone()`

**Method** `new()`: Initialize KMeansPredictor Class

*Usage:*

```
KMeansPredictor$new(endpoint_name, sagemaker_session = NULL)
```

*Arguments:*

`endpoint_name` (str): Name of the Amazon SageMaker endpoint to which requests are sent.  
`sagemaker_session` (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, one is created using the default AWS configuration chain.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

KNN

*An index-based algorithm. It uses a non-parametric method for classification or regression.*

### Description

For classification problems, the algorithm queries the k points that are closest to the sample point and returns the most frequently used label of their class as the predicted label. For regression problems, the algorithm queries the k closest points to the sample point and returns the average of their feature values as the predicted value.

### Super classes

```
sagemaker.mlcore::EstimatorBase -> sagemaker.mlcore::AmazonAlgorithmEstimatorBase
-> KNN
```

### Public fields

```
repo_name  sagemaker repo name for framework
repo_version version of framework
.module    mimic python module
```

### Active bindings

```
k      Number of nearest neighbors.
sample_size  Number of data points to be sampled from the training data set
predictor_type  Type of inference to use on the data's labels
dimension_reduction_target  Target dimension to reduce to
dimension_reduction_type  Type of dimension reduction technique to use
index_metric  Distance metric to measure between points when finding nearest neighbors
index_type  Type of index to use. Valid values are "faiss.Flat", "faiss.IVFFlat", "faiss.IVFPQ".
faiss_index_ivf_nlists  Number of centroids to construct in the index
faiss_index_pq_m  Number of vector sub-components to construct in the index
```

### Methods

#### Public methods:

- `KNN$new()`
- `KNN$create_model()`
- `KNN$.prepare_for_training()`
- `KNN$clone()`

**Method** `new()`: k-nearest neighbors (KNN) is :class:'Estimator' used for classification and regression. This Estimator may be fit via calls to :meth:'~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimator.fit'. It requires Amazon :class:'~sagemaker.amazon.record\_pb2.Record' protobuf serialized data to be stored in S3. There is an utility :meth:'~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.record\_set' that can be used to upload data to S3 and creates :class:'~sagemaker.amazon.amazon\_estimator.RecordSet' to be passed to the 'fit' call. To learn more about the Amazon protobuf Record class and how to prepare bulk data in this format, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/training.html> After this Estimator is fit, model data is stored in S3. The model may be deployed to an Amazon SageMaker Endpoint by invoking :meth:'~sagemaker.amazon.estimator.EstimatorBase.deploy'. As well as deploying an Endpoint, `deploy` returns a :class:'~sagemaker.amazon.knn.KNNPredictor' object that can be used for inference calls using the trained model hosted in the SageMaker Endpoint. KNN Estimators can be configured by setting hyperparameters. The available hyperparameters for KNN are documented below. For further information on the AWS KNN algorithm, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/knn.html>

*Usage:*

```
KNN$new(
  role,
  instance_count,
  instance_type,
  k,
  sample_size,
  predictor_type,
  dimension_reduction_type = NULL,
  dimension_reduction_target = NULL,
  index_type = NULL,
  index_metric = NULL,
  faiss_index_ivf_nlists = NULL,
  faiss_index_pq_m = NULL,
  ...
)
```

*Arguments:*

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if accessing AWS resource.

`instance_count` (int): Number of Amazon EC2 instances to use for training.

`instance_type` (str): Type of EC2 instance to use for training, for example, 'ml.c4.xlarge'.

`k` (int): Required. Number of nearest neighbors.

`sample_size` (int): Required. Number of data points to be sampled from the training data set.

`predictor_type` (str): Required. Type of inference to use on the data's labels, allowed values are 'classifier' and 'regressor'.

`dimension_reduction_type` (str): Optional. Type of dimension reduction technique to use. Valid values: "sign", "fjlt"

`dimension_reduction_target` (int): Optional. Target dimension to reduce to. Required when `dimension_reduction_type` is specified.

`index_type` (str): Optional. Type of index to use. Valid values are "faiss.Flat", "faiss.IVFFlat", "faiss.IVFPQ".

`index_metric` (str): Optional. Distance metric to measure between points when finding nearest neighbors. Valid values are "COSINE", "INNER\_PRODUCT", "L2"

`faiss_index_ivf_nlists` (str): Optional. Number of centroids to construct in the index if `index_type` is "faiss.IVFFlat" or "faiss.IVFPQ".

`faiss_index_pq_m` (int): Optional. Number of vector sub-components to construct in the index, if `index_type` is "faiss.IVFPQ".

... : base class keyword argument values.

**Method** `create_model()`: Return a :class:`~sagemaker.amazon.KNNModel` referencing the latest s3 model data produced by this Estimator.

*Usage:*

```
KNN$create_model(vpc_config_override = "VPC_CONFIG_DEFAULT", ...)
```

*Arguments:*

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* 'Subnets' (list[str]): List of subnet ids. \* 'SecurityGroupIds' (list[str]): List of security group ids.

... : Additional kwargs passed to the KNNModel constructor.

**Method** `.prepare_for_training()`: Set hyperparameters needed for training. This method will also validate "source\_dir".

*Usage:*

```
KNN$.prepare_for_training(records, mini_batch_size = NULL, job_name = NULL)
```

*Arguments:*

`records` (RecordSet) – The records to train this Estimator on.

`mini_batch_size` (int or None) – The size of each mini-batch to use when training. If None, a default value will be used.

`job_name` (str): Name of the training job to be created. If not specified, one is generated, using the base name given to the constructor if applicable.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

KNNModel

Reference S3 model data created by KNN estimator.

---

## Description

Calling :meth:`~sagemaker.model.Model.deploy` creates an Endpoint and returns :class:`~KNNPredictor`.

**Super classes**

`sagemaker.mlcore::ModelBase` -> `sagemaker.mlcore::Model` -> `KNNModel`

**Methods****Public methods:**

- `KNNModel$new()`
- `KNNModel$clone()`

**Method** `new()`: Initialize KNNModel Class

*Usage:*

`KNNModel$new(model_data, role, sagemaker_session = NULL, ...)`

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz“ file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`sagemaker_session` (`sagemaker.session.Session`): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`...` : Keyword arguments passed to the “FrameworkModel“ initializer.

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

*Usage:*

`KNNModel$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

---

KNNPredictor

*Performs classification or regression prediction from input vectors.*

---

**Description**

The implementation of :meth:‘~sagemaker.predictor.Predictor.predict‘ in this ‘Predictor‘ requires a numpy “ndarray“ as input. The array should contain the same number of columns as the feature-dimension of the data used to fit the model this Predictor performs inference on. :func:‘predict‘ returns a list of :class:‘~sagemaker.amazon.record\_pb2.Record‘ objects, one for each row in the input “ndarray“. The prediction is stored in the “predicted\_label“ key of the “Record.label“ field.

**Super classes**

`sagemaker.mlcore::PredictorBase` -> `sagemaker.mlcore::Predictor` -> `KNNPredictor`

## Methods

### Public methods:

- `KNNPredictor$new()`
- `KNNPredictor$clone()`

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

*Usage:*

```
KNNPredictor$new(endpoint_name, sagemaker_session = NULL)
```

*Arguments:*

`endpoint_name` (str): Name of the Amazon SageMaker endpoint to which requests are sent.  
`sagemaker_session` (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, one is created using the default AWS configuration chain.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

LDA

*An unsupervised learning algorithm attempting to describe data as distinct categories.*

---

## Description

LDA is most commonly used to discover a user-specified number of topics shared by documents within a text corpus. Here each observation is a document, the features are the presence (or occurrence count) of each word, and the categories are the topics.

## Super classes

```
sagemaker.mlcore::EstimatorBase -> sagemaker.mlcore::AmazonAlgorithmEstimatorBase  
-> LDA
```

## Public fields

```
repo_name sagemaker repo name for framework  
repo_version version of framework  
.module mimic python module
```

**Active bindings**

`num_topics` The number of topics for LDA to find within the data

`alpha0` Initial guess for the concentration parameter

`max_restarts` The number of restarts to perform during the Alternating Least Squares

`max_iterations` The maximum number of iterations to perform during the ALS phase of the algorithm.

`tol` Target error tolerance for the ALS phase of the algorithm.

**Methods****Public methods:**

- `LDA$new()`
- `LDA$create_model()`
- `LDA$.prepare_for_training()`
- `LDA$clone()`

**Method** `new()`: Latent Dirichlet Allocation (LDA) is :class:‘Estimator’ used for unsupervised learning. Amazon SageMaker Latent Dirichlet Allocation is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of distinct categories. LDA is most commonly used to discover a user-specified number of topics shared by documents within a text corpus. Here each observation is a document, the features are the presence (or occurrence count) of each word, and the categories are the topics. This Estimator may be fit via calls to :meth:‘~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.fit’. It requires Amazon :class:‘~sagemaker.amazon.record\_pb2.Record’ protobuf serialized data to be stored in S3. There is an utility :meth:‘~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.record\_set’ that can be used to upload data to S3 and creates :class:‘~sagemaker.amazon.amazon\_estimator.RecordSet’ to be passed to the ‘fit’ call. To learn more about the Amazon protobuf Record class and how to prepare bulk data in this format, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/lda.html> After this Estimator is fit, model data is stored in S3. The model may be deployed to an Amazon SageMaker Endpoint by invoking :meth:‘~sagemaker.amazon.estimator.EstimatorBase.deploy’. As well as deploying an Endpoint, `deploy` returns a :class:‘~sagemaker.amazon.lda.LDAPredictor’ object that can be used for inference calls using the trained model hosted in the SageMaker Endpoint. LDA Estimators can be configured by setting hyperparameters. The available hyperparameters for LDA are documented below. For further information on the AWS LDA algorithm, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/lda.html>

*Usage:*

```
LDA$new(
  role,
  instance_type,
  num_topics,
  alpha0 = NULL,
  max_restarts = NULL,
  max_iterations = NULL,
  tol = NULL,
  ...
)
```

*Arguments:*

**role** (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if accessing AWS resource.

**instance\_type** (str): Type of EC2 instance to use for training, for example, 'ml.c4.xlarge'.

**num\_topics** (int): The number of topics for LDA to find within the data.

**alpha0** (float): Optional. Initial guess for the concentration parameter

**max\_restarts** (int): Optional. The number of restarts to perform during the Alternating Least Squares (ALS) spectral decomposition phase of the algorithm.

**max\_iterations** (int): Optional. The maximum number of iterations to perform during the ALS phase of the algorithm.

**tol** (float): Optional. Target error tolerance for the ALS phase of the algorithm.

... : base class keyword argument values.

**Method** `create_model()`: Return a :class:`~sagemaker.amazon.LDAModel` referencing the latest s3 model data produced by this Estimator.

*Usage:*

```
LDA$create_model(vpc_config_override = "VPC_CONFIG_DEFAULT", ...)
```

*Arguments:*

**vpc\_config\_override** (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* 'Subnets' (list[str]): List of subnet ids. \* 'SecurityGroupIds' (list[str]): List of security group ids.

... : Additional kwargs passed to the LDAModel constructor.

**Method** `.prepare_for_training()`: Set hyperparameters needed for training. This method will also validate "source\_dir".

*Usage:*

```
LDA$.prepare_for_training(records, mini_batch_size = NULL, job_name = NULL)
```

*Arguments:*

**records** (RecordSet) – The records to train this Estimator on.

**mini\_batch\_size** (int or None) – The size of each mini-batch to use when training. If None, a default value will be used.

**job\_name** (str): Name of the training job to be created. If not specified, one is generated, using the base name given to the constructor if applicable.

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

*Usage:*

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

*Arguments:*

**deep** Whether to make a deep clone.

LDAModel

*Reference LDA s3 model data created by LDA estimator.***Description**

Calling :meth:`~sagemaker.model.Model.deploy` creates an Endpoint and return a Predictor that transforms vectors to a lower-dimensional representation.

**Super classes**

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> LDAModel
```

**Methods****Public methods:**

- `LDAModel$new()`
- `LDAModel$clone()`

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

*Usage:*

```
LDAModel$new(model_data, role, sagemaker_session = NULL, ...)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`sagemaker_session` (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`...` : Keyword arguments passed to the “FrameworkModel” initializer.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

LDAPredictor

*Transforms input vectors to lower-dimesional representations.***Description**

The implementation of :meth:`~sagemaker.predictor.Predictor.predict` in this ‘Predictor’ requires a numpy “ndarray” as input. The array should contain the same number of columns as the feature-dimension of the data used to fit the model this Predictor performs inference on. :meth:`~predict()` returns a list of :class:`~sagemaker.amazon.record\_pb2.Record` objects, one for each row in the input “ndarray”. The lower dimension vector result is stored in the “projection” key of the “Record.label” field.

**Super classes**

```
sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> LDAPredictor
```

**Methods****Public methods:**

- `LDAPredictor$new()`
- `LDAPredictor$clone()`

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

*Usage:*

```
LDAPredictor$new(endpoint_name, sagemaker_session = NULL)
```

*Arguments:*

`endpoint_name` (str): Name of the Amazon SageMaker endpoint to which requests are sent.  
`sagemaker_session` (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, one is created using the default AWS configuration chain.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|               |                                                                                                 |
|---------------|-------------------------------------------------------------------------------------------------|
| LinearLearner | <i>A supervised learning algorithms used for solving classification or regression problems.</i> |
|---------------|-------------------------------------------------------------------------------------------------|

---

## Description

For input, you give the model labeled examples (x, y). x is a high-dimensional vector and y is a numeric label. For binary classification problems, the label must be either 0 or 1. For multiclass classification problems, the labels must be from 0 to num\_classes - 1. For regression problems, y is a real number. The algorithm learns a linear function, or, for classification problems, a linear threshold function, and maps a vector x to an approximation of the label y

## Super classes

```
sagemaker.mlcore::EstimatorBase -> sagemaker.mlcore::AmazonAlgorithmEstimatorBase
-> LinearLearner
```

## Public fields

```
repo_name sagemaker repo name for framework
repo_version version of framework
DEFAULT_MINI_BATCH_SIZE The size of each mini-batch to use when training.
.module mimic python module
```

## Active bindings

```
predictor_type The type of predictor to learn. Either "binary_classifier" or "multiclass_classifier"
or "regressor".
binary_classifier_model_selection_criteria One of 'accuracy', 'f1', 'f_beta', 'precision_at_target_recall',
'recall_at_target_precision', 'cross_entropy_loss', 'loss_function'
target_recall Only applicable if binary_classifier_model_selection_criteria is precision_at_target_recall
target_precision Only applicable if binary_classifier_model_selection_criteria is recall_at_target_precision.
positive_example_weight_mult The importance weight of positive examples is multiplied by
this constant.
epochs The maximum number of passes to make over the training data.
use_bias Whether to include a bias field
num_models Number of models to train in parallel
num_calibration_samples Number of observations to use from validation dataset for doing model
calibration
init_method Function to use to set the initial model weights.
init_scale For "uniform" init, the range of values.
init_sigma For "normal" init, the standard-deviation.
```

`init_bias` Initial weight for bias term

`optimizer` One of 'sgd', 'adam', 'rmsprop' or 'auto'

`loss` One of 'logistic', 'squared\_loss', 'absolute\_loss', 'hinge\_loss', 'eps\_insensitive\_squared\_loss', 'eps\_insensitive\_absolute\_loss', 'quantile\_loss', 'huber\_loss' or 'softmax\_loss' or 'auto'.

`wd` L2 regularization parameter

`l1` L1 regularization parameter.

`momentum` Momentum parameter of sgd optimizer.

`learning_rate` The SGD learning rate

`beta_1` Exponential decay rate for first moment estimates.

`beta_2` Exponential decay rate for second moment estimates.

`bias_lr_mult` Allows different learning rate for the bias term.

`bias_wd_mult` Allows different regularization for the bias term.

`use_lr_scheduler` If true, we use a scheduler for the learning rate.

`lr_scheduler_step` The number of steps between decreases of the learning rate

`lr_scheduler_factor` Every `lr_scheduler_step` the learning rate will decrease by this quantity.

`lr_scheduler_minimum_lr` Every `lr_scheduler_step` the learning rate will decrease by this quantity.

`normalize_data` Normalizes the features before training to have standard deviation of 1.0.

`normalize_label` Normalizes the regression label to have a standard deviation of 1.0.

`unbias_data` If true, features are modified to have mean 0.0.

`unbias_label` If true, labels are modified to have mean 0.0.

`num_point_for_scaler` The number of data points to use for calculating the normalizing and unbiasing terms.

`margin` The margin for hinge\_loss.

`quantile` Quantile for quantile loss.

`loss_insensitivity` Parameter for epsilon insensitive loss type.

`huber_delta` Parameter for Huber loss.

`early_stopping_patience` The number of epochs to wait before ending training if no improvement is made.

`early_stopping_tolerance` Relative tolerance to measure an improvement in loss.

`num_classes` The number of classes for the response variable.

`accuracy_top_k` The value of k when computing the Top K

`f_beta` The value of beta to use when calculating F score metrics for binary or multiclass classification.

`balance_multiclass_weights` Whether to use class weights which give each class equal importance in the loss function.

## Methods

### Public methods:

- `LinearLearner$new()`
- `LinearLearner$create_model()`
- `LinearLearner$.prepare_for_training()`
- `LinearLearner$clone()`

**Method new():** An :class:‘Estimator’ for binary classification and regression. Amazon SageMaker Linear Learner provides a solution for both classification and regression problems, allowing for exploring different training objectives simultaneously and choosing the best solution from a validation set. It allows the user to explore a large number of models and choose the best, which optimizes either continuous objectives such as mean square error, cross entropy loss, absolute error, etc., or discrete objectives suited for classification such as F1 measure, precision@recall, accuracy. The implementation provides a significant speedup over naive hyperparameter optimization techniques and an added convenience, when compared with solutions providing a solution only to continuous objectives. This Estimator may be fit via calls to :meth:‘~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.fit\_ndarray’ or :meth:‘~sagemaker.amazon\_estimator.AmazonAlgorithmEstimatorBase.fit\_record’. The former allows a LinearLearner model to be fit on a 2-dimensional numpy array. The latter requires Amazon :class:‘~sagemaker.amazon.record\_pb2.Record’ protobuf serialized data to be stored in S3. To learn more about the Amazon protobuf Record class and how to prepare bulk data in this format, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/cdf-training.html> After this Estimator is fit, model data is stored in S3. The model may be deployed to an Amazon SageMaker Endpoint by invoking :meth:‘~sagemaker.amazon.estimator.EstimatorBase.deploy’. As well as deploying an Endpoint, “deploy” returns a :class:‘~sagemaker.amazon.linear\_learner.LinearLearnerPredictor’ object that can be used to make class or regression predictions, using the trained model. LinearLearner Estimators can be configured by setting hyperparameters. The available hyperparameters for LinearLearner are documented below. For further information on the AWS LinearLearner algorithm, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/linear-learner.html>

### Usage:

```
LinearLearner$new(
  role,
  instance_count,
  instance_type,
  predictor_type,
  binary_classifier_model_selection_criteria = NULL,
  target_recall = NULL,
  target_precision = NULL,
  positive_example_weight_mult = NULL,
  epochs = NULL,
  use_bias = NULL,
  num_models = NULL,
  num_calibration_samples = NULL,
  init_method = NULL,
  init_scale = NULL,
  init_sigma = NULL,
  init_bias = NULL,
```

```

optimizer = NULL,
loss = NULL,
wd = NULL,
l1 = NULL,
momentum = NULL,
learning_rate = NULL,
beta_1 = NULL,
beta_2 = NULL,
bias_lr_mult = NULL,
bias_wd_mult = NULL,
use_lr_scheduler = NULL,
lr_scheduler_step = NULL,
lr_scheduler_factor = NULL,
lr_scheduler_minimum_lr = NULL,
normalize_data = NULL,
normalize_label = NULL,
unbias_data = NULL,
unbias_label = NULL,
num_point_for_scaler = NULL,
margin = NULL,
quantile = NULL,
loss_insensitivity = NULL,
huber_delta = NULL,
early_stopping_patience = NULL,
early_stopping_tolerance = NULL,
num_classes = NULL,
accuracy_top_k = NULL,
f_beta = NULL,
balance_multiclass_weights = NULL,
...
)

```

*Arguments:*

**role** (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if accessing AWS resource.

**instance\_count** (int): Number of Amazon EC2 instances to use for training.

**instance\_type** (str): Type of EC2 instance to use for training, for example, 'ml.c4.xlarge'.

**predictor\_type** (str): The type of predictor to learn. Either "binary\_classifier" or "multi-class\_classifier" or "regressor".

**binary\_classifier\_model\_selection\_criteria** (str): One of 'accuracy', 'f1', 'f\_beta', 'precision\_at\_target\_recall', 'recall\_at\_target\_precision', 'cross\_entropy\_loss', 'loss\_function'

**target\_recall** (float): Target recall. Only applicable if binary\_classifier\_model\_selection\_criteria is precision\_at\_target\_recall.

**target\_precision** (float): Target precision. Only applicable if binary\_classifier\_model\_selection\_criteria is recall\_at\_target\_precision.

**positive\_example\_weight\_mult** (float): The importance weight of positive examples is multiplied by this constant. Useful for skewed datasets. Only applies for classification tasks.

epochs (int): The maximum number of passes to make over the training data.  
 use\_bias (bool): Whether to include a bias field  
 num\_models (int): Number of models to train in parallel. If not set, the number of parallel models to train will be decided by the algorithm itself. One model will be trained according to the given training parameter (regularization, optimizer, loss) and the rest by close by parameters.  
 num\_calibration\_samples (int): Number of observations to use from validation dataset for doing model calibration (finding the best threshold).  
 init\_method (str): Function to use to set the initial model weights. One of "uniform" or "normal"  
 init\_scale (float): For "uniform" init, the range of values.  
 init\_sigma (float): For "normal" init, the standard-deviation.  
 init\_bias (float): Initial weight for bias term  
 optimizer (str): One of 'sgd', 'adam', 'rmsprop' or 'auto'  
 loss (str): One of 'logistic', 'squared\_loss', 'absolute\_loss', 'hinge\_loss', 'eps\_insensitive\_squared\_loss', 'eps\_insensitive\_absolute\_loss', 'quantile\_loss', 'huber\_loss' or 'softmax\_loss' or 'auto'.  
 wd (float): L2 regularization parameter i.e. the weight decay parameter. Use 0 for no L2 regularization.  
 l1 (float): L1 regularization parameter. Use 0 for no L1 regularization.  
 momentum (float): Momentum parameter of sgd optimizer.  
 learning\_rate (float): The SGD learning rate  
 beta\_1 (float): Exponential decay rate for first moment estimates. Only applies for adam optimizer.  
 beta\_2 (float): Exponential decay rate for second moment estimates. Only applies for adam optimizer.  
 bias\_lr\_mult (float): Allows different learning rate for the bias term. The actual learning rate for the bias is learning rate times bias\_lr\_mult.  
 bias\_wd\_mult (float): Allows different regularization for the bias term. The actual L2 regularization weight for the bias is wd times bias\_wd\_mult. By default there is no regularization on the bias term.  
 use\_lr\_scheduler (bool): If true, we use a scheduler for the learning rate.  
 lr\_scheduler\_step (int): The number of steps between decreases of the learning rate. Only applies to learning rate scheduler.  
 lr\_scheduler\_factor (float): Every lr\_scheduler\_step the learning rate will decrease by this quantity. Only applies for learning rate scheduler.  
 lr\_scheduler\_minimum\_lr (float): The learning rate will never decrease to a value lower than this. Only applies for learning rate scheduler.  
 normalize\_data (bool): Normalizes the features before training to have standard deviation of 1.0.  
 normalize\_label (bool): Normalizes the regression label to have a standard deviation of 1.0. If set for classification, it will be ignored.  
 unbiased\_data (bool): If true, features are modified to have mean 0.0.  
 unbiased\_label (bool): If true, labels are modified to have mean 0.0.  
 num\_point\_for\_scaler (int): The number of data points to use for calculating the normalizing and unbiasing terms.

`margin` (float): The margin for `hinge_loss`.

`quantile` (float): Quantile for quantile loss. For quantile  $q$ , the model will attempt to produce predictions such that `true_label < prediction` with probability  $q$ .

`loss_insensitivity` (float): Parameter for epsilon insensitive loss type. During training and metric evaluation, any error smaller than this is considered to be zero.

`huber_delta` (float): Parameter for Huber loss. During training and metric evaluation, compute L2 loss for errors smaller than `delta` and L1 loss for errors larger than `delta`.

`early_stopping_patience` (int): The number of epochs to wait before ending training if no improvement is made. The improvement is training loss if validation data is not provided, or else it is the validation loss or the binary classification model selection criteria like accuracy, f1-score etc. To disable early stopping, set `early_stopping_patience` to a value larger than epochs.

`early_stopping_tolerance` (float): Relative tolerance to measure an improvement in loss. If the ratio of the improvement in loss divided by the previous best loss is smaller than this value, early stopping will consider the improvement to be zero.

`num_classes` (int): The number of classes for the response variable. Required when `predictor_type` is `multiclass_classifier` and ignored otherwise. The classes are assumed to be labeled 0, ..., `num_classes` - 1.

`accuracy_top_k` (int): The value of  $k$  when computing the Top K Accuracy metric for multi-class classification. An example is scored as correct if the model assigns one of the top  $k$  scores to the true label.

`f_beta` (float): The value of  $\beta$  to use when calculating F score metrics for binary or multi-class classification. Also used if `binary_classifier_model_selection_criteria` is `f_beta`.

`balance_multiclass_weights` (bool): Whether to use class weights which give each class equal importance in the loss function. Only used when `predictor_type` is `multiclass_classifier`.

... : base class keyword argument values.

**Method** `create_model()`: Return a :class:`~sagemaker.amazon.LinearLearnerModel` referencing the latest s3 model data produced by this Estimator.

*Usage:*

```
LinearLearner.create_model(vpc_config_override = "VPC_CONFIG_DEFAULT", ...)
```

*Arguments:*

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* `'Subnets'` (list[str]): List of subnet ids. \* `'SecurityGroupIds'` (list[str]): List of security group ids.

... : Additional kwargs passed to the `LinearLearnerModel` constructor.

**Method** `.prepare_for_training()`: Set hyperparameters needed for training. This method will also validate `"source_dir"`.

*Usage:*

```
LinearLearner.prepare_for_training(
    records,
    mini_batch_size = NULL,
    job_name = NULL
)
```

*Arguments:*

`records` (RecordSet) – The records to train this Estimator on.

`mini_batch_size` (int or None) – The size of each mini-batch to use when training. If None, a default value will be used.

`job_name` (str): Name of the training job to be created. If not specified, one is generated, using the base name given to the constructor if applicable.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                    |                                               |
|--------------------|-----------------------------------------------|
| LinearLearnerModel | <i>Reference LinearLearner s3 model data.</i> |
|--------------------|-----------------------------------------------|

---

**Description**

Calling `:meth:'~sagemaker.model.Model.deploy'` creates an Endpoint and returns a `:class:'LinearLearnerPredictor'`

**Super classes**

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> LinearLearnerModel
```

**Methods****Public methods:**

- `LinearLearnerModel$new()`
- `LinearLearnerModel$clone()`

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

*Usage:*

```
LinearLearnerModel$new(model_data, role, sagemaker_session = NULL, ...)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz“ file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`sagemaker_session` (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`...` : Keyword arguments passed to the “FrameworkModel“ initializer.

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

---

LinearLearnerPredictor

*Performs binary-classification or regression prediction from input vectors.*

---

## Description

The implementation of :meth:`~sagemaker.predictor.Predictor.predict` in this ‘Predictor’ requires a numpy “ndarray” as input. The array should contain the same number of columns as the feature-dimension of the data used to fit the model this Predictor performs inference on. :func:`predict` returns a list of :class:`~sagemaker.amazon.record\_pb2.Record` objects, one for each row in the input “ndarray”. The prediction is stored in the “predicted\_label” key of the “Record.label” field.

## Super classes

```
sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> LinearLearnerPredictor
```

## Methods

### Public methods:

- [LinearLearnerPredictor\\$new\(\)](#)
- [LinearLearnerPredictor\\$clone\(\)](#)

**Method** new(): Initialize LinearLearnerPredictor Class

*Usage:*

```
LinearLearnerPredictor$new(endpoint_name, sagemaker_session = NULL)
```

*Arguments:*

endpoint\_name (str): Name of the Amazon SageMaker endpoint to which requests are sent.  
 sagemaker\_session (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, one is created using the default AWS configuration chain.

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

MXNet

*MXNet Class***Description**

Handle end-to-end training and deployment of custom MXNet code.

**Super classes**

`sagemaker.mlcore::EstimatorBase` -> `sagemaker.mlcore::Framework` -> MXNet

**Public fields**

`.LOWEST_SCRIPT_MODE_VERSION` Lowest MXNet version that can be executed  
`.module` mimic python module

**Methods****Public methods:**

- `MXNet$new()`
- `MXNet$create_model()`
- `MXNet$clone()`

**Method `new()`:** This “Estimator” executes an MXNet script in a managed MXNet execution environment, within a SageMaker Training Job. The managed MXNet environment is an Amazon-built Docker container that executes functions defined in the supplied “entry\_point” Python script. Training is started by calling `:meth:~sagemaker.amazon.estimator.Framework.fit` on this Estimator. After training is complete, calling `:meth:~sagemaker.amazon.estimator.Framework.deploy` creates a hosted SageMaker endpoint and returns an `:class:~sagemaker.amazon.mxnet.model.MXNetPredictor` instance that can be used to perform inference against the hosted model. Technical documentation on preparing MXNet scripts for SageMaker training and using the MXNet Estimator is available on the project home-page: <https://github.com/aws/sagemaker-python-sdk>

*Usage:*

```
MXNet$new(
  entry_point,
  framework_version = NULL,
  py_version = NULL,
  source_dir = NULL,
  hyperparameters = NULL,
  image_uri = NULL,
  distribution = NULL,
  ...
)
```

*Arguments:*

- entry\_point** (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to training. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.
- framework\_version** (str): MXNet version you want to use for executing your model training code. Defaults to ‘None’. Required unless “image\_uri” is provided. List of supported versions. <https://github.com/aws/sagemaker-python-sdk#mxnet-sagemaker-estimators>.
- py\_version** (str): Python version you want to use for executing your model training code. One of ‘py2’ or ‘py3’. Defaults to “None”. Required unless “image\_uri” is provided.
- source\_dir** (str): Path (absolute, relative or an S3 URI) to a directory with any other training source code dependencies aside from the entry point file (default: None). If “source\_dir” is an S3 URI, it must point to a tar.gz file. Structure within this directory are preserved when training on Amazon SageMaker.
- hyperparameters** (dict): Hyperparameters that will be used for training (default: None). The hyperparameters are made accessible as a dict[str, str] to the training code on SageMaker. For convenience, this accepts other types for keys and values, but “str()” will be called to convert them before training.
- image\_uri** (str): If specified, the estimator will use this image for training and hosting, instead of selecting the appropriate SageMaker official image based on framework\_version and py\_version. It can be an ECR url or dockerhub image and tag. Examples: \* “123412341234.dkr.ecr.us-west-2.amazonaws.com/my-custom-image:1.0” \* “custom-image:latest” If “framework\_version” or “py\_version” are “None”, then “image\_uri” is required. If also “None”, then a “ValueError” will be raised.
- distribution** (dict): A dictionary with information on how to run distributed training (default: None). Currently we support distributed training with parameter server and MPI [Horovod].
- ... : Additional kwargs passed to the :class:~sagemaker.estimator.Framework’ constructor.

**Method** `create_model()`: Create a SageMaker “MXNetModel” object that can be deployed to an “Endpoint”.

*Usage:*

```
MXNet$create_model(
    model_server_workers = NULL,
    role = NULL,
    vpc_config_override = "VPC_CONFIG_DEFAULT",
    entry_point = NULL,
    source_dir = NULL,
    dependencies = NULL,
    image_uri = NULL,
    ...
)
```

*Arguments:*

- model\_server\_workers** (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.
- role** (str): The “ExecutionRoleArn” IAM Role ARN for the “Model”, which is also used during transform jobs. If not specified, the role from the Estimator will be used.
- vpc\_config\_override** (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* ‘Subnets’ (list[str]): List of subnet ids. \* ‘SecurityGroupIds’ (list[str]): List of security group ids.

**entry\_point** (str): Path (absolute or relative) to the local Python source file which should be executed as the entry point to training. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”. If not specified, the training entry point is used.

**source\_dir** (str): Path (absolute or relative) to a directory with any other serving source code dependencies aside from the entry point file. If not specified, the model source directory from training is used.

**dependencies** (list[str]): A list of paths to directories (absolute or relative) with any additional libraries that will be exported to the container. If not specified, the dependencies from training are used. This is not supported with "local code" in Local Mode.

**image\_uri** (str): If specified, the estimator will use this image for hosting, instead of selecting the appropriate SageMaker official image based on framework\_version and py\_version. It can be an ECR url or dockerhub image and tag. Examples: \* “123412341234.dkr.ecr.us-west-2.amazonaws.com/my-custom-image:1.0” \* “custom-image:latest”

... : Additional kwargs passed to the :class:`~sagemaker.mxnet.model.MXNetModel` constructor.

**Returns:** `sagemaker.mxnet.model.MXNetModel`: A SageMaker “MXNetModel” object. See :func:`~sagemaker.mxnet.model.MXNetModel` for full details.

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

*Usage:*

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

*Arguments:*

**deep** Whether to make a deep clone.

---

MXNetModel

*MXNetModel Class*

---

## Description

An MXNet SageMaker “Model” that can be deployed to a SageMaker “Endpoint”.

## Super classes

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> sagemaker.mlcore::FrameworkModel
-> MXNetModel
```

## Public fields

`.LOWEST_MMS_VERSION` Lowest Multi Model Server MXNet version that can be executed

## Methods

### Public methods:

- `MXNetModel$new()`
- `MXNetModel$prepare_container_def()`
- `MXNetModel$serving_image_uri()`
- `MXNetModel$clone()`

**Method** `new()`: Initialize an MXNetModel.

*Usage:*

```
MXNetModel$new(
  model_data,
  role,
  entry_point,
  framework_version = NULL,
  py_version = NULL,
  image_uri = NULL,
  predictor_cls = MXNetPredictor,
  model_server_workers = NULL,
  ...
)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`entry_point` (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to model hosting. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

`framework_version` (str): MXNet version you want to use for executing your model training code. Defaults to “None”. Required unless “image\_uri” is provided.

`py_version` (str): Python version you want to use for executing your model training code. Defaults to “None”. Required unless “image\_uri” is provided.

`image_uri` (str): A Docker image URI (default: None). If not specified, a default image for MXNet will be used. If “framework\_version” or “py\_version” are “None”, then “image\_uri” is required. If also “None”, then a “ValueError” will be raised.

`predictor_cls` (callable[str, sagemaker.session.Session]): A function to call to create a predictor with an endpoint name and SageMaker “Session”. If specified, “deploy()” returns the result of invoking this function on the created endpoint name.

`model_server_workers` (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

... : Keyword arguments passed to the superclass :class:~sagemaker.model.FrameworkModel and, subsequently, its superclass :class:~sagemaker.model.Model.

**Method** `prepare_container_def()`: Return a container definition with framework configuration set in model environment variables.

*Usage:*

```
MXNetModel$prepare_container_def(instance_type = NULL, accelerator_type = NULL)
```

*Arguments:*

`instance_type` (str): The EC2 instance type to deploy this Model to. For example, 'ml.p2.xlarge'.

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the model. For example, 'ml.eia1.medium'.

*Returns:* dict[str, str]: A container definition object usable with the CreateModel API.

**Method** `serving_image_uri()`: Create a URI for the serving image.

*Usage:*

```
MXNetModel$serving_image_uri(
  region_name,
  instance_type,
  accelerator_type = NULL
)
```

*Arguments:*

`region_name` (str): AWS region where the image is uploaded.

`instance_type` (str): SageMaker instance type. Used to determine device type (cpu/gpu/family-specific optimized).

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the model (default: None). For example, 'ml.eia1.medium'.

*Returns:* str: The appropriate image URI based on the given parameters.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

MXNetPredictor

MXNetPredictor Class

---

## Description

A Predictor for inference against MXNet Endpoints. This is able to serialize Python lists, dictionaries, and numpy arrays to multidimensional tensors for MXNet inference.

## Super classes

```
sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> MXNetPredictor
```

**Methods****Public methods:**

- [MXNetPredictor\\$new\(\)](#)
- [MXNetPredictor\\$clone\(\)](#)

**Method** `new()`: Initialize an “MXNetPredictor”.

*Usage:*

```
MXNetPredictor$new(
  endpoint_name,
  sagemaker_session = NULL,
  serializer = JsonSerializer$new(),
  deserializer = JSONDeserializer$new()
)
```

*Arguments:*

`endpoint_name` (str): The name of the endpoint to perform inference on.

`sagemaker_session` (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`serializer` (callable): Optional. Default serializes input data to json. Handles dicts, lists, and numpy arrays.

`deserializer` (callable): Optional. Default parses the response using “`json.load(...)`”.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

MXNetProcessor

*MXNetProcessor class*


---

**Description**

Handles Amazon SageMaker processing tasks for jobs using MXNet containers.

**Super classes**

```
sagemaker.common::Processor -> sagemaker.common::ScriptProcessor -> sagemaker.common::FrameworkProcess
-> MXNetProcessor
```

**Public fields**

`estimator_cls` Estimator object

## Methods

### Public methods:

- `MXNetProcessor$new()`
- `MXNetProcessor$clone()`

**Method** `new()`: This processor executes a Python script in a managed MXNet execution environment. Unless “image\_uri” is specified, the MXNet environment is an Amazon-built Docker container that executes functions defined in the supplied “code” Python script.

#### Usage:

```
MXNetProcessor$new(
  framework_version,
  role,
  instance_count,
  instance_type,
  py_version = "py3",
  image_uri = NULL,
  command = NULL,
  volume_size_in_gb = 30,
  volume_kms_key = NULL,
  output_kms_key = NULL,
  code_location = NULL,
  max_runtime_in_seconds = NULL,
  base_job_name = NULL,
  sagemaker_session = NULL,
  env = NULL,
  tags = NULL,
  network_config = NULL
)
```

#### Arguments:

- `framework_version` (str): The version of the framework. Value is ignored when “image\_uri” is provided.
- `role` (str): An AWS IAM role name or ARN. Amazon SageMaker Processing uses this role to access AWS resources, such as data stored in Amazon S3.
- `instance_count` (int): The number of instances to run a processing job with.
- `instance_type` (str): The type of EC2 instance to use for processing, for example, ‘ml.c4.xlarge’.
- `py_version` (str): Python version you want to use for executing your model training code. One of ‘py2’ or ‘py3’. Defaults to ‘py3’. Value is ignored when “image\_uri” is provided.
- `image_uri` (str): The URI of the Docker image to use for the processing jobs (default: None).
- `command` ([str]): The command to run, along with any command-line flags to \*precede\* the “code script”. Example: ["python3", "-v"]. If not provided, ["python"] will be chosen (default: None).
- `volume_size_in_gb` (int): Size in GB of the EBS volume to use for storing data during processing (default: 30).
- `volume_kms_key` (str): A KMS key for the processing volume (default: None).
- `output_kms_key` (str): The KMS key ID for processing job outputs (default: None).

`code_location` (str): The S3 prefix URI where custom code will be uploaded (default: None). The code file uploaded to S3 is `'code_location/job-name/source/sourcedir.tar.gz'`. If not specified, the default "code location" is `'s3://sagemaker-default-bucket'`

`max_runtime_in_seconds` (int): Timeout in seconds (default: None). After this amount of time, Amazon SageMaker terminates the job, regardless of its current status. If `'max_runtime_in_seconds'` is not specified, the default value is 24 hours.

`base_job_name` (str): Prefix for processing name. If not specified, the processor generates a default job name, based on the processing image name and current timestamp (default: None).

`sagemaker_session` (:class:~sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker and any other AWS services needed. If not specified, the processor creates one using the default AWS configuration chain (default: None).

`env` (dict[str, str]): Environment variables to be passed to the processing jobs (default: None).

`tags` (list[dict]): List of tags to be passed to the processing job (default: None). For more, see [https://docs.aws.amazon.com/sagemaker/latest/dg/API\\_Tag.html](https://docs.aws.amazon.com/sagemaker/latest/dg/API_Tag.html).

`network_config` (:class:~sagemaker.network.NetworkConfig): A :class:~sagemaker.network.NetworkConfig object that configures network isolation, encryption of inter-container traffic, security group IDs, and subnets (default: None).

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

NTM

*An unsupervised learning algorithm used to organize a corpus of documents into topics*

---

## Description

The resulting topics contain word groupings based on their statistical distribution. Documents that contain frequent occurrences of words such as "bike", "car", "train", "mileage", and "speed" are likely to share a topic on "transportation" for example.

## Super classes

```
sagemaker.mlcore:EstimatorBase -> sagemaker.mlcore:AmazonAlgorithmEstimatorBase
-> NTM
```

## Public fields

```
repo_name sagemaker repo name for framework
repo_version version of framework
.module mimic python module
```

### Active bindings

`num_topics` The number of topics for NTM to find within the data  
`encoder_layers` Represents number of layers in the encoder and the output size of each layer  
`epochs` Maximum number of passes over the training data.  
`encoder_layers_activation` Activation function to use in the encoder layers.  
`optimizer` Optimizer to use for training.  
`tolerance` Maximum relative change in the loss function within the last `num_patience_epochs` number of epochs below which early stopping is triggered.  
`num_patience_epochs` Number of successive epochs over which early stopping criterion is evaluated.  
`batch_norm` Whether to use batch normalization during training.  
`rescale_gradient` Rescale factor for gradient  
`clip_gradient` Maximum magnitude for each gradient component.  
`weight_decay` Weight decay coefficient.  
`learning_rate` Learning rate for the optimizer.

### Methods

#### Public methods:

- `NTM$new()`
- `NTM$create_model()`
- `NTM$.prepare_for_training()`
- `NTM$clone()`

**Method `new()`:** Neural Topic Model (NTM) is :class:‘Estimator’ used for unsupervised learning.

This Estimator may be fit via calls to :meth:‘~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.fit’.

It requires Amazon :class:‘~sagemaker.amazon.record\_pb2.Record’ protobuf serialized data to be

stored in S3. There is an utility :meth:‘~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.record\_set

that can be used to upload data to S3 and creates :class:‘~sagemaker.amazon.amazon\_estimator.RecordSet’

to be passed to the ‘fit’ call. To learn more about the Amazon protobuf Record class and how to

prepare bulk data in this format, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/ntm.html>

training.html After this Estimator is fit, model data is stored in S3. The model may be deployed to

an Amazon SageMaker Endpoint by invoking :meth:‘~sagemaker.amazon.estimator.EstimatorBase.deploy’.

As well as deploying an Endpoint, deploy returns a :class:‘~sagemaker.amazon.ntm.NTMPredictor’

object that can be used for inference calls using the trained model hosted in the SageMaker End-

point. NTM Estimators can be configured by setting hyperparameters. The available hyperparam-

eters for NTM are documented below. For further information on the AWS NTM algorithm, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/ntm.html>

#### Usage:

```
NTM$new(
  role,
  instance_count,
  instance_type,
  num_topics,
```

```

        encoder_layers = NULL,
        epochs = NULL,
        encoder_layers_activation = NULL,
        optimizer = NULL,
        tolerance = NULL,
        num_patience_epochs = NULL,
        batch_norm = NULL,
        rescale_gradient = NULL,
        clip_gradient = NULL,
        weight_decay = NULL,
        learning_rate = NULL,
        ...
    )

```

*Arguments:*

**role** (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if accessing AWS resource.

**instance\_count** (int): Number of Amazon EC2 instances to use for training.

**instance\_type** (str): Type of EC2 instance to use for training, for example, 'ml.c4.xlarge'.

**num\_topics** (int): Required. The number of topics for NTM to find within the data.

**encoder\_layers** (list): Optional. Represents number of layers in the encoder and the output size of each layer.

**epochs** (int): Optional. Maximum number of passes over the training data.

**encoder\_layers\_activation** (str): Optional. Activation function to use in the encoder layers.

**optimizer** (str): Optional. Optimizer to use for training.

**tolerance** (float): Optional. Maximum relative change in the loss function within the last **num\_patience\_epochs** number of epochs below which early stopping is triggered.

**num\_patience\_epochs** (int): Optional. Number of successive epochs over which early stopping criterion is evaluated.

**batch\_norm** (bool): Optional. Whether to use batch normalization during training.

**rescale\_gradient** (float): Optional. Rescale factor for gradient.

**clip\_gradient** (float): Optional. Maximum magnitude for each gradient component.

**weight\_decay** (float): Optional. Weight decay coefficient. Adds L2 regularization.

**learning\_rate** (float): Optional. Learning rate for the optimizer.

... : base class keyword argument values.

**Method** `create_model()`: Return a :class:`~sagemaker.amazon.NTMMModel` referencing the latest s3 model data produced by this Estimator.

*Usage:*

```
NTM.create_model(vpc_config_override = "VPC_CONFIG_DEFAULT", ...)
```

*Arguments:*

**vpc\_config\_override** (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* 'Subnets' (list[str]): List of subnet ids. \* 'SecurityGroupIds' (list[str]): List of security group ids.

... : Additional kwargs passed to the NTMModel constructor.

**Method** `.prepare_for_training()`: Set hyperparameters needed for training. This method will also validate “source\_dir”.

*Usage:*

```
NTM$.prepare_for_training(records, mini_batch_size, job_name = NULL)
```

*Arguments:*

`records` (RecordSet) – The records to train this Estimator on.

`mini_batch_size` (int or None) – The size of each mini-batch to use when training. If None, a default value will be used.

`job_name` (str): Name of the training job to be created. If not specified, one is generated, using the base name given to the constructor if applicable.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

NTMModel

Reference NTM s3 model data.

---

## Description

Calling `:meth:~sagemaker.model.Model.deploy` creates an Endpoint and return a Predictor that transforms vectors to a lower-dimensional representation.

## Super classes

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> NTMModel
```

## Methods

### Public methods:

- `NTMModel$new()`
- `NTMModel$clone()`

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

*Usage:*

```
NTMModel$new(model_data, role, sagemaker_session = NULL, ...)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`sagemaker_session` (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

... : Keyword arguments passed to the “FrameworkModel” initializer.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

NTMPredictor

*Transforms input vectors to lower-dimesional representations.*


---

## Description

The implementation of :meth:~sagemaker.predictor.Predictor.predict in this ‘Predictor’ requires a numpy “ndarray” as input. The array should contain the same number of columns as the feature-dimension of the data used to fit the model this Predictor performs inference on. :meth:~predict() returns a list of :class:~sagemaker.amazon.record\_pb2.Record objects, one for each row in the input “ndarray”. The lower dimension vector result is stored in the “projection” key of the “Record.label” field.

## Super classes

```
sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> NTMPredictor
```

## Methods

### Public methods:

- `NTMPredictor$new()`
- `NTMPredictor$clone()`

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

*Usage:*

```
NTMPredictor$new(endpoint_name, sagemaker_session = NULL)
```

*Arguments:*

`endpoint_name` (str): Name of the Amazon SageMaker endpoint to which requests are sent.

`sagemaker_session` (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, one is created using the default AWS configuration chain.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

Object2Vec

*A general-purpose neural embedding algorithm that is highly customizable.*

---

## Description

It can learn low-dimensional dense embeddings of high-dimensional objects. The embeddings are learned in a way that preserves the semantics of the relationship between pairs of objects in the original space in the embedding space.

## Super classes

```
sagemaker.mlcore::EstimatorBase -> sagemaker.mlcore::AmazonAlgorithmEstimatorBase
-> Object2Vec
```

## Public fields

```
repo_name sagemaker repo name for framework
repo_version version of framework
MINI_BATCH_SIZE The size of each mini-batch to use when training.
.module mimic python module
```

## Active bindings

```
epochs Total number of epochs for SGD training
enc_dim Dimension of the output of the embedding layer
mini_batch_size mini batch size for SGD training
early_stopping_patience The allowed number of consecutive epochs without improvement before early stopping is applied
early_stopping_tolerance The value used to determine whether the algorithm has made improvement between two consecutive epochs for early stopping
dropout Dropout probability on network layers
weight_decay Weight decay parameter during optimization
bucket_width The allowed difference between data sequence length when bucketing is enabled
num_classes Number of classes for classification
mlp_layers Number of MLP layers in the network
```

`mlp_dim` Dimension of the output of MLP layer  
`mlp_activation` Type of activation function for the MLP layer  
`output_layer` Type of output layer  
`optimizer` Type of optimizer for training  
`learning_rate` Learning rate for SGD training  
`negative_sampling_rate` Negative sampling rate  
`comparator_list` Customization of comparator operator  
`tied_token_embedding_weight` Tying of token embedding layer weight  
`token_embedding_storage_type` Type of token embedding storage  
`enc0_network` Network model of encoder "enc0"  
`enc1_network` Network model of encoder "enc1"  
`enc0_cnn_filter_width` CNN filter width  
`enc1_cnn_filter_width` CNN filter width  
`enc0_max_seq_len` Maximum sequence length  
`enc1_max_seq_len` Maximum sequence length  
`enc0_token_embedding_dim` Output dimension of token embedding layer  
`enc1_token_embedding_dim` Output dimension of token embedding layer  
`enc0_vocab_size` Vocabulary size of tokens  
`enc1_vocab_size` Vocabulary size of tokens  
`enc0_layers` Number of layers in encoder  
`enc1_layers` Number of layers in encoder  
`enc0_freeze_pretrained_embedding` Freeze pretrained embedding weights  
`enc1_freeze_pretrained_embedding` Freeze pretrained embedding weights

## Methods

### Public methods:

- `Object2Vec$new()`
- `Object2Vec$create_model()`
- `Object2Vec$.prepare_for_training()`
- `Object2Vec$clone()`

**Method** `new()`: `Object2Vec` is :class:'Estimator' used for anomaly detection. This Estimator may be fit via calls to :meth:'~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.fit'. There is an utility :meth:'~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.record\_set' that can be used to upload data to S3 and creates :class:'~sagemaker.amazon.amazon\_estimator.RecordSet' to be passed to the 'fit' call. After this Estimator is fit, model data is stored in S3. The model may be deployed to an Amazon SageMaker Endpoint by invoking :meth:'~sagemaker.amazon.estimator.EstimatorBase.deploy'. As well as deploying an Endpoint, `deploy` returns a :class:'~sagemaker.amazon.Predictor' object that can be used for inference calls using the trained model hosted in the SageMaker Endpoint. `Object2Vec` Estimators can be configured by setting hyperparameters. The available hyperparameters for `Object2Vec` are documented below. For further information on the AWS `Object2Vec` algorithm, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/object2vec.html>

*Usage:*

```
Object2Vec$new(
  role,
  instance_count,
  instance_type,
  epochs,
  enc0_max_seq_len,
  enc0_vocab_size,
  enc_dim = NULL,
  mini_batch_size = NULL,
  early_stopping_patience = NULL,
  early_stopping_tolerance = NULL,
  dropout = NULL,
  weight_decay = NULL,
  bucket_width = NULL,
  num_classes = NULL,
  mlp_layers = NULL,
  mlp_dim = NULL,
  mlp_activation = NULL,
  output_layer = NULL,
  optimizer = NULL,
  learning_rate = NULL,
  negative_sampling_rate = NULL,
  comparator_list = NULL,
  tied_token_embedding_weight = NULL,
  token_embedding_storage_type = NULL,
  enc0_network = NULL,
  enc1_network = NULL,
  enc0_cnn_filter_width = NULL,
  enc1_cnn_filter_width = NULL,
  enc1_max_seq_len = NULL,
  enc0_token_embedding_dim = NULL,
  enc1_token_embedding_dim = NULL,
  enc1_vocab_size = NULL,
  enc0_layers = NULL,
  enc1_layers = NULL,
  enc0_freeze_pretrained_embedding = NULL,
  enc1_freeze_pretrained_embedding = NULL,
  ...
)
```

*Arguments:*

**role** (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if accessing AWS resource.

**instance\_count** (int): Number of Amazon EC2 instances to use for training.

**instance\_type** (str): Type of EC2 instance to use for training, for example, 'ml.c4.xlarge'.

**epochs** (int): Total number of epochs for SGD training

`enc0_max_seq_len` (int): Maximum sequence length  
`enc0_vocab_size` (int): Vocabulary size of tokens  
`enc_dim` (int): Optional. Dimension of the output of the embedding layer  
`mini_batch_size` (int): Optional. mini batch size for SGD training  
`early_stopping_patience` (int): Optional. The allowed number of consecutive epochs without improvement before early stopping is applied  
`early_stopping_tolerance` (float): Optional. The value used to determine whether the algorithm has made improvement between two consecutive epochs for early stopping  
`dropout` (float): Optional. Dropout probability on network layers  
`weight_decay` (float): Optional. Weight decay parameter during optimization  
`bucket_width` (int): Optional. The allowed difference between data sequence length when bucketing is enabled  
`num_classes` (int): Optional. Number of classes for classification  
`mlp_layers` (int): Optional. Number of MLP layers in the network  
`mlp_dim` (int): Optional. Dimension of the output of MLP layer  
`mlp_activation` (str): Optional. Type of activation function for the MLP layer  
`output_layer` (str): Optional. Type of output layer  
`optimizer` (str): Optional. Type of optimizer for training  
`learning_rate` (float): Optional. Learning rate for SGD training  
`negative_sampling_rate` (int): Optional. Negative sampling rate  
`comparator_list` (str): Optional. Customization of comparator operator  
`tied_token_embedding_weight` (bool): Optional. Tying of token embedding layer weight  
`token_embedding_storage_type` (str): Optional. Type of token embedding storage  
`enc0_network` (str): Optional. Network model of encoder "enc0"  
`enc1_network` (str): Optional. Network model of encoder "enc1"  
`enc0_cnn_filter_width` (int): Optional. CNN filter width  
`enc1_cnn_filter_width` (int): Optional. CNN filter width  
`enc1_max_seq_len` (int): Optional. Maximum sequence length  
`enc0_token_embedding_dim` (int): Optional. Output dimension of token embedding layer  
`enc1_token_embedding_dim` (int): Optional. Output dimension of token embedding layer  
`enc1_vocab_size` (int): Optional. Vocabulary size of tokens  
`enc0_layers` (int): Optional. Number of layers in encoder  
`enc1_layers` (int): Optional. Number of layers in encoder  
`enc0_freeze_pretrained_embedding` (bool): Optional. Freeze pretrained embedding weights  
`enc1_freeze_pretrained_embedding` (bool): Optional. Freeze pretrained embedding weights  
... : base class keyword argument values.  
`training` (ignored for regression problems)

**Method** `create_model()`: Return a :class:`~sagemaker.amazon.Object2VecModel` referencing the latest s3 model data produced by this Estimator.

*Usage:*

```
Object2Vec.create_model(vpc_config_override = "VPC_CONFIG_DEFAULT", ...)
```

*Arguments:*

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model.  
 Default: use subnets and security groups from this Estimator. \* `'Subnets'` (list[str]): List of subnet ids. \* `'SecurityGroupIds'` (list[str]): List of security group ids.  
 ... : Additional kwargs passed to the Object2VecModel constructor.

**Method** `.prepare_for_training()`: Set hyperparameters needed for training. This method will also validate `"source_dir"`.

*Usage:*

```
Object2Vec$.prepare_for_training(
  records,
  mini_batch_size = NULL,
  job_name = NULL
)
```

*Arguments:*

`records` (RecordSet) – The records to train this Estimator on.  
`mini_batch_size` (int or None) – The size of each mini-batch to use when training. If None, a default value will be used.  
`job_name` (str): Name of the training job to be created. If not specified, one is generated, using the base name given to the constructor if applicable.

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

*Usage:*

```
Object2Vec$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

Object2VecModel

*Reference Object2Vec s3 model data.*

---

## Description

Calling `:meth:'~sagemaker.model.Model.deploy'` creates an Endpoint and returns a Predictor that calculates anomaly scores for datapoints.

## Super classes

`sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> Object2VecModel`

## Methods

### Public methods:

- `Object2VecModel$new()`
- `Object2VecModel$clone()`

**Method** `new()`: Initialize Object2VecModel class

*Usage:*

```
Object2VecModel$new(model_data, role, sagemaker_session = NULL, ...)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`sagemaker_session` (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`...` : Keyword arguments passed to the “FrameworkModel” initializer.

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

*Usage:*

```
Object2VecModel$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

PCA

*An unsupervised machine learning algorithm to reduce feature dimensionality.*

---

**Description**

As a result, number of features within a dataset is reduced but the dataset still retain as much information as possible.

**Super classes**

```
sagemaker.mlcore::EstimatorBase -> sagemaker.mlcore::AmazonAlgorithmEstimatorBase
-> PCA
```

**Public fields**

`repo_name` sagemaker repo name for framework

`repo_version` version of framework

`DEFAULT_MINI_BATCH_SIZE` The size of each mini-batch to use when training.

`.module` mimic python module

**Active bindings**

`num_components` The number of principal components. Must be greater than zero.

`algorithm_mode` Mode for computing the principal components.

`subtract_mean` Whether the data should be unbiased both during train and at inference.

`extra_components` As the value grows larger, the solution becomes more accurate but the runtime and memory consumption increase linearly.

**Methods****Public methods:**

- `PCA$new()`
- `PCA$create_model()`
- `PCA$.prepare_for_training()`
- `PCA$clone()`

**Method** `new()`: A Principal Components Analysis (PCA) :class: '~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimator'. This Estimator may be fit via calls to :meth: '~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.fit\_numpy' or :meth: '~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.fit\_record'. The former allows a PCA model to be fit on a 2-dimensional numpy array. The latter requires Amazon :class: '~sagemaker.amazon.record\_pb2.Record' protobuf serialized data to be stored in S3. To learn more about the Amazon protobuf Record class and how to prepare bulk data in this format, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/cdf-training.html>. After this Estimator is fit, model data is stored in S3. The model may be deployed to an Amazon SageMaker Endpoint by invoking :meth: '~sagemaker.amazon.estimator.EstimatorBase.deploy'. As well as deploying an Endpoint, `deploy` returns a :class: '~sagemaker.amazon.pca.PCAPredictor' object that can be used to project input vectors to the learned lower-dimensional representation, using the trained PCA model hosted in the SageMaker Endpoint. PCA Estimators can be configured by setting hyperparameters. The available hyperparameters for PCA are documented below. For further information on the AWS PCA algorithm, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/pca.html>. This Estimator uses Amazon SageMaker PCA to perform training and host deployed models. To learn more about Amazon SageMaker PCA, please read: <https://docs.aws.amazon.com/sagemaker/latest/dg/how-pca-works.html>

*Usage:*

```
PCA$new(
  role,
  instance_count,
  instance_type,
  num_components,
  algorithm_mode = NULL,
  subtract_mean = NULL,
  extra_components = NULL,
  ...
)
```

*Arguments:*

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if accessing AWS resource.

`instance_count` (int): Number of Amazon EC2 instances to use for training.

`instance_type` (str): Type of EC2 instance to use for training, for example, 'ml.c4.xlarge'.

`num_components` (int): The number of principal components. Must be greater than zero.

`algorithm_mode` (str): Mode for computing the principal components. One of 'regular' or 'randomized'.

`subtract_mean` (bool): Whether the data should be unbiased both during train and at inference.

`extra_components` (int): As the value grows larger, the solution becomes more accurate but the runtime and memory consumption increase linearly. If this value is unset or set to -1, then a default value equal to the maximum of 10 and `num_components` will be used. Valid for randomized mode only.

... : base class keyword argument values.

**Method** `create_model()`: Return a :class:`~sagemaker.amazon.pca.PCAModel` referencing the latest s3 model data produced by this Estimator.

*Usage:*

```
PCA$create_model(vpc_config_override = "VPC_CONFIG_DEFAULT", ...)
```

*Arguments:*

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model.

Default: use subnets and security groups from this Estimator. \* 'Subnets' (list[str]): List of subnet ids. \* 'SecurityGroupIds' (list[str]): List of security group ids.

... : Additional kwargs passed to the PCAModel constructor.

**Method** `.prepare_for_training()`: Set hyperparameters needed for training.

*Usage:*

```
PCA$.prepare_for_training(records, mini_batch_size = NULL, job_name = NULL)
```

*Arguments:*

`records` (:class:`~RecordSet`): The records to train this "Estimator" on.

`mini_batch_size` (int or None): The size of each mini-batch to use when training. If "None", a default value will be used.

`job_name` (str): Name of the training job to be created. If not specified, one is generated, using the base name given to the constructor if applicable.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

PCAModel

*Reference PCA s3 model data.***Description**

Calling :meth:`~sagemaker.model.Model.deploy` creates an Endpoint and return a Predictor that transforms vectors to a lower-dimensional representation.

**Super classes**

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> PCAModel
```

**Methods****Public methods:**

- `PCAModel$new()`
- `PCAModel$clone()`

**Method** `new()`: initialize PCAModel Class

*Usage:*

```
PCAModel$new(model_data, role, sagemaker_session = NULL, ...)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz“ file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`sagemaker_session` (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`...` : Keyword arguments passed to the “FrameworkModel“ initializer.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

PCAPredictor

*Transforms input vectors to lower-dimesional representations.*

## Description

The implementation of :meth:`~sagemaker.predictor.Predictor.predict` in this ‘Predictor’ requires a numpy “ndarray” as input. The array should contain the same number of columns as the feature-dimension of the data used to fit the model this Predictor performs inference on. :meth:`~predict()` returns a list of :class:`~sagemaker.amazon.record\_pb2.Record` objects, one for each row in the input “ndarray”. The lower dimension vector result is stored in the “projection” key of the “Record.label” field.

## Super classes

```
sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> PCAPredictor
```

## Methods

### Public methods:

- `PCAPredictor$new()`
- `PCAPredictor$clone()`

**Method** `new()`: Initialize PCAPredictor Class

*Usage:*

```
PCAPredictor$new(endpoint_name, sagemaker_session = NULL)
```

*Arguments:*

`endpoint_name` (str): Name of the Amazon SageMaker endpoint to which requests are sent.  
`sagemaker_session` (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: None). If not specified, one is created using the default AWS configuration chain.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                  |                               |
|------------------|-------------------------------|
| PySparkProcessor | <i>PySparkProcessor Class</i> |
|------------------|-------------------------------|

---

## Description

Handles Amazon SageMaker processing tasks for jobs using PySpark.

## Super classes

`sagemaker.common::Processor` -> `sagemaker.common::ScriptProcessor` -> `sagemaker.mlframework::SparkProcessor`  
 -> `PySparkProcessor`

## Methods

### Public methods:

- `PySparkProcessor$new()`
- `PySparkProcessor$get_args_run()`
- `PySparkProcessor$run()`
- `PySparkProcessor$clone()`

**Method new():** Initialize an “PySparkProcessor” instance. The PySparkProcessor handles Amazon SageMaker processing tasks for jobs using SageMaker PySpark.

### Usage:

```
PySparkProcessor$new(
  role,
  instance_type,
  instance_count,
  framework_version = NULL,
  py_version = NULL,
  container_version = NULL,
  image_uri = NULL,
  volume_size_in_gb = 30,
  volume_kms_key = NULL,
  output_kms_key = NULL,
  max_runtime_in_seconds = NULL,
  base_job_name = NULL,
  sagemaker_session = NULL,
  env = NULL,
  tags = NULL,
  network_config = NULL
)
```

### Arguments:

**role (str):** An AWS IAM role name or ARN. The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

instance\_type (str): Type of EC2 instance to use for processing, for example, 'ml.c4.xlarge'.  
 instance\_count (int): The number of instances to run the Processing job with. Defaults to 1.  
 framework\_version (str): The version of SageMaker PySpark.  
 py\_version (str): The version of python.  
 container\_version (str): The version of spark container.  
 image\_uri (str): The container image to use for training.  
 volume\_size\_in\_gb (int): Size in GB of the EBS volume to use for storing data during processing (default: 30).  
 volume\_kms\_key (str): A KMS key for the processing volume.  
 output\_kms\_key (str): The KMS key id for all ProcessingOutputs.  
 max\_runtime\_in\_seconds (int): Timeout in seconds. After this amount of time Amazon SageMaker terminates the job regardless of its current status.  
 base\_job\_name (str): Prefix for processing name. If not specified, the processor generates a default job name, based on the training image name and current timestamp.  
 sagemaker\_session (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the processor creates one using the default AWS configuration chain.  
 env (dict): Environment variables to be passed to the processing job.  
 tags ([dict]): List of tags to be passed to the processing job.  
 network\_config (sagemaker.network.NetworkConfig): A NetworkConfig object that configures network isolation, encryption of inter-container traffic, security group IDs, and subnets.

**Method** `get_args_run()`: Returns a RunArgs object. This object contains the normalized inputs, outputs and arguments needed when using a “PySparkProcessor” in a :class:~sagemaker.workflow.steps.ProcessingStep

*Usage:*

```

PySparkProcessor$get_args_run(
    submit_app,
    submit_py_files = NULL,
    submit_jars = NULL,
    submit_files = NULL,
    inputs = NULL,
    outputs = NULL,
    arguments = NULL,
    job_name = NULL,
    configuration = NULL,
    spark_event_logs_s3_uri = NULL
)
  
```

*Arguments:*

submit\_app (str): Path (local or S3) to Python file to submit to Spark as the primary application. This is translated to the 'code' property on the returned 'RunArgs' object.  
 submit\_py\_files (list[str]): List of paths (local or S3) to provide for 'spark-submit --py-files' option  
 submit\_jars (list[str]): List of paths (local or S3) to provide for 'spark-submit --jars' option  
 submit\_files (list[str]): List of paths (local or S3) to provide for 'spark-submit --files' option

`inputs` (list[:class: '~sagemaker.processing.ProcessingInput']): Input files for the processing job. These must be provided as :class: '~sagemaker.processing.ProcessingInput' objects (default: None).

`outputs` (list[:class: '~sagemaker.processing.ProcessingOutput']): Outputs for the processing job. These can be specified as either path strings or :class: '~sagemaker.processing.ProcessingOutput' objects (default: None).

`arguments` (list[str]): A list of string arguments to be passed to a processing job (default: None).

`job_name` (str): Processing job name. If not specified, the processor generates a default job name, based on the base job name and current timestamp.

`configuration` (list[dict] or dict): Configuration for Hadoop, Spark, or Hive. List or dictionary of EMR-style classifications. <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-configure-apps.html>

`spark_event_logs_s3_uri` (str): S3 path where spark application events will be published to.

**Method** `run()`: Runs a processing job.

*Usage:*

```
PySparkProcessor.run(
    submit_app,
    submit_py_files = NULL,
    submit_jars = NULL,
    submit_files = NULL,
    inputs = NULL,
    outputs = NULL,
    arguments = NULL,
    wait = TRUE,
    logs = TRUE,
    job_name = NULL,
    experiment_config = NULL,
    configuration = NULL,
    spark_event_logs_s3_uri = NULL,
    kms_key = NULL
)
```

*Arguments:*

`submit_app` (str): Path (local or S3) to Python file to submit to Spark as the primary application

`submit_py_files` (list[str]): List of paths (local or S3) to provide for 'spark-submit --py-files' option

`submit_jars` (list[str]): List of paths (local or S3) to provide for 'spark-submit --jars' option

`submit_files` (list[str]): List of paths (local or S3) to provide for 'spark-submit --files' option

`inputs` (list[:class: '~sagemaker.processing.ProcessingInput']): Input files for the processing job. These must be provided as :class: '~sagemaker.processing.ProcessingInput' objects (default: None).

`outputs` (list[:class: '~sagemaker.processing.ProcessingOutput']): Outputs for the processing job. These can be specified as either path strings or :class: '~sagemaker.processing.ProcessingOutput' objects (default: None).

`arguments` (list[str]): A list of string arguments to be passed to a processing job (default: None).

`wait` (bool): Whether the call should wait until the job completes (default: True).

`logs` (bool): Whether to show the logs produced by the job. Only meaningful when `wait` is True (default: True).

`job_name` (str): Processing job name. If not specified, the processor generates a default job name, based on the base job name and current timestamp.

`experiment_config` (dict[str, str]): Experiment management configuration. Dictionary contains three optional keys: 'ExperimentName', 'TrialName', and 'TrialComponentDisplayName'.

`configuration` (list[dict] or dict): Configuration for Hadoop, Spark, or Hive. List or dictionary of EMR-style classifications. <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-configure-apps.html>

`spark_event_logs_s3_uri` (str): S3 path where spark application events will be published to.

`kms_key` (str): The ARN of the KMS key that is used to encrypt the user code file (default: None).

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

PyTorch

*PyTorch Class*


---

## Description

Handle end-to-end training and deployment of custom PyTorch code.

## Super classes

```
sagemaker.mlcore::EstimatorBase -> sagemaker.mlcore::Framework -> PyTorch
```

## Public fields

`.module` mimic python module

## Methods

### Public methods:

- `PyTorch$new()`
- `PyTorch$hyperparameters()`
- `PyTorch$create_model()`
- `PyTorch$clone()`

**Method** `new()`: This “Estimator” executes an PyTorch script in a managed PyTorch execution environment, within a SageMaker Training Job. The managed PyTorch environment is an Amazon-built Docker container that executes functions defined in the supplied “entry\_point” Python script. Training is started by calling `:meth:`~sagemaker.amazon.estimator.Framework.fit`` on this Estimator. After training is complete, calling `:meth:`~sagemaker.amazon.estimator.Framework.deploy`` creates a hosted SageMaker endpoint and returns an `:class:`~sagemaker.amazon.pytorch.model.PyTorchPredictor`` instance that can be used to perform inference against the hosted model. Technical documentation on preparing PyTorch scripts for SageMaker training and using the PyTorch Estimator is available on the project home-page: <https://github.com/aws/sagemaker-python-sdk>

*Usage:*

```
PyTorch$new(
    entry_point,
    framework_version = NULL,
    py_version = NULL,
    source_dir = NULL,
    hyperparameters = NULL,
    image_uri = NULL,
    distribution = NULL,
    ...
)
```

*Arguments:*

`entry_point` (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to training. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

`framework_version` (str): PyTorch version you want to use for executing your model training code. Defaults to “None”. Required unless “image\_uri” is provided. List of supported versions: <https://github.com/aws/sagemaker-python-sdk#pytorch-sagemaker-estimators>.

`py_version` (str): Python version you want to use for executing your model training code. One of ‘py2’ or ‘py3’. Defaults to “None”. Required unless “image\_uri” is provided.

`source_dir` (str): Path (absolute, relative or an S3 URI) to a directory with any other training source code dependencies aside from the entry point file (default: None). If “source\_dir” is an S3 URI, it must point to a tar.gz file. Structure within this directory are preserved when training on Amazon SageMaker.

`hyperparameters` (dict): Hyperparameters that will be used for training (default: None). The hyperparameters are made accessible as a dict[str, str] to the training code on SageMaker. For convenience, this accepts other types for keys and values, but “str()” will be called to convert them before training.

`image_uri` (str): If specified, the estimator will use this image for training and hosting, instead of selecting the appropriate SageMaker official image based on `framework_version` and `py_version`. It can be an ECR url or dockerhub image and tag. Examples: \* “123412341234.dkr.ecr.us-west-2.amazonaws.com/my-custom-image:1.0” \* “custom-image:latest” If “framework\_version” or “py\_version” are “None”, then “image\_uri” is required. If also “None”, then a “ValueError” will be raised.

`distribution` (list): A dictionary with information on how to run distributed training (default: None). Currently, the following are supported: distributed training with parameter servers, SageMaker Distributed (SMD) Data and Model Parallelism, and MPI. SMD Model Parallelism can only be used with MPI. To enable parameter server use the following setup:

... : Additional kwargs passed to the :class:`~sagemaker.estimator.Framework` constructor.

**Method** `hyperparameters()`: Return hyperparameters used by your custom PyTorch code during model training.

*Usage:*

```
PyTorch$hyperparameters()
```

**Method** `create_model()`: Create a SageMaker “PyTorchModel” object that can be deployed to an “Endpoint”.

*Usage:*

```
PyTorch$create_model(
    model_server_workers = NULL,
    role = NULL,
    vpc_config_override = "VPC_CONFIG_DEFAULT",
    entry_point = NULL,
    source_dir = NULL,
    dependencies = NULL,
    ...
)
```

*Arguments:*

`model_server_workers` (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

`role` (str): The “ExecutionRoleArn” IAM Role ARN for the “Model”, which is also used during transform jobs. If not specified, the role from the Estimator will be used.

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* ‘Subnets’ (list[str]): List of subnet ids. \* ‘SecurityGroupIds’ (list[str]): List of security group ids.

`entry_point` (str): Path (absolute or relative) to the local Python source file which should be executed as the entry point to training. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”. If not specified, the training entry point is used.

`source_dir` (str): Path (absolute or relative) to a directory with any other serving source code dependencies aside from the entry point file. If not specified, the model source directory from training is used.

`dependencies` (list[str]): A list of paths to directories (absolute or relative) with any additional libraries that will be exported to the container. If not specified, the dependencies from training are used. This is not supported with “local code” in Local Mode.

... : Additional kwargs passed to the :class:`~sagemaker.pytorch.model.PyTorchModel` constructor.

*Returns:* `sagemaker.pytorch.model.PyTorchModel`: A SageMaker “PyTorchModel” object. See :func:`~sagemaker.pytorch.model.PyTorchModel` for full details.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

PyTorchModel

*PyTorchModel class***Description**

An PyTorch SageMaker “Model” that can be deployed to a SageMaker “Endpoint”.

**Super classes**

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> sagemaker.mlcore::FrameworkModel
-> PyTorchModel
```

**Public fields**

.LOWEST\_MMS\_VERSION Lowest Multi Model Server PyTorch version that can be executed

**Methods****Public methods:**

- `PyTorchModel$new()`
- `PyTorchModel$register()`
- `PyTorchModel$prepare_container_def()`
- `PyTorchModel$serving_image_uri()`
- `PyTorchModel$clone()`

**Method** `new()`: Initialize a PyTorchModel.

*Usage:*

```
PyTorchModel$new(
  model_data,
  role,
  entry_point,
  framework_version = NULL,
  py_version = NULL,
  image_uri = NULL,
  predictor_cls = PyTorchPredictor,
  model_server_workers = NULL,
  ...
)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

**entry\_point** (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to model hosting. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

**framework\_version** (str): PyTorch version you want to use for executing your model training code. Defaults to None. Required unless “image\_uri” is provided.

**py\_version** (str): Python version you want to use for executing your model training code. Defaults to “None”. Required unless “image\_uri” is provided.

**image\_uri** (str): A Docker image URI (default: None). If not specified, a default image for PyTorch will be used. If “framework\_version” or “py\_version” are “None”, then “image\_uri” is required. If also “None”, then a “ValueError” will be raised.

**predictor\_cls** (callable[str, sagemaker.session.Session]): A function to call to create a predictor with an endpoint name and SageMaker “Session”. If specified, “deploy()” returns the result of invoking this function on the created endpoint name.

**model\_server\_workers** (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

... : Keyword arguments passed to the superclass :class:~sagemaker.model.FrameworkModel and, subsequently, its superclass :class:~sagemaker.model.Model’.

**Method** `register()`: Creates a model package for creating SageMaker models or listing on Marketplace.

*Usage:*

```
PyTorchModel.register(
    content_types,
    response_types,
    inference_instances,
    transform_instances,
    model_package_name = NULL,
    model_package_group_name = NULL,
    image_uri = NULL,
    model_metrics = NULL,
    metadata_properties = NULL,
    marketplace_cert = FALSE,
    approval_status = NULL,
    description = NULL,
    drift_check_baselines = NULL
)
```

*Arguments:*

**content\_types** (list): The supported MIME types for the input data.

**response\_types** (list): The supported MIME types for the output data.

**inference\_instances** (list): A list of the instance types that are used to generate inferences in real-time.

**transform\_instances** (list): A list of the instance types on which a transformation job can be run or on which an endpoint can be deployed.

**model\_package\_name** (str): Model Package name, exclusive to ‘model\_package\_group\_name’, using ‘model\_package\_name’ makes the Model Package un-versioned (default: None).

**model\_package\_group\_name** (str): Model Package Group name, exclusive to ‘model\_package\_name’, using ‘model\_package\_group\_name’ makes the Model Package versioned (default: None).

`image_uri` (str): Inference image uri for the container. Model class' self.image will be used if it is None (default: None).

`model_metrics` (ModelMetrics): ModelMetrics object (default: None).

`metadata_properties` (MetadataProperties): MetadataProperties object (default: None).

`marketplace_cert` (bool): A boolean value indicating if the Model Package is certified for AWS Marketplace (default: False).

`approval_status` (str): Model Approval Status, values can be "Approved", "Rejected", or "PendingManualApproval" (default: "PendingManualApproval").

`description` (str): Model Package description (default: None).

`drift_check_baselines` (DriftCheckBaselines): DriftCheckBaselines object (default: None).

*Returns:* A 'sagemaker.model.ModelPackage' instance.

**Method** `prepare_container_def()`: Return a container definition with framework configuration set in model environment variables.

*Usage:*

```
PyTorchModel$prepare_container_def(
    instance_type = NULL,
    accelerator_type = NULL
)
```

*Arguments:*

`instance_type` (str): The EC2 instance type to deploy this Model to. For example, 'ml.p2.xlarge'.

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the model.

*Returns:* dict[str, str]: A container definition object usable with the CreateModel API.

**Method** `serving_image_uri()`: Create a URI for the serving image.

*Usage:*

```
PyTorchModel$serving_image_uri(
    region_name,
    instance_type,
    accelerator_type = NULL
)
```

*Arguments:*

`region_name` (str): AWS region where the image is uploaded.

`instance_type` (str): SageMaker instance type. Used to determine device type (cpu/gpu/family-specific optimized).

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the model.

*Returns:* str: The appropriate image URI based on the given parameters

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                  |                                                             |
|------------------|-------------------------------------------------------------|
| PyTorchPredictor | <i>A Predictor for inference against PyTorch Endpoints.</i> |
|------------------|-------------------------------------------------------------|

---

## Description

This is able to serialize Python lists, dictionaries, and numpy arrays to multidimensional tensors for PyTorch inference.

## Super classes

`sagemaker.mlcore::PredictorBase` -> `sagemaker.mlcore::Predictor` -> `PyTorchPredictor`

## Methods

### Public methods:

- `PyTorchPredictor$new()`
- `PyTorchPredictor$clone()`

**Method** `new()`: Initialize an “PyTorchPredictor”.

*Usage:*

```
PyTorchPredictor$new(
  endpoint_name,
  sagemaker_session = NULL,
  serializer = NumpySerializer$new(),
  deserializer = NumpyDeserializer$new()
)
```

*Arguments:*

`endpoint_name` (str): The name of the endpoint to perform inference on.

`sagemaker_session` (`sagemaker.session.Session`): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`serializer` (`sagemaker.serializers.BaseSerializer`): Optional. Default serializes input data to .npy format. Handles lists and numpy arrays.

`deserializer` (`sagemaker.deserializers.BaseDeserializer`): Optional. Default parses the response from .npy format to numpy array.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                  |                               |
|------------------|-------------------------------|
| PyTorchProcessor | <i>PyTorchProcessor class</i> |
|------------------|-------------------------------|

---

## Description

Handles Amazon SageMaker processing tasks for jobs using PyTorch containers.

## Super classes

`sagemaker.common::Processor` -> `sagemaker.common::ScriptProcessor` -> `sagemaker.common::FrameworkProcessor`  
 -> `PyTorchProcessor`

## Public fields

`estimator_cls` Estimator object

## Methods

### Public methods:

- `PyTorchProcessor$new()`
- `PyTorchProcessor$clone()`

**Method** `new()`: This processor executes a Python script in a PyTorch execution environment. Unless “image\_uri” is specified, the PyTorch environment is an Amazon-built Docker container that executes functions defined in the supplied “code” Python script.

*Usage:*

```
PyTorchProcessor$new(
  framework_version,
  role,
  instance_count,
  instance_type,
  py_version = "py3",
  image_uri = NULL,
  command = NULL,
  volume_size_in_gb = 30,
  volume_kms_key = NULL,
  output_kms_key = NULL,
  code_location = NULL,
  max_runtime_in_seconds = NULL,
  base_job_name = NULL,
  sagemaker_session = NULL,
  env = NULL,
  tags = NULL,
  network_config = NULL
)
```

*Arguments:*

`framework_version` (str): The version of the framework. Value is ignored when “`image_uri`” is provided.

`role` (str): An AWS IAM role name or ARN. Amazon SageMaker Processing uses this role to access AWS resources, such as data stored in Amazon S3.

`instance_count` (int): The number of instances to run a processing job with.

`instance_type` (str): The type of EC2 instance to use for processing, for example, ‘`ml.c4.xlarge`’.

`py_version` (str): Python version you want to use for executing your model training code. One of ‘`py2`’ or ‘`py3`’. Defaults to ‘`py3`’. Value is ignored when “`image_uri`” is provided.

`image_uri` (str): The URI of the Docker image to use for the processing jobs (default: None).

`command` ([str]): The command to run, along with any command-line flags to \*precede\* the “`code script`”. Example: [“`python3`”, “`-v`”]. If not provided, [“`python`”] will be chosen (default: None).

`volume_size_in_gb` (int): Size in GB of the EBS volume to use for storing data during processing (default: 30).

`volume_kms_key` (str): A KMS key for the processing volume (default: None).

`output_kms_key` (str): The KMS key ID for processing job outputs (default: None).

`code_location` (str): The S3 prefix URI where custom code will be uploaded (default: None). The code file uploaded to S3 is ‘`code_location/job-name/source/sourcedir.tar.gz`’. If not specified, the default “code location” is ‘`s3://sagemaker-default-bucket`’.

`max_runtime_in_seconds` (int): Timeout in seconds (default: None). After this amount of time, Amazon SageMaker terminates the job, regardless of its current status. If ‘`max_runtime_in_seconds`’ is not specified, the default value is 24 hours.

`base_job_name` (str): Prefix for processing name. If not specified, the processor generates a default job name, based on the processing image name and current timestamp (default: None).

`sagemaker_session` (:class:‘~sagemaker.session.Session’): Session object which manages interactions with Amazon SageMaker and any other AWS services needed. If not specified, the processor creates one using the default AWS configuration chain (default: None).

`env` (dict[str, str]): Environment variables to be passed to the processing jobs (default: None).

`tags` (list[dict]): List of tags to be passed to the processing job (default: None). For more, see [https://docs.aws.amazon.com/sagemaker/latest/dg/API\\_Tag.html](https://docs.aws.amazon.com/sagemaker/latest/dg/API_Tag.html).

`network_config` (:class:‘~sagemaker.network.NetworkConfig’): A :class:‘~sagemaker.network.NetworkConfig’ object that configures network isolation, encryption of inter-container traffic, security group IDs, and subnets (default: None).

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                 |                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------|
| RandomCutForest | <i>An unsupervised algorithm for detecting anomalous data points within a data set.</i> |
|-----------------|-----------------------------------------------------------------------------------------|

---

## Description

These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points.

## Super classes

```
sagemaker.mlcore::EstimatorBase -> sagemaker.mlcore::AmazonAlgorithmEstimatorBase
-> RandomCutForest
```

## Public fields

repo\_name sagemaker repo name for framework  
 repo\_version version of framework  
 MINI\_BATCH\_SIZE The size of each mini-batch to use when training.  
 .module mimic python module

## Active bindings

eval\_metrics JSON list of metrics types to be used for reporting the score for the model  
 num\_trees The number of trees used in the forest.  
 num\_samples\_per\_tree The number of samples used to build each tree in the forest.  
 feature\_dim Doc string place

## Methods

### Public methods:

- `RandomCutForest$new()`
- `RandomCutForest$create_model()`
- `RandomCutForest$.prepare_for_training()`
- `RandomCutForest$clone()`

**Method new():** An 'Estimator' class implementing a Random Cut Forest. Typically used for anomaly detection, this Estimator may be fit via calls to :meth:`~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmE It requires Amazon :class:`~sagemaker.amazon.record\_pb2.Record` protobuf serialized data to be stored in S3. There is an utility :meth:`~sagemaker.amazon.amazon\_estimator.AmazonAlgorithmEstimatorBase.record\_set that can be used to upload data to S3 and creates :class:`~sagemaker.amazon.amazon\_estimator.RecordSet` to be passed to the 'fit' call. To learn more about the Amazon protobuf Record class and how to prepare bulk data in this format, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/training.html> After this Estimator is fit, model data is stored in S3. The model may be deployed to

an Amazon SageMaker Endpoint by invoking `:meth:~sagemaker.amazon.estimator.EstimatorBase.deploy`. As well as deploying an Endpoint, `deploy` returns a `:class:~sagemaker.amazon.ntm.RandomCutForestPredictor` object that can be used for inference calls using the trained model hosted in the SageMaker Endpoint. RandomCutForest Estimators can be configured by setting hyperparameters. The available hyperparameters for RandomCutForest are documented below. For further information on the AWS Random Cut Forest algorithm, please consult AWS technical documentation: <https://docs.aws.amazon.com/sagemaker/latest/dg/randomcutforest.html>

*Usage:*

```
RandomCutForest$new(
  role,
  instance_count,
  instance_type,
  num_samples_per_tree = NULL,
  num_trees = NULL,
  eval_metrics = NULL,
  ...
)
```

*Arguments:*

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if accessing AWS resource.

`instance_count` (int): Number of Amazon EC2 instances to use for training.

`instance_type` (str): Type of EC2 instance to use for training, for example, 'ml.c4.xlarge'.

`num_samples_per_tree` (int): Optional. The number of samples used to build each tree in the forest. The total number of samples drawn from the train dataset is `num_trees * num_samples_per_tree`.

`num_trees` (int): Optional. The number of trees used in the forest.

`eval_metrics` (list): Optional. JSON list of metrics types to be used for reporting the score for the model. Allowed values are "accuracy", "precision\_recall\_fscore": positive and negative precision, recall, and f1 scores. If test data is provided, the score shall be reported in terms of all requested metrics.

`...` : base class keyword argument values.

**Method** `create_model()`: Return a `:class:~sagemaker.amazon.RandomCutForestModel` referencing the latest s3 model data produced by this Estimator.

*Usage:*

```
RandomCutForest$create_model(vpc_config_override = "VPC_CONFIG_DEFAULT", ...)
```

*Arguments:*

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model.

Default: use subnets and security groups from this Estimator. \* 'Subnets' (list[str]): List of subnet ids. \* 'SecurityGroupIds' (list[str]): List of security group ids.

`...` : Additional kwargs passed to the RandomCutForestModel constructor.

**Method** `.prepare_for_training()`: Set hyperparameters needed for training. This method will also validate "source\_dir".

*Usage:*

```
RandomCutForest$.prepare_for_training(
  records,
  mini_batch_size = NULL,
  job_name = NULL
)
```

*Arguments:*

`records` (RecordSet) – The records to train this Estimator on.

`mini_batch_size` (int or None) – The size of each mini-batch to use when training. If None, a default value will be used.

`job_name` (str): Name of the training job to be created. If not specified, one is generated, using the base name given to the constructor if applicable.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

RandomCutForestModel    *Reference RandomCutForest s3 model data.*

---

## Description

Calling `:meth:'~sagemaker.model.Model.deploy'` creates an Endpoint and returns a Predictor that calculates anomaly scores for datapoints.

## Super classes

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> RandomCutForestModel
```

## Methods

**Public methods:**

- `RandomCutForestModel$new()`
- `RandomCutForestModel$clone()`

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

*Usage:*

```
RandomCutForestModel$new(model_data, role, sagemaker_session = NULL, ...)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz“ file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

sagemaker\_session (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

... : Keyword arguments passed to the “FrameworkModel” initializer.

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

*Usage:*

RandomCutForestModel\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

RandomCutForestPredictor

*Assigns an anomaly score to each of the datapoints provided.*

---

## Description

The implementation of :meth:~sagemaker.predictor.Predictor.predict in this ‘Predictor’ requires a numpy “ndarray” as input. The array should contain the same number of columns as the feature-dimension of the data used to fit the model this Predictor performs inference on.

## Super classes

sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> RandomCutForestPredictor

## Methods

### Public methods:

- RandomCutForestPredictor\$new()
- RandomCutForestPredictor\$clone()

**Method** new(): Initialize RandomCutForestPredictor class

*Usage:*

RandomCutForestPredictor\$new(endpoint\_name, sagemaker\_session = NULL)

*Arguments:*

endpoint\_name (str): Name of the Amazon SageMaker endpoint to which requests are sent.

sagemaker\_session (sagemaker.session.Session): A SageMaker Session object, used for SageMaker interactions (default: NULL). If not specified, one is created using the default AWS configuration chain.

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

*Usage:*

RandomCutForestPredictor\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

RLEstimator

*RLEstimator Class***Description**

Handle end-to-end training and deployment of custom RLEstimator code.

**Super classes**

`sagemaker.mlcore::EstimatorBase` -> `sagemaker.mlcore::Framework` -> RLEstimator

**Public fields**

COACH\_LATEST\_VERSION\_TF latest version of toolkit coach for tensorflow

COACH\_LATEST\_VERSION\_MXNET latest version of toolkit coach for mxnet

RAY\_LATEST\_VERSION latest version of toolkit ray

.module mimic python module

**Methods****Public methods:**

- `RLEstimator$new()`
- `RLEstimator$create_model()`
- `RLEstimator$training_image_uri()`
- `RLEstimator$hyperparameters()`
- `RLEstimator$default_metric_definitions()`
- `RLEstimator$clone()`

**Method new():** Creates an RLEstimator for managed Reinforcement Learning (RL). It will execute an RLEstimator script within a SageMaker Training Job. The managed RL environment is an Amazon-built Docker container that executes functions defined in the supplied “entry\_point” Python script. Training is started by calling :meth:‘~sagemaker.amazon.estimator.Framework.fit’ on this Estimator. After training is complete, calling :meth:‘~sagemaker.amazon.estimator.Framework.deploy’ creates a hosted SageMaker endpoint and based on the specified framework returns an :class:‘~sagemaker.amazon.mxnet.model.TensorFlowPredictor’ or :class:‘~sagemaker.amazon.tensorflow.model.TensorFlowPredictor’ instance that can be used to perform inference against the hosted model. Technical documentation on preparing RLEstimator scripts for SageMaker training and using the RLEstimator is available on the project homepage: <https://github.com/aws/sagemaker-python-sdk>

*Usage:*

```
RLEstimator$new(
    entry_point,
    toolkit = NULL,
    toolkit_version = NULL,
    framework = NULL,
    source_dir = NULL,
```

```

        hyperparameters = NULL,
        image_uri = NULL,
        metric_definitions = NULL,
        ...
    )

```

*Arguments:*

**entry\_point** (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to training. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

**toolkit** (sagemaker.rl.RLToolkit): RL toolkit you want to use for executing your model training code.

**toolkit\_version** (str): RL toolkit version you want to be use for executing your model training code.

**framework** (sagemaker.rl.RLFramework): Framework (MXNet or TensorFlow) you want to be used as a toolkit backed for reinforcement learning training.

**source\_dir** (str): Path (absolute, relative or an S3 URI) to a directory with any other training source code dependencies aside from the entry point file (default: NULL). If “source\_dir” is an S3 URI, it must point to a tar.gz file. Structure within this directory are preserved when training on Amazon SageMaker.

**hyperparameters** (dict): Hyperparameters that will be used for training (default: NULL). The hyperparameters are made accessible as a dict[str, str] to the training code on SageMaker. For convenience, this accepts other types for keys and values.

**image\_uri** (str): An ECR url. If specified, the estimator will use this image for training and hosting, instead of selecting the appropriate SageMaker official image based on framework\_version and py\_version. Example: 123.dkr.ecr.us-west-2.amazonaws.com/my-custom-image:1.0

**metric\_definitions** (list[dict]): A list of dictionaries that defines the metric(s) used to evaluate the training jobs. Each dictionary contains two keys: 'Name' for the name of the metric, and 'Regex' for the regular expression used to extract the metric from the logs. This should be defined only for jobs that don't use an Amazon algorithm.

... : Additional kwargs passed to the :class:~sagemaker.estimator.Framework' constructor. ..

tip:: You can find additional parameters for initializing this class at :class:~sagemaker.estimator.Framework' and :class:~sagemaker.estimator.EstimatorBase'.

**Method** `create_model()`: Create a SageMaker “RLEstimatorModel” object that can be deployed to an Endpoint.

*Usage:*

```

RLEstimator$create_model(
    role = NULL,
    vpc_config_override = "VPC_CONFIG_DEFAULT",
    entry_point = NULL,
    source_dir = NULL,
    dependencies = NULL,
    ...
)

```

*Arguments:*

**role** (str): The “ExecutionRoleArn” IAM Role ARN for the “Model”, which is also used during transform jobs. If not specified, the role from the Estimator will be used.

**vpc\_config\_override** (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* ‘Subnets’ (list[str]): List of subnet ids. \* ‘SecurityGroupIds’ (list[str]): List of security group ids.

**entry\_point** (str): Path (absolute or relative) to the Python source file which should be executed as the entry point for MXNet hosting (default: self.entry\_point). If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

**source\_dir** (str): Path (absolute or relative) to a directory with any other training source code dependencies aside from the entry point file (default: self.source\_dir). Structure within this directory are preserved when hosting on Amazon SageMaker.

**dependencies** (list[str]): A list of paths to directories (absolute or relative) with any additional libraries that will be exported to the container (default: self.dependencies). The library folders will be copied to SageMaker in the same folder where the entry\_point is copied. If the “source\_dir” points to S3, code will be uploaded and the S3 location will be used instead. This is not supported with "local code" in Local Mode.

**...** : Additional kwargs passed to the :class:‘~sagemaker.model.FrameworkModel’ constructor.

**Returns:** sagemaker.model.FrameworkModel: Depending on input parameters returns one of the following: \* :class:‘~sagemaker.model.FrameworkModel’ - if “image\_uri” is specified on the estimator; \* :class:‘~sagemaker.mxnet.MXNetModel’ - if “image\_uri” isn’t specified and MXNet is used as the RL backend; \* :class:‘~sagemaker.tensorflow.model.TensorFlowModel’ - if “image\_uri” isn’t specified and TensorFlow is used as the RL backend.

**Method** `training_image_uri()`: Return the Docker image to use for training. The :meth:‘~sagemaker.estimator.Estimator.training\_image\_uri’ method, which does the model training, calls this method to find the image to use for model training.

*Usage:*

```
RLEstimator.training_image_uri()
```

**Returns:** str: The URI of the Docker image.

**Method** `hyperparameters()`: Return hyperparameters used by your custom TensorFlow code during model training.

*Usage:*

```
RLEstimator.hyperparameters()
```

**Method** `default_metric_definitions()`: Provides default metric definitions based on provided toolkit.

*Usage:*

```
RLEstimator.default_metric_definitions(toolkit)
```

*Arguments:*

**toolkit** (sagemaker.rl.RLToolkit): RL Toolkit to be used for training.

**Returns:** list: metric definitions

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

---

RLFramework

*RLFramework enum environment list*


---

**Description**

Framework (MXNet, TensorFlow or PyTorch) you want to be used as a toolkit backed for reinforcement learning training.

**Usage**

```
RLFramework
```

**Format**

An object of class Enum (inherits from environment) of length 3.

**Value**

environment containing [TENSORFLOW, MXNET, PYTORCH]

---

RLToolkit

*RLToolkit enum environment list*


---

**Description**

RL toolkit you want to use for executing your model training code.

**Usage**

```
RLToolkit
```

**Format**

An object of class Enum (inherits from environment) of length 2.

**Value**

environment containing [COACH, RAY]

SKLearn

*Scikit-learn Class***Description**

Handle end-to-end training and deployment of custom Scikit-learn code.

**Super classes**

`sagemaker.mlcore::EstimatorBase` -> `sagemaker.mlcore::Framework` -> SKLearn

**Public fields**

`.module` mimic python module

**Methods****Public methods:**

- `SKLearn$new()`
- `SKLearn$create_model()`
- `SKLearn$clone()`

**Method** `new()`: This “Estimator“ executes an Scikit-learn script in a managed Scikit-learn execution environment, within a SageMaker Training Job. The managed Scikit-learn environment is an Amazon-built Docker container that executes functions defined in the supplied “entry\_point“ Python script. Training is started by calling `:meth:~sagemaker.amazon.estimator.Framework.fit` on this Estimator. After training is complete, calling `:meth:~sagemaker.amazon.estimator.Framework.deploy` creates a hosted SageMaker endpoint and returns an `:class:~sagemaker.amazon.sklearn.model.SKLearnPredictor` instance that can be used to perform inference against the hosted model. Technical documentation on preparing Scikit-learn scripts for SageMaker training and using the Scikit-learn Estimator is available on the project home-page: <https://github.com/aws/sagemaker-python-sdk>

*Usage:*

```
SKLearn$new(
  entry_point,
  framework_version = NULL,
  py_version = "py3",
  source_dir = NULL,
  hyperparameters = NULL,
  image_uri = NULL,
  ...
)
```

*Arguments:*

`entry_point` (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to training. If “source\_dir“ is specified, then “entry\_point“ must point to a file located at the root of “source\_dir“.

`framework_version` (str): Scikit-learn version you want to use for executing your model training code. Defaults to “None”. Required unless “`image_uri`” is provided. List of supported versions: <https://github.com/aws/sagemaker-python-sdk#sklearn-sagemaker-estimators>

`py_version` (str): Python version you want to use for executing your model training code (default: ‘py3’). Currently, ‘py3’ is the only supported version. If “None” is passed in, “`image_uri`” must be provided.

`source_dir` (str): Path (absolute, relative or an S3 URI) to a directory with any other training source code dependencies aside from the entry point file (default: None). If “`source_dir`” is an S3 URI, it must point to a tar.gz file. Structure within this directory are preserved when training on Amazon SageMaker.

`hyperparameters` (dict): Hyperparameters that will be used for training (default: None). The hyperparameters are made accessible as a dict[str, str] to the training code on SageMaker. For convenience, this accepts other types for keys and values, but “`str()`” will be called to convert them before training.

`image_uri` (str): If specified, the estimator will use this image for training and hosting, instead of selecting the appropriate SageMaker official image based on `framework_version` and `py_version`. It can be an ECR url or dockerhub image and tag. Examples: 123.dkr.ecr.us-west-2.amazonaws.com/my-custom-image:1.0 custom-image:latest. If “`framework_version`” or “`py_version`” are “None”, then “`image_uri`” is required. If also “None”, then a “ValueError” will be raised.

... : Additional kwargs passed to the :class:~sagemaker.estimator.Framework constructor.

**Method** `create_model()`: Create a SageMaker “SKLearnModel” object that can be deployed to an “Endpoint”.

*Usage:*

```
SKLearn$create_model(
  model_server_workers = NULL,
  role = NULL,
  vpc_config_override = "VPC_CONFIG_DEFAULT",
  entry_point = NULL,
  source_dir = NULL,
  dependencies = NULL,
  ...
)
```

*Arguments:*

`model_server_workers` (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

`role` (str): The “ExecutionRoleArn” IAM Role ARN for the “Model”, which is also used during transform jobs. If not specified, the role from the Estimator will be used.

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* ‘Subnets’ (list[str]): List of subnet ids. \* ‘SecurityGroupIds’ (list[str]): List of security group ids.

`entry_point` (str): Path (absolute or relative) to the local Python source file which should be executed as the entry point to training. If “`source_dir`” is specified, then “`entry_point`” must point to a file located at the root of “`source_dir`”. If not specified, the training entry point is used.

`source_dir` (str): Path (absolute or relative) to a directory with any other serving source code dependencies aside from the entry point file. If not specified, the model source directory from training is used.

`dependencies` (list[str]): A list of paths to directories (absolute or relative) with any additional libraries that will be exported to the container. If not specified, the dependencies from training are used. This is not supported with "local code" in Local Mode.

... : Additional kwargs passed to the :class:`~sagemaker.sklearn.model.SKLearnModel` constructor.

**Returns:** `sagemaker.sklearn.model.SKLearnModel`: A SageMaker "SKLearnModel" object. See :func:`~sagemaker.sklearn.model.SKLearnModel` for full details.

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

*Usage:*

```
SKLearnModel.clone(deep = False)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

SKLearnModel

*SKLearnModel Class*

---

## Description

An Scikit-learn SageMaker "Model" that can be deployed to a SageMaker "Endpoint".

## Super classes

```
sagemaker.mlcore:ModelBase -> sagemaker.mlcore:Model -> sagemaker.mlcore:FrameworkModel
-> SKLearnModel
```

## Methods

### Public methods:

- `SKLearnModel.new()`
- `SKLearnModel.prepare_container_def()`
- `SKLearnModel.serving_image_uri()`
- `SKLearnModel.clone()`

**Method** `new()`: Initialize an SKLearnModel.

*Usage:*

```
SKLearnModel.new(
    model_data,
    role,
    entry_point,
    framework_version = NULL,
    py_version = "py3",
```

```

        image_uri = NULL,
        predictor_cls = SKLearnPredictor,
        model_server_workers = NULL,
        ...
    )

```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`entry_point` (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to model hosting. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

`framework_version` (str): Scikit-learn version you want to use for executing your model training code. Defaults to “None”. Required unless “image\_uri” is provided.

`py_version` (str): Python version you want to use for executing your model training code (default: ‘py3’). Currently, ‘py3’ is the only supported version. If “None” is passed in, “image\_uri” must be provided.

`image_uri` (str): A Docker image URI (default: None). If not specified, a default image for Scikit-learn will be used. If “framework\_version” or “py\_version” are “None”, then “image\_uri” is required. If also “None”, then a “ValueError” will be raised.

`predictor_cls` (callable[str, sagemaker.session.Session]): A function to call to create a predictor with an endpoint name and SageMaker “Session”. If specified, “deploy()” returns the result of invoking this function on the created endpoint name.

`model_server_workers` (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

... : Keyword arguments passed to the “FrameworkModel” initializer.

**Method** `prepare_container_def()`: Return a container definition with framework configuration set in model environment variables.

*Usage:*

```

SKLearnModel.prepare_container_def(
    instance_type = NULL,
    accelerator_type = NULL
)

```

*Arguments:*

`instance_type` (str): The EC2 instance type to deploy this Model to. This parameter is unused because Scikit-learn supports only CPU.

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the model. This parameter is unused because accelerator types are not supported by SKLearnModel.

*Returns:* dict[str, str]: A container definition object usable with the CreateModel API.

**Method** `serving_image_uri()`: Create a URI for the serving image.

*Usage:*

```
SKLearnModel$serving_image_uri(region_name, instance_type)
```

*Arguments:*

region\_name (str): AWS region where the image is uploaded.

instance\_type (str): SageMaker instance type.

*Returns:* str: The appropriate image URI based on the given parameters.

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

---

SKLearnPredictor

*A Predictor for inference against Scikit-learn Endpoints.*


---

**Description**

This is able to serialize Python lists, dictionaries, and numpy arrays to multidimensional tensors for Scikit-learn inference.

**Super classes**

```
sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> SKLearnPredictor
```

**Methods****Public methods:**

- [SKLearnPredictor\\$new\(\)](#)
- [SKLearnPredictor\\$clone\(\)](#)

**Method** new(): Initialize an “SKLearnPredictor”.

*Usage:*

```
SKLearnPredictor$new(
  endpoint_name,
  sagemaker_session = NULL,
  serializer = NumpySerializer$new(),
  deserializer = NumpyDeserializer$new()
)
```

*Arguments:*

endpoint\_name (str): The name of the endpoint to perform inference on.

sagemaker\_session (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`serializer` (sagemaker.serializers.BaseSerializer): Optional. Default serializes input data to .npy format. Handles lists and numpy arrays.

`deserializer` (sagemaker.deserializers.BaseDeserializer): Optional. Default parses the response from .npy format to numpy array.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

SKLearnProcessor

SKLearnProcessor Class

---

## Description

Handles Amazon SageMaker processing tasks for jobs using scikit-learn.

## Super classes

```
sagemaker.common::Processor -> sagemaker.common::ScriptProcessor -> sagemaker.common::FrameworkProcess
-> SKLearnProcessor
```

## Public fields

`estimator_cls` Estimator object

## Methods

### Public methods:

- [SKLearnProcessor\\$new\(\)](#)
- [SKLearnProcessor\\$clone\(\)](#)

**Method** `new()`: Initialize an “SKLearnProcessor” instance. The SKLearnProcessor handles Amazon SageMaker processing tasks for jobs using scikit-learn.

*Usage:*

```
SKLearnProcessor$new(
  framework_version,
  role,
  instance_type,
  instance_count,
  py_version = "py3",
  image_uri = NULL,
  command = NULL,
  volume_size_in_gb = 30,
  volume_kms_key = NULL,
```

```

        output_kms_key = NULL,
        code_location = NULL,
        max_runtime_in_seconds = NULL,
        base_job_name = NULL,
        sagemaker_session = NULL,
        env = NULL,
        tags = NULL,
        network_config = NULL
    )

```

*Arguments:*

`framework_version` (str): The version of the framework. Value is ignored when “image\_uri” is provided.

`role` (str): An AWS IAM role name or ARN. Amazon SageMaker Processing uses this role to access AWS resources, such as data stored in Amazon S3.

`instance_type` (str): The type of EC2 instance to use for processing, for example, ‘ml.c4.xlarge’.

`instance_count` (int): The number of instances to run a processing job with.

`py_version` (str): Python version you want to use for executing your model training code. One of ‘py2’ or ‘py3’. Defaults to ‘py3’. Value is ignored when “image\_uri” is provided.

`image_uri` (str): The URI of the Docker image to use for the processing jobs (default: None).

`command` ([str]): The command to run, along with any command-line flags to \*precede\* the “code script”. Example: [“python3”, “-v”]. If not provided, [“python”] will be chosen (default: None).

`volume_size_in_gb` (int): Size in GB of the EBS volume to use for storing data during processing (default: 30).

`volume_kms_key` (str): A KMS key for the processing volume (default: None).

`output_kms_key` (str): The KMS key ID for processing job outputs (default: None).

`code_location` (str): The S3 prefix URI where custom code will be uploaded (default: None). The code file uploaded to S3 is ‘code\_location/job-name/source/sourcedir.tar.gz’. If not specified, the default “code location” is ‘s3://sagemaker-default-bucket’

`max_runtime_in_seconds` (int): Timeout in seconds (default: None). After this amount of time, Amazon SageMaker terminates the job, regardless of its current status. If ‘max\_runtime\_in\_seconds’ is not specified, the default value is 24 hours.

`base_job_name` (str): Prefix for processing name. If not specified, the processor generates a default job name, based on the processing image name and current timestamp (default: None).

`sagemaker_session` (:class:‘~sagemaker.session.Session’): Session object which manages interactions with Amazon SageMaker and any other AWS services needed. If not specified, the processor creates one using the default AWS configuration chain (default: None).

`env` (dict[str, str]): Environment variables to be passed to the processing jobs (default: None).

`tags` (list[dict]): List of tags to be passed to the processing job (default: None). For more, see [https://docs.aws.amazon.com/sagemaker/latest/dg/API\\_Tag.html](https://docs.aws.amazon.com/sagemaker/latest/dg/API_Tag.html).

`network_config` (:class:‘~sagemaker.network.NetworkConfig’): A :class:‘~sagemaker.network.NetworkConfig’ object that configures network isolation, encryption of inter-container traffic, security group IDs, and subnets (default: None).

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

---

|                   |                                |
|-------------------|--------------------------------|
| SparkJarProcessor | <i>SparkJarProcessor Class</i> |
|-------------------|--------------------------------|

---

**Description**

Handles Amazon SageMaker processing tasks for jobs using Spark with Java or Scala Jars.

**Super classes**

```
sagemaker.common::Processor -> sagemaker.common::ScriptProcessor -> sagemaker.mlframework::SparkProcessor
-> SparkJarProcessor
```

**Methods****Public methods:**

- [SparkJarProcessor\\$new\(\)](#)
- [SparkJarProcessor\\$get\\_run\\_args\(\)](#)
- [SparkJarProcessor\\$run\(\)](#)
- [SparkJarProcessor\\$clone\(\)](#)

**Method new():** Initialize a “SparkJarProcessor” instance. The SparkProcessor handles Amazon SageMaker processing tasks for jobs using SageMaker Spark.

*Usage:*

```
SparkJarProcessor$new(
  role,
  instance_type,
  instance_count,
  framework_version = NULL,
  py_version = NULL,
  container_version = NULL,
  image_uri = NULL,
  volume_size_in_gb = 30,
  volume_kms_key = NULL,
  output_kms_key = NULL,
  max_runtime_in_seconds = NULL,
  base_job_name = NULL,
  sagemaker_session = NULL,
  env = NULL,
  tags = NULL,
  network_config = NULL
)
```

*Arguments:*

`role` (str): An AWS IAM role name or ARN. The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`instance_type` (str): Type of EC2 instance to use for processing, for example, 'ml.c4.xlarge'.

`instance_count` (int): The number of instances to run the Processing job with. Defaults to 1.

`framework_version` (str): The version of SageMaker PySpark.

`py_version` (str): The version of python.

`container_version` (str): The version of spark container.

`image_uri` (str): The container image to use for training.

`volume_size_in_gb` (int): Size in GB of the EBS volume to use for storing data during processing (default: 30).

`volume_kms_key` (str): A KMS key for the processing volume.

`output_kms_key` (str): The KMS key id for all ProcessingOutputs.

`max_runtime_in_seconds` (int): Timeout in seconds. After this amount of time Amazon SageMaker terminates the job regardless of its current status.

`base_job_name` (str): Prefix for processing name. If not specified, the processor generates a default job name, based on the training image name and current timestamp.

`sagemaker_session` (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the processor creates one using the default AWS configuration chain.

`env` (dict): Environment variables to be passed to the processing job.

`tags` ([dict]): List of tags to be passed to the processing job.

`network_config` (sagemaker.network.NetworkConfig): A NetworkConfig object that configures network isolation, encryption of inter-container traffic, security group IDs, and subnets.

**Method** `get_run_args()`: This object contains the normalized inputs, outputs and arguments needed when using a “SparkJarProcessor” in a :class:`~sagemaker.workflow.steps.ProcessingStep`.

*Usage:*

```
SparkJarProcessor$get_run_args(
    submit_app,
    submit_class = NULL,
    submit_jars = NULL,
    submit_files = NULL,
    inputs = NULL,
    outputs = NULL,
    arguments = NULL,
    job_name = NULL,
    configuration = NULL,
    spark_event_logs_s3_uri = NULL
)
```

*Arguments:*

`submit_app` (str): Path (local or S3) to Python file to submit to Spark as the primary application. This is translated to the 'code' property on the returned 'RunArgs' object

submit\_class (str): Java class reference to submit to Spark as the primary application  
 submit\_jars (list[str]): List of paths (local or S3) to provide for 'spark-submit --jars' option  
 submit\_files (list[str]): List of paths (local or S3) to provide for 'spark-submit --files' option  
 inputs (list[:class:'~sagemaker.processing.ProcessingInput']): Input files for the processing job. These must be provided as :class:'~sagemaker.processing.ProcessingInput' objects (default: None).  
 outputs (list[:class:'~sagemaker.processing.ProcessingOutput']): Outputs for the processing job. These can be specified as either path strings or :class:'~sagemaker.processing.ProcessingOutput' objects (default: None).  
 arguments (list[str]): A list of string arguments to be passed to a processing job (default: None).  
 job\_name (str): Processing job name. If not specified, the processor generates a default job name, based on the base job name and current timestamp.  
 configuration (list[dict] or dict): Configuration for Hadoop, Spark, or Hive. List or dictionary of EMR-style classifications. <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-configure-apps.html>  
 spark\_event\_logs\_s3\_uri (str): S3 path where spark application events will be published to.  
*Returns:* Returns a RunArgs object.

**Method run():** Runs a processing job.

*Usage:*

```

SparkJarProcessor$.run(
  submit_app,
  submit_class = NULL,
  submit_jars = NULL,
  submit_files = NULL,
  inputs = NULL,
  outputs = NULL,
  arguments = NULL,
  wait = TRUE,
  logs = TRUE,
  job_name = NULL,
  experiment_config = NULL,
  configuration = NULL,
  spark_event_logs_s3_uri = NULL,
  kms_key = NULL
)
  
```

*Arguments:*

submit\_app (str): Path (local or S3) to Jar file to submit to Spark as the primary application  
 submit\_class (str): Java class reference to submit to Spark as the primary application  
 submit\_jars (list[str]): List of paths (local or S3) to provide for 'spark-submit --jars' option  
 submit\_files (list[str]): List of paths (local or S3) to provide for 'spark-submit --files' option  
 inputs (list[:class:'~sagemaker.processing.ProcessingInput']): Input files for the processing job. These must be provided as :class:'~sagemaker.processing.ProcessingInput' objects (default: None).

**outputs** (list[:class:'~sagemaker.processing.ProcessingOutput']): Outputs for the processing job. These can be specified as either path strings or :class:'~sagemaker.processing.ProcessingOutput' objects (default: None).

**arguments** (list[str]): A list of string arguments to be passed to a processing job (default: None).

**wait** (bool): Whether the call should wait until the job completes (default: True).

**logs** (bool): Whether to show the logs produced by the job. Only meaningful when wait is True (default: True).

**job\_name** (str): Processing job name. If not specified, the processor generates a default job name, based on the base job name and current timestamp.

**experiment\_config** (dict[str, str]): Experiment management configuration. Dictionary contains three optional keys: 'ExperimentName', 'TrialName', and 'TrialComponentDisplayName'.

**configuration** (list[dict] or dict): Configuration for Hadoop, Spark, or Hive. List or dictionary of EMR-style classifications. <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-configure-apps.html>

**spark\_event\_logs\_s3\_uri** (str): S3 path where spark application events will be published to.

**kms\_key** (str): The ARN of the KMS key that is used to encrypt the user code file (default: None).

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

*Usage:*

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

*Arguments:*

**deep** Whether to make a deep clone.

---

SparkMLModel

*SparkMLModel class*

---

## Description

Model data and S3 location holder for MLeap serialized SparkML model. Calling :meth:'~sagemaker.model.Model.deploy' creates an Endpoint and return a Predictor to performs predictions against an MLeap serialized SparkML model .

## Super classes

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> SparkMLModel
```

## Methods

### Public methods:

- `SparkMLModel$new()`
- `SparkMLModel$clone()`

**Method** `new()`: Initialize a SparkMLModel.

*Usage:*

```
SparkMLModel$new(  
  model_data,  
  role = NULL,  
  spark_version = "2.4",  
  sagemaker_session = NULL,  
  ...  
)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file. For SparkML, this will be the output that has been produced by the Spark job after serializing the Model via MLeap.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`spark_version` (str): Spark version you want to use for executing the inference (default: ‘2.4’).

`sagemaker_session` (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain. For local mode, please do not pass this variable.

... : Additional parameters passed to the :class:‘~sagemaker.model.Model’ constructor.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

SparkMLPredictor

*Performs predictions against an MLeap serialized SparkML model.*

---

## Description

The implementation of :meth:‘~sagemaker.predictor.Predictor.predict’ in this ‘Predictor’ requires a json as input. The input should follow the json format as documented. “predict()” returns a csv output, comma separated if the output is a list.

## Super classes

`sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> SparkMLPredictor`

## Methods

### Public methods:

- [SparkMLPredictor\\$new\(\)](#)
- [SparkMLPredictor\\$clone\(\)](#)

**Method** `new()`: Initializes a `SparkMLPredictor` which should be used with `SparkMLModel` to perform predictions against SparkML models serialized via MLeap. The response is returned in text/csv format which is the default response format for SparkML Serving container.

#### Usage:

```
SparkMLPredictor$new(
  endpoint_name,
  sagemaker_session = NULL,
  serializer = CSVSerializer$new(),
  ...
)
```

#### Arguments:

`endpoint_name` (str): The name of the endpoint to perform inference on.

`sagemaker_session` (`sagemaker.session.Session`): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`serializer` (`sagemaker.serializers.BaseSerializer`): Optional. Default serializes input data to text/csv.

`...` : Additional parameters passed to the `:class:~sagemaker.Predictor` constructor.

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

#### Usage:

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

#### Arguments:

`deep` Whether to make a deep clone.

---

TensorFlow

*TensorFlow Class*

---

## Description

Handle end-to-end training and deployment of user-provided TensorFlow code.

## Super classes

[sagemaker.mlcore::EstimatorBase](#) -> [sagemaker.mlcore::Framework](#) -> TensorFlow

## Public fields

`.module` mimic python module

## Methods

### Public methods:

- `TensorFlow$new()`
- `TensorFlow$create_model()`
- `TensorFlow$hyperparameters()`
- `TensorFlow$transformer()`
- `TensorFlow$clone()`

**Method** `new()`: Initialize a “TensorFlow” estimator.

#### Usage:

```
TensorFlow$new(
  py_version = NULL,
  framework_version = NULL,
  model_dir = NULL,
  image_uri = NULL,
  distribution = NULL,
  ...
)
```

#### Arguments:

`py_version` (str): Python version you want to use for executing your model training code. Defaults to “None”. Required unless “image\_uri” is provided.

`framework_version` (str): TensorFlow version you want to use for executing your model training code. Defaults to “None”. Required unless “image\_uri” is provided. List of supported versions: <https://github.com/aws/sagemaker-python-sdk#tensorflow-sagemaker-estimators>.

`model_dir` (str): S3 location where the checkpoint data and models can be exported to during training (default: None). It will be passed in the training script as one of the command line arguments. If not specified, one is provided based on your training configuration: \* \*distributed training with SMDistributed or MPI with Horovod\* - “/opt/ml/model” \* \*single-machine training or distributed training without MPI\* - “s3://output\_path/model” \* \*Local Mode with local sources (file:// instead of s3://)\* - “/opt/ml/shared/model” To disable having “model\_dir” passed to your training script, set “model\_dir=False”.

`image_uri` (str): If specified, the estimator will use this image for training and hosting, instead of selecting the appropriate SageMaker official image based on `framework_version` and `py_version`. It can be an ECR url or dockerhub image and tag. Examples: 123.dkr.ecr.us-west-2.amazonaws.com/my-custom-image:1.0 custom-image:latest. If “framework\_version” or “py\_version” are “None”, then “image\_uri” is required. If also “None”, then a “ValueError” will be raised.

`distribution` (dict): A dictionary with information on how to run distributed training (default: None). Currently, the following are supported: distributed training with parameter servers, SageMaker Distributed (SMD) Data and Model Parallelism, and MPI. SMD Model Parallelism can only be used with MPI. To enable parameter server use the following setup:

```
.. code:: python
"parameter_server": "enabled": True
To enable MPI: .. code:: python
"mpi": "enabled": True
To enable SMDistributed Data Parallel or Model Parallel: .. code:: python
```

```
"smdistributed": "dataparallel": "enabled": True, "modelparallel": "enabled": True, "parameters":
```

... : Additional kwargs passed to the Framework constructor.

**Method** `create_model()`: Create a “TensorFlowModel” object that can be used for creating SageMaker model entities, deploying to a SageMaker endpoint, or starting SageMaker Batch Transform jobs.

*Usage:*

```
TensorFlow#create_model(
    role = NULL,
    vpc_config_override = "VPC_CONFIG_DEFAULT",
    entry_point = NULL,
    source_dir = NULL,
    dependencies = NULL,
    ...
)
```

*Arguments:*

`role` (str): The “TensorFlowModel”, which is also used during transform jobs. If not specified, the role from the Estimator is used.

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* ‘Subnets’ (list[str]): List of subnet ids. \* ‘SecurityGroupIds’ (list[str]): List of security group ids.

`entry_point` (str): Path (absolute or relative) to the local Python source file which should be executed as the entry point to training. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”. If not specified and “endpoint\_type” is ‘tensorflow-serving’, no entry point is used. If “endpoint\_type” is also “None”, then the training entry point is used.

`source_dir` (str): Path (absolute or relative or an S3 URI) to a directory with any other serving source code dependencies aside from the entry point file (default: None).

`dependencies` (list[str]): A list of paths to directories (absolute or relative) with any additional libraries that will be exported to the container (default: None).

... : Additional kwargs passed to :class:~sagemaker.tensorflow.model.TensorFlowModel’.

*Returns:* sagemaker.tensorflow.model.TensorFlowModel: A “TensorFlowModel” object. See :class:~sagemaker.tensorflow.model.TensorFlowModel’ for full details.

**Method** `hyperparameters()`: Return hyperparameters used by your custom TensorFlow code during model training.

*Usage:*

```
TensorFlow$hyperparameters()
```

**Method** `transformer()`: Return a “Transformer” that uses a SageMaker Model based on the training job. It reuses the SageMaker Session and base job name used by the Estimator.

*Usage:*

```
TensorFlow$transformer(
    instance_count,
    instance_type,
```

```

    strategy = NULL,
    assemble_with = NULL,
    output_path = NULL,
    output_kms_key = NULL,
    accept = NULL,
    env = NULL,
    max_concurrent_transforms = NULL,
    max_payload = NULL,
    tags = NULL,
    role = NULL,
    volume_kms_key = NULL,
    entry_point = NULL,
    vpc_config_override = "VPC_CONFIG_DEFAULT",
    enable_network_isolation = NULL,
    model_name = NULL
)

```

*Arguments:*

`instance_count` (int): Number of EC2 instances to use.

`instance_type` (str): Type of EC2 instance to use, for example, 'ml.c4.xlarge'.

`strategy` (str): The strategy used to decide how to batch records in a single request (default: None). Valid values: 'MultiRecord' and 'SingleRecord'.

`assemble_with` (str): How the output is assembled (default: None). Valid values: 'Line' or 'None'.

`output_path` (str): S3 location for saving the transform result. If not specified, results are stored to a default bucket.

`output_kms_key` (str): Optional. KMS key ID for encrypting the transform output (default: None).

`accept` (str): The accept header passed by the client to the inference endpoint. If it is supported by the endpoint, it will be the format of the batch transform output.

`env` (dict): Environment variables to be set for use during the transform job (default: None).

`max_concurrent_transforms` (int): The maximum number of HTTP requests to be made to each individual transform container at one time.

`max_payload` (int): Maximum size of the payload in a single HTTP request to the container in MB.

`tags` (list[dict]): List of tags for labeling a transform job. If none specified, then the tags used for the training job are used for the transform job.

`role` (str): The IAM Role ARN for the "TensorFlowModel", which is also used during transform jobs. If not specified, the role from the Estimator is used.

`volume_kms_key` (str): Optional. KMS key ID for encrypting the volume attached to the ML compute instance (default: None).

`entry_point` (str): Path (absolute or relative) to the local Python source file which should be executed as the entry point to training. If "source\_dir" is specified, then "entry\_point" must point to a file located at the root of "source\_dir". If not specified and "endpoint\_type" is 'tensorflow-serving', no entry point is used. If "endpoint\_type" is also "None", then the training entry point is used.

`vpc_config_override` (dict[str, list[str]]): Optional override for the VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* `'Subnets'` (list[str]): List of subnet ids. \* `'SecurityGroupIds'` (list[str]): List of security group ids.

`enable_network_isolation` (bool): Specifies whether container will run in network isolation mode. Network isolation mode restricts the container access to outside networks (such as the internet). The container does not make any inbound or outbound network calls. If True, a channel named "code" will be created for any user entry script for inference. Also known as Internet-free mode. If not specified, this setting is taken from the estimator's current configuration.

`model_name` (str): Name to use for creating an Amazon SageMaker model. If not specified, the estimator generates a default job name based on the training image name and current timestamp.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                 |                              |
|-----------------|------------------------------|
| TensorFlowModel | <i>TensorFlowModel Class</i> |
|-----------------|------------------------------|

---

## Description

A “FrameworkModel” implementation for inference with TensorFlow Serving.

## Super classes

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> sagemaker.mlcore::FrameworkModel
-> TensorFlowModel
```

## Public fields

`LOG_LEVEL_PARAM_NAME` logging level

`LOG_LEVEL_MAP` logging level map

`LATEST_EIA_VERSION` latest eia version supported

## Methods

### Public methods:

- `TensorFlowModel$new()`
- `TensorFlowModel$register()`
- `TensorFlowModel$deploy()`
- `TensorFlowModel$prepare_container_def()`
- `TensorFlowModel$serving_image_uri()`

- `TensorFlowModel$clone()`

**Method** `new()`: Initialize a Model.

*Usage:*

```
TensorFlowModel$new(
  model_data,
  role,
  entry_point = NULL,
  image_uri = NULL,
  framework_version = NULL,
  container_log_level = NULL,
  predictor_cls = TensorFlowPredictor,
  ...
)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`entry_point` (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to model hosting. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

`image_uri` (str): A Docker image URI (default: None). If not specified, a default image for TensorFlow Serving will be used. If “framework\_version” is “None”, then “image\_uri” is required. If also “None”, then a “ValueError” will be raised.

`framework_version` (str): Optional. TensorFlow Serving version you want to use. Defaults to “None”. Required unless “image\_uri” is provided.

`container_log_level` (int): Log level to use within the container (default: logging.ERROR). Valid values are defined in the Python logging module.

`predictor_cls` (callable[str, sagemaker.session.Session]): A function to call to create a predictor with an endpoint name and SageMaker “Session”. If specified, “deploy()” returns the result of invoking this function on the created endpoint name.

... : Keyword arguments passed to the superclass :class:`~sagemaker.model.FrameworkModel` and, subsequently, its superclass :class:`~sagemaker.model.Model`. .. tip:: You can find additional parameters for initializing this class at :class:`~sagemaker.model.FrameworkModel` and :class:`~sagemaker.model.Model`.

**Method** `register()`: Creates a model package for creating SageMaker models or listing on Marketplace.

*Usage:*

```
TensorFlowModel$register(
  content_types,
  response_types,
  inference_instances,
  transform_instances,
```

```

    model_package_name = NULL,
    model_package_group_name = NULL,
    image_uri = NULL,
    model_metrics = NULL,
    metadata_properties = NULL,
    marketplace_cert = FALSE,
    approval_status = NULL,
    description = NULL
)

```

*Arguments:*

`content_types` (list): The supported MIME types for the input data.

`response_types` (list): The supported MIME types for the output data.

`inference_instances` (list): A list of the instance types that are used to generate inferences in real-time.

`transform_instances` (list): A list of the instance types on which a transformation job can be run or on which an endpoint can be deployed.

`model_package_name` (str): Model Package name, exclusive to 'model\_package\_group\_name', using 'model\_package\_name' makes the Model Package un-versioned (default: None).

`model_package_group_name` (str): Model Package Group name, exclusive to 'model\_package\_name', using 'model\_package\_group\_name' makes the Model Package versioned (default: None).

`image_uri` (str): Inference image uri for the container. Model class' self.image will be used if it is None (default: None).

`model_metrics` (ModelMetrics): ModelMetrics object (default: None).

`metadata_properties` (MetadataProperties): MetadataProperties object (default: None).

`marketplace_cert` (bool): A boolean value indicating if the Model Package is certified for AWS Marketplace (default: False).

`approval_status` (str): Model Approval Status, values can be "Approved", "Rejected", or "PendingManualApproval" (default: "PendingManualApproval").

`description` (str): Model Package description (default: None).

*Returns:* str: A string of SageMaker Model Package ARN.

**Method** `deploy()`: Deploy a Tensorflow “Model“ to a SageMaker “Endpoint“.

*Usage:*

```

TensorFlowModel$deploy(
  initial_instance_count = NULL,
  instance_type = NULL,
  serializer = NULL,
  deserializer = NULL,
  accelerator_type = NULL,
  endpoint_name = NULL,
  tags = NULL,
  kms_key = NULL,
  wait = TRUE,
  data_capture_config = NULL,
  update_endpoint = NULL,
  serverless_inference_config = NULL
)

```

*Arguments:*

`initial_instance_count` (int): The initial number of instances to run in the “Endpoint” created from this “Model”.

`instance_type` (str): The EC2 instance type to deploy this Model to. For example, 'ml.p2.xlarge', or 'local' for local mode.

`serializer` (:class:`~sagemaker.serializers.BaseSerializer`): A serializer object, used to encode data for an inference endpoint (default: None). If “serializer” is not None, then “serializer” will override the default serializer. The default serializer is set by the “predictor\_cls”.

`deserializer` (:class:`~sagemaker.deserializers.BaseDeserializer`): A deserializer object, used to decode data from an inference endpoint (default: None). If “deserializer” is not None, then “deserializer” will override the default deserializer. The default deserializer is set by the “predictor\_cls”.

`accelerator_type` (str): Type of Elastic Inference accelerator to deploy this model for model loading and inference, for example, 'ml.eia1.medium'. If not specified, no Elastic Inference accelerator will be attached to the endpoint. For more information: <https://docs.aws.amazon.com/sagemaker/latest/dg/>

`endpoint_name` (str): The name of the endpoint to create (Default: NULL). If not specified, a unique endpoint name will be created.

`tags` (List[dict[str, str]]): The list of tags to attach to this specific endpoint.

`kms_key` (str): The ARN of the KMS key that is used to encrypt the data on the storage volume attached to the instance hosting the endpoint.

`wait` (bool): Whether the call should wait until the deployment of this model completes (default: True).

`data_capture_config` (sagemaker.model\_monitor.DataCaptureConfig): Specifies configuration related to Endpoint data capture for use with Amazon SageMaker Model Monitoring. Default: None.

`update_endpoint` : Placeholder

`serverless_inference_config` (ServerlessInferenceConfig): Specifies configuration related to serverless endpoint. Use this configuration when trying to create serverless endpoint and make serverless inference. If empty object passed through, we will use pre-defined values in “ServerlessInferenceConfig” class to deploy serverless endpoint (default: None)

*Returns:* callable[string, sagemaker.session.Session] or None: Invocation of “self.predictor\_cls” on the created endpoint name, if “self.predictor\_cls” is not None. Otherwise, return None.

**Method** `prepare_container_def()`: Prepare the container definition.

*Usage:*

```
TensorFlowModel$prepare_container_def(
    instance_type = NULL,
    accelerator_type = NULL
)
```

*Arguments:*

`instance_type` : Instance type of the container.

`accelerator_type` : Accelerator type, if applicable.

*Returns:* A container definition for deploying a “Model” to an “Endpoint”.

**Method** `serving_image_uri()`: Create a URI for the serving image.

*Usage:*

```
TensorFlowModel$serving_image_uri()
```

*Arguments:*

`region_name` (str): AWS region where the image is uploaded.

`instance_type` (str): SageMaker instance type. Used to determine device type (cpu/gpu/family-specific optimized).

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the

`model` (default: None). For example, 'ml.eia1.medium'.

*Returns:* str: The appropriate image URI based on the given parameters.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

TensorFlowPredictor      *TensorFlowPredictor Class*

---

**Description**

A “Predictor” implementation for inference against TensorFlow Serving endpoints.

**Super classes**

```
sagemaker.mlcore::PredictorBase -> sagemaker.mlcore::Predictor -> TensorFlowPredictor
```

**Methods****Public methods:**

- TensorFlowPredictor\$new()
- TensorFlowPredictor\$classify()
- TensorFlowPredictor\$regress()
- TensorFlowPredictor\$predict()
- TensorFlowPredictor\$clone()

**Method** `new()`: Initialize a “TensorFlowPredictor”. See :class:`~sagemaker.predictor.Predictor` for more info about parameters.

*Usage:*

```
TensorFlowPredictor$new(
  endpoint_name,
  sagemaker_session = NULL,
  serializer = JsonSerializer$new(),
  deserializer = JSONDeserializer$new(),
  model_name = NULL,
  model_version = NULL,
  ...
)
```

*Arguments:*

`endpoint_name` (str): The name of the endpoint to perform inference on.

`sagemaker_session` (sagemaker.session.Session): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`serializer` (callable): Optional. Default serializes input data to json. Handles dicts, lists, and numpy arrays.

`deserializer` (callable): Optional. Default parses the response using “`json.load(...)`”.

`model_name` (str): Optional. The name of the SavedModel model that should handle the request. If not specified, the endpoint’s default model will handle the request.

`model_version` (str): Optional. The version of the SavedModel model that should handle the request. If not specified, the latest version of the model will be used.

`...` : Additional parameters passed to the Predictor constructor.

**Method** `classify()`: Placeholder*Usage:*

```
TensorFlowPredictor$classify(data)
```

*Arguments:*

`data` :

**Method** `regress()`: Placeholder*Usage:*

```
TensorFlowPredictor$regress(data)
```

*Arguments:*

`data` :

**Method** `predict()`: Return the inference from the specified endpoint.*Usage:*

```
TensorFlowPredictor$predict(data, initial_args = NULL)
```

*Arguments:*

`data` (object): Input data for which you want the model to provide inference. If a serializer was specified when creating the Predictor, the result of the serializer is sent as input data. Otherwise the data must be sequence of bytes, and the predict method then sends the bytes in the request body as is.

`initial_args` (list[str,str]): Optional. Default arguments for boto3 “`invoke_endpoint`” call. Default is NULL (no default arguments).

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

---

|                     |                                  |
|---------------------|----------------------------------|
| TensorFlowProcessor | <i>TensorFlowProcessor Class</i> |
|---------------------|----------------------------------|

---

## Description

Handles Amazon SageMaker processing tasks for jobs using TensorFlow containers.

## Super classes

```
sagemaker.common::Processor -> sagemaker.common::ScriptProcessor -> sagemaker.common::FrameworkProcess
-> TensorFlowProcessor
```

## Public fields

estimator\_cls Estimator object

## Methods

### Public methods:

- [TensorFlowProcessor\\$new\(\)](#)
- [TensorFlowProcessor\\$clone\(\)](#)

**Method** new(): This processor executes a Python script in a TensorFlow execution environment. Unless “image\_uri” is specified, the TensorFlow environment is an Amazon-built Docker container that executes functions defined in the supplied “code” Python script.

*Usage:*

```
TensorFlowProcessor$new(
  framework_version,
  role,
  instance_count,
  instance_type,
  py_version = "py3",
  image_uri = NULL,
  command = NULL,
  volume_size_in_gb = 30,
  volume_kms_key = NULL,
  output_kms_key = NULL,
  code_location = NULL,
  max_runtime_in_seconds = NULL,
  base_job_name = NULL,
```

```

    sagemaker_session = NULL,
    env = NULL,
    tags = NULL,
    network_config = NULL
)

```

**Arguments:**

**framework\_version** (str): The version of the framework. Value is ignored when “image\_uri” is provided.

**role** (str): An AWS IAM role name or ARN. Amazon SageMaker Processing uses this role to access AWS resources, such as data stored in Amazon S3.

**instance\_count** (int): The number of instances to run a processing job with.

**instance\_type** (str): The type of EC2 instance to use for processing, for example, ‘ml.c4.xlarge’.

**py\_version** (str): Python version you want to use for executing your model training code. One of ‘py2’ or ‘py3’. Defaults to ‘py3’. Value is ignored when “image\_uri” is provided.

**image\_uri** (str): The URI of the Docker image to use for the processing jobs (default: None).

**command** ([str]): The command to run, along with any command-line flags to \*precede\* the “code script”. Example: ["python3", "-v"]. If not provided, ["python"] will be chosen (default: None).

**volume\_size\_in\_gb** (int): Size in GB of the EBS volume to use for storing data during processing (default: 30).

**volume\_kms\_key** (str): A KMS key for the processing volume (default: None).

**output\_kms\_key** (str): The KMS key ID for processing job outputs (default: None).

**code\_location** (str): The S3 prefix URI where custom code will be uploaded (default: None). The code file uploaded to S3 is ‘code\_location/job-name/source/sourcedir.tar.gz’. If not specified, the default “code location” is ‘s3://sagemaker-default-bucket’.

**max\_runtime\_in\_seconds** (int): Timeout in seconds (default: None). After this amount of time, Amazon SageMaker terminates the job, regardless of its current status. If ‘max\_runtime\_in\_seconds’ is not specified, the default value is 24 hours.

**base\_job\_name** (str): Prefix for processing name. If not specified, the processor generates a default job name, based on the processing image name and current timestamp (default: None).

**sagemaker\_session** (:class:‘~sagemaker.session.Session’): Session object which manages interactions with Amazon SageMaker and any other AWS services needed. If not specified, the processor creates one using the default AWS configuration chain (default: None).

**env** (dict[str, str]): Environment variables to be passed to the processing jobs (default: None).

**tags** (list[dict]): List of tags to be passed to the processing job (default: None). For more, see [https://docs.aws.amazon.com/sagemaker/latest/dg/API\\_Tag.html](https://docs.aws.amazon.com/sagemaker/latest/dg/API_Tag.html).

**network\_config** (:class:‘~sagemaker.network.NetworkConfig’): A :class:‘~sagemaker.network.NetworkConfig’ object that configures network isolation, encryption of inter-container traffic, security group IDs, and subnets (default: None).

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

**Usage:**

```
TensorFlowProcessor$.clone(deep = FALSE)
```

**Arguments:**

**deep** Whether to make a deep clone.

XGBoost

*XGBoost Class***Description**

Handle end-to-end training and deployment of XGBoost booster training or training using customer provided XGBoost entry point script.

**Super classes**

`sagemaker.mlcore::EstimatorBase` -> `sagemaker.mlcore::Framework` -> XGBoost

**Public fields**

`.module` mimic python module

**Methods****Public methods:**

- `XGBoost$new()`
- `XGBoost$create_model()`
- `XGBoost$attach()`
- `XGBoost$clone()`

**Method** `new()`: This “Estimator” executes an XGBoost based SageMaker Training Job. The managed XGBoost environment is an Amazon-built Docker container that executes functions defined in the supplied “entry\_point” Python script. Training is started by calling `:meth:~sagemaker.amazon.estimator.Framework` on this Estimator. After training is complete, calling `:meth:~sagemaker.amazon.estimator.Framework.deploy` creates a hosted SageMaker endpoint and returns an `:class:~sagemaker.amazon.xgboost.model.XGBoostPredictor` instance that can be used to perform inference against the hosted model. Technical documentation on preparing XGBoost scripts for SageMaker training and using the XGBoost Estimator is available on the project home-page: <https://github.com/aws/sagemaker-python-sdk>

*Usage:*

```
XGBoost$new(
  entry_point,
  framework_version,
  source_dir = NULL,
  hyperparameters = NULL,
  py_version = "py3",
  image_uri = NULL,
  ...
)
```

*Arguments:*

`entry_point` (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to training. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

`framework_version` (str): XGBoost version you want to use for executing your model training code.

`source_dir` (str): Path (absolute, relative or an S3 URI) to a directory with any other training source code dependencies aside from the entry point file (default: None). If “source\_dir” is an S3 URI, it must point to a tar.gz file. Structure within this directory are preserved when training on Amazon SageMaker.

`hyperparameters` (dict): Hyperparameters that will be used for training (default: None). The hyperparameters are made accessible as a dict[str, str] to the training code on SageMaker. For convenience, this accepts other types for keys and values, but “str()” will be called to convert them before training.

`py_version` (str): Python version you want to use for executing your model training code (default: ‘py3’).

`image_uri` (str): If specified, the estimator will use this image for training and hosting, instead of selecting the appropriate SageMaker official image based on `framework_version` and `py_version`. It can be an ECR url or dockerhub image and tag. Examples: 123.dkr.ecr.us-west-2.amazonaws.com/my-custom-image:1.0 custom-image:latest.

... : Additional kwargs passed to the :class:~sagemaker.estimator.Framework‘ constructor.

**Method** `create_model()`: Create a SageMaker “XGBoostModel” object that can be deployed to an “Endpoint”.

*Usage:*

```
XGBoost$create_model(
    model_server_workers = NULL,
    role = NULL,
    vpc_config_override = "VPC_CONFIG_DEFAULT",
    entry_point = NULL,
    source_dir = NULL,
    dependencies = NULL,
    ...
)
```

*Arguments:*

`model_server_workers` (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

`role` (str): The “ExecutionRoleArn” IAM Role ARN for the “Model”, which is also used during transform jobs. If not specified, the role from the Estimator will be used.

`vpc_config_override` (dict[str, list[str]]): Optional override for VpcConfig set on the model. Default: use subnets and security groups from this Estimator. \* ‘Subnets’ (list[str]): List of subnet ids. \* ‘SecurityGroupIds’ (list[str]): List of security group ids.

`entry_point` (str): Path (absolute or relative) to the local Python source file which should be executed as the entry point to training. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”. If not specified, the training entry point is used.

`source_dir` (str): Path (absolute or relative) to a directory with any other serving source code dependencies aside from the entry point file. If not specified, the model source directory from training is used.

`dependencies` (list[str]): A list of paths to directories (absolute or relative) with any additional libraries that will be exported to the container. If not specified, the dependencies from training are used. This is not supported with "local code" in Local Mode.

... : Additional kwargs passed to the :class:`~sagemaker.xgboost.model.XGBoostModel` constructor.

*Returns:* `sagemaker.xgboost.model.XGBoostModel`: A SageMaker "XGBoostModel" object. See :func:`~sagemaker.xgboost.model.XGBoostModel` for full details.

**Method** `attach()`: Attach to an existing training job. Create an Estimator bound to an existing training job, each subclass is responsible to implement "`_prepare_init_params_from_job_description()`" as this method delegates the actual conversion of a training job description to the arguments that the class constructor expects. After attaching, if the training job has a Complete status, it can be "`deploy()`" ed to create a SageMaker Endpoint and return a "Predictor". If the training job is in progress, attach will block and display log messages from the training job, until the training job completes. Examples: `>> my_estimator.fit(wait=False)` `>> training_job_name = my_estimator.latest_training_job.name` Later on: `>> attached_estimator = Estimator.attach(training_job_name)` `>> attached_estimator.deploy()`

*Usage:*

```
XGBoost$attach(
    training_job_name,
    sagemaker_session = NULL,
    model_channel_name = "model"
)
```

*Arguments:*

`training_job_name` (str): The name of the training job to attach to.

`sagemaker_session` (`sagemaker.session.Session`): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`model_channel_name` (str): Name of the channel where pre-trained model data will be downloaded (default: 'model'). If no channel with the same name exists in the training job, this option will be ignored.

*Returns:* Instance of the calling "Estimator" Class with the attached training job.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|              |                           |
|--------------|---------------------------|
| XGBoostModel | <i>XGBoostModel Class</i> |
|--------------|---------------------------|

---

## Description

An XGBoost SageMaker “Model” that can be deployed to a SageMaker “Endpoint”.

## Super classes

```
sagemaker.mlcore::ModelBase -> sagemaker.mlcore::Model -> sagemaker.mlcore::FrameworkModel
-> XGBoostModel
```

## Methods

### Public methods:

- `XGBoostModel$new()`
- `XGBoostModel$prepare_container_def()`
- `XGBoostModel$serving_image_uri()`
- `XGBoostModel$clone()`

**Method** `new()`: Initialize an XGBoostModel.

*Usage:*

```
XGBoostModel$new(
  model_data,
  role,
  entry_point,
  framework_version,
  image_uri = NULL,
  py_version = "py3",
  predictor_cls = XGBoostPredictor,
  model_server_workers = NULL,
  ...
)
```

*Arguments:*

`model_data` (str): The S3 location of a SageMaker model data “.tar.gz” file.

`role` (str): An AWS IAM role (either name or full ARN). The Amazon SageMaker training jobs and APIs that create Amazon SageMaker endpoints use this role to access training data and model artifacts. After the endpoint is created, the inference code might use the IAM role, if it needs to access an AWS resource.

`entry_point` (str): Path (absolute or relative) to the Python source file which should be executed as the entry point to model hosting. If “source\_dir” is specified, then “entry\_point” must point to a file located at the root of “source\_dir”.

`framework_version` (str): XGBoost version you want to use for executing your model training code.

`image_uri` (str): A Docker image URI (default: None). If not specified, a default image for XGBoost is be used.

`py_version` (str): Python version you want to use for executing your model training code (default: 'py3').

`predictor_cls` (callable[str, sagemaker.session.Session]): A function to call to create a predictor with an endpoint name and SageMaker “Session“. If specified, “`deploy()`“ returns the result of invoking this function on the created endpoint name.

`model_server_workers` (int): Optional. The number of worker processes used by the inference server. If None, server will use one worker per vCPU.

... : Keyword arguments passed to the “FrameworkModel“ initializer.

**Method** `prepare_container_def()`: Return a container definition with framework configuration set in model environment variables.

*Usage:*

```
XGBoostModel$prepare_container_def(
    instance_type = NULL,
    accelerator_type = NULL
)
```

*Arguments:*

`instance_type` (str): The EC2 instance type to deploy this Model to. This parameter is unused because XGBoost supports only CPU.

`accelerator_type` (str): The Elastic Inference accelerator type to deploy to the instance for loading and making inferences to the model. This parameter is unused because accelerator types are not supported by XGBoostModel.

*Returns:* dict[str, str]: A container definition object usable with the CreateModel API.

**Method** `serving_image_uri()`: Create a URI for the serving image.

*Usage:*

```
XGBoostModel$serving_image_uri(region_name, instance_type)
```

*Arguments:*

`region_name` (str): AWS region where the image is uploaded.

`instance_type` (str): SageMaker instance type. Must be a CPU instance type.

*Returns:* str: The appropriate image URI based on the given parameters.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                  |                               |
|------------------|-------------------------------|
| XGBoostPredictor | <i>XGBoostPredictor Class</i> |
|------------------|-------------------------------|

---

## Description

Predictor for inference against XGBoost Endpoints. This is able to serialize Python lists, dictionaries, and numpy arrays to xgb.DMatrix for XGBoost inference.

## Super classes

`sagemaker.mlcore::PredictorBase` -> `sagemaker.mlcore::Predictor` -> XGBoostPredictor

## Methods

### Public methods:

- `XGBoostPredictor$new()`
- `XGBoostPredictor$clone()`

**Method** `new()`: Initialize an “XGBoostPredictor”.

*Usage:*

```
XGBoostPredictor$new(
  endpoint_name,
  sagemaker_session = NULL,
  serializer = LibSVMSerializer$new(),
  deserializer = CSVDeserializer$new()
)
```

*Arguments:*

`endpoint_name` (str): The name of the endpoint to perform inference on.

`sagemaker_session` (`sagemaker.session.Session`): Session object which manages interactions with Amazon SageMaker APIs and any other AWS services needed. If not specified, the estimator creates one using the default AWS configuration chain.

`serializer` (`sagemaker.serializers.BaseSerializer`): Optional. Default serializes input data to LibSVM format

`deserializer` (`sagemaker.deserializers.BaseDeserializer`): Optional. Default parses the response from text/csv to a Python list.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

---

|                  |                               |
|------------------|-------------------------------|
| XGBoostProcessor | <i>XGBoostProcessor class</i> |
|------------------|-------------------------------|

---

**Description**

Handles Amazon SageMaker processing tasks for jobs using XGBoost containers.

**Super classes**

```
sagemaker.common::Processor -> sagemaker.common::ScriptProcessor -> sagemaker.common::FrameworkProcess
-> XGBoostProcessor
```

**Public fields**

estimator\_cls Estimator object

**Methods****Public methods:**

- `XGBoostProcessor$new()`
- `XGBoostProcessor$clone()`

**Method** `new()`: This processor executes a Python script in an XGBoost execution environment. Unless “image\_uri” is specified, the XGBoost environment is an Amazon-built Docker container that executes functions defined in the supplied “code” Python script.

*Usage:*

```
XGBoostProcessor$new(
  framework_version,
  role,
  instance_count,
  instance_type,
  py_version = "py3",
  image_uri = NULL,
  command = NULL,
  volume_size_in_gb = 30,
  volume_kms_key = NULL,
  output_kms_key = NULL,
  code_location = NULL,
  max_runtime_in_seconds = NULL,
  base_job_name = NULL,
  sagemaker_session = NULL,
  env = NULL,
  tags = NULL,
  network_config = NULL
)
```

*Arguments:*

`framework_version` (str): The version of the framework. Value is ignored when “`image_uri`” is provided.

`role` (str): An AWS IAM role name or ARN. Amazon SageMaker Processing uses this role to access AWS resources, such as data stored in Amazon S3.

`instance_count` (int): The number of instances to run a processing job with.

`instance_type` (str): The type of EC2 instance to use for processing, for example, ‘`ml.c4.xlarge`’.

`py_version` (str): Python version you want to use for executing your model training code. One of ‘`py2`’ or ‘`py3`’. Defaults to ‘`py3`’. Value is ignored when “`image_uri`” is provided.

`image_uri` (str): The URI of the Docker image to use for the processing jobs (default: None).

`command` ([str]): The command to run, along with any command-line flags to \*precede\* the “`code script`”. Example: [“`python3`”, “`-v`”]. If not provided, [“`python`”] will be chosen (default: None).

`volume_size_in_gb` (int): Size in GB of the EBS volume to use for storing data during processing (default: 30).

`volume_kms_key` (str): A KMS key for the processing volume (default: None).

`output_kms_key` (str): The KMS key ID for processing job outputs (default: None).

`code_location` (str): The S3 prefix URI where custom code will be uploaded (default: None). The code file uploaded to S3 is ‘`code_location/job-name/source/sourcedir.tar.gz`’. If not specified, the default “code location” is ‘`s3://sagemaker-default-bucket`’.

`max_runtime_in_seconds` (int): Timeout in seconds (default: None). After this amount of time, Amazon SageMaker terminates the job, regardless of its current status. If ‘`max_runtime_in_seconds`’ is not specified, the default value is 24 hours.

`base_job_name` (str): Prefix for processing name. If not specified, the processor generates a default job name, based on the processing image name and current timestamp (default: None).

`sagemaker_session` (:class:‘~sagemaker.session.Session’): Session object which manages interactions with Amazon SageMaker and any other AWS services needed. If not specified, the processor creates one using the default AWS configuration chain (default: None).

`env` (dict[str, str]): Environment variables to be passed to the processing jobs (default: None).

`tags` (list[dict]): List of tags to be passed to the processing job (default: None). For more, see [https://docs.aws.amazon.com/sagemaker/latest/dg/API\\_Tag.html](https://docs.aws.amazon.com/sagemaker/latest/dg/API_Tag.html).

`network_config` (:class:‘~sagemaker.network.NetworkConfig’): A :class:‘~sagemaker.network.NetworkConfig’ object that configures network isolation, encryption of inter-container traffic, security group IDs, and subnets (default: None).

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

# Index

## \* datasets

- RLFramework, [107](#)
- RLToolkit, [107](#)
- .SparkProcessorBase, [4](#)
  
- AutoML, [7](#)
- AutoMLInput, [13](#)
- AutoMLJob, [14](#)
  
- CandidateEstimator, [15](#)
- CandidateStep, [17](#)
- Chainer, [18](#)
- ChainerModel, [21](#)
- ChainerPredictor, [23](#)
  
- FactorizationMachines, [24](#)
- FactorizationMachinesModel, [28](#)
- FactorizationMachinesPredictor, [29](#)
  
- HuggingFace, [30](#)
- HuggingFaceModel, [33](#)
- HuggingFacePredictor, [36](#)
- HuggingFaceProcessor, [37](#)
  
- IPInsights, [39](#)
- IPInsightsModel, [42](#)
- IPInsightsPredictor, [43](#)
  
- KMeans, [44](#)
- KMeansModel, [47](#)
- KMeansPredictor, [48](#)
- KNN, [49](#)
- KNNModel, [51](#)
- KNNPredictor, [52](#)
  
- LDA, [53](#)
- LDAModel, [56](#)
- LDAPredictor, [57](#)
- LinearLearner, [58](#)
- LinearLearnerModel, [64](#)
- LinearLearnerPredictor, [65](#)

- MXNet, [66](#)
- MXNetModel, [68](#)
- MXNetPredictor, [70](#)
- MXNetProcessor, [71](#)
  
- NTM, [73](#)
- NTMModel, [76](#)
- NTMPredictor, [77](#)
  
- Object2Vec, [78](#)
- Object2VecModel, [82](#)
  
- PCA, [83](#)
- PCAModel, [86](#)
- PCAPredictor, [87](#)
- PySparkProcessor, [88](#)
- PyTorch, [91](#)
- PyTorchModel, [94](#)
- PyTorchPredictor, [97](#)
- PyTorchProcessor, [98](#)
  
- RandomCutForest, [100](#)
- RandomCutForestModel, [102](#)
- RandomCutForestPredictor, [103](#)
- RLEstimator, [104](#)
- RLFramework, [107](#)
- RLToolkit, [107](#)
  
- sagemaker.common::Job, [14](#)
- sagemaker.common::FrameworkProcessor, [37](#), [71](#), [98](#), [113](#), [130](#), [138](#)
- sagemaker.common::Processor, [4](#), [37](#), [71](#), [88](#), [98](#), [113](#), [115](#), [130](#), [138](#)
- sagemaker.common::ScriptProcessor, [4](#), [37](#), [71](#), [88](#), [98](#), [113](#), [115](#), [130](#), [138](#)
- sagemaker.mlcore::AmazonAlgorithmEstimatorBase, [24](#), [39](#), [44](#), [49](#), [53](#), [58](#), [73](#), [78](#), [83](#), [100](#)
- sagemaker.mlcore::EstimatorBase, [18](#), [24](#), [30](#), [39](#), [44](#), [49](#), [53](#), [58](#), [66](#), [73](#), [78](#), [83](#), [91](#), [100](#), [104](#), [108](#), [120](#), [132](#)

- sagemaker.mlcore::Framework, [18](#), [30](#), [66](#),  
[91](#), [104](#), [108](#), [120](#), [132](#)
- sagemaker.mlcore::FrameworkModel, [21](#),  
[33](#), [68](#), [94](#), [110](#), [124](#), [135](#)
- sagemaker.mlcore::Model, [21](#), [28](#), [33](#), [42](#),  
[47](#), [52](#), [56](#), [64](#), [68](#), [76](#), [82](#), [86](#), [94](#),  
[102](#), [110](#), [118](#), [124](#), [135](#)
- sagemaker.mlcore::ModelBase, [21](#), [28](#), [33](#),  
[42](#), [47](#), [52](#), [56](#), [64](#), [68](#), [76](#), [82](#), [86](#), [94](#),  
[102](#), [110](#), [118](#), [124](#), [135](#)
- sagemaker.mlcore::Predictor, [23](#), [29](#), [36](#),  
[43](#), [48](#), [52](#), [57](#), [65](#), [70](#), [77](#), [87](#), [97](#),  
[103](#), [112](#), [119](#), [128](#), [137](#)
- sagemaker.mlcore::PredictorBase, [23](#), [29](#),  
[36](#), [43](#), [48](#), [52](#), [57](#), [65](#), [70](#), [77](#), [87](#), [97](#),  
[103](#), [112](#), [119](#), [128](#), [137](#)
- sagemaker.mlframework  
    (sagemaker.mlframework-package),  
    [3](#)
- sagemaker.mlframework-package, [3](#)
- sagemaker.mlframework::SparkProcessorBase,  
    [88](#), [115](#)
- SKLearn, [108](#)
- SKLearnModel, [110](#)
- SKLearnPredictor, [112](#)
- SKLearnProcessor, [113](#)
- SparkJarProcessor, [115](#)
- SparkMLModel, [118](#)
- SparkMLPredictor, [119](#)
  
- TensorFlow, [120](#)
- TensorFlowModel, [124](#)
- TensorFlowPredictor, [128](#)
- TensorFlowProcessor, [130](#)
  
- XGBoost, [132](#)
- XGBoostModel, [135](#)
- XGBoostPredictor, [137](#)
- XGBoostProcessor, [138](#)