tf.estimator.EstimatorSpec

Contents

Class **`EstimatorSpec`**

Defined in `tensorflow/python/estimator/model_fn.py`.

Ops and objects returned from a `model_fn` and passed to an `Estimator`.

`EstimatorSpec` fully defines the model to be run by an `Estimator`.

Properties

`eval_metric_ops`

Alias for field number 4

`evaluation_hooks`

Alias for field number 9

`export_outputs`

Alias for field number 5

`loss`

Alias for field number 2

`mode`

Alias for field number 0

`predictions`

Alias for field number 1

`scaffold`

Alias for field number 8

`train_op`

Alias for field number 3

`training_chief_hooks`

Alias for field number 6

`training_hooks`

Alias for field number 7

Methods

`__new__`

```
@staticmethod
__new__(
    cls,
    mode,
    predictions=None,
    loss=None,
    train_op=None,
    eval_metric_ops=None,
    export_outputs=None,
    training_chief_hooks=None,
    training_hooks=None,
    scaffold=None,
    evaluation_hooks=None
)
```

Creates a validated `EstimatorSpec` instance.

Depending on the value of `mode`, different arguments are required. Namely

- For `mode == ModeKeys.TRAIN`: required fields are `loss` and `train_op`.
- For `mode == ModeKeys.EVAL`: required field is `loss`.
- For `mode == ModeKeys.PREDICT`: required fields are `predictions`.

model_fn can populate all arguments independent of mode. In this case, some arguments will be ignored by an `Estimator`. E.g. `train_op` will be ignored in eval and infer modes. Example:

```
def my_model_fn(mode, features, labels):
  predictions = ...
  loss = ...
  train_op = ...
  return tf.estimator.EstimatorSpec(
      mode=mode,
      predictions=predictions,
      loss=loss,
      train_op=train_op)
```

Alternatively, model_fn can just populate the arguments appropriate to the given mode. Example:

```
def my_model_fn(mode, features, labels):
  if (mode == tf.estimator.ModeKeys.TRAIN or
      mode == tf.estimator.ModeKeys.EVAL):
    loss = ...
  else:
    loss = None
  if mode == tf.estimator.ModeKeys.TRAIN:
    train_op = ...
  else:
    train_op = None
  if mode == tf.estimator.ModeKeys.PREDICT:
    predictions = ...
  else:
    predictions = None

  return tf.estimator.EstimatorSpec(
      mode=mode,
      predictions=predictions,
      loss=loss,
      train_op=train_op)
```

Args:

- `mode`: A `ModeKeys`. Specifies if this is training, evaluation or prediction.
- `predictions`: Predictions `Tensor` or dict of `Tensor`.
- `loss`: Training loss `Tensor`. Must be either scalar, or with shape `[1]`.
- `train_op`: Op for the training step.
- `eval_metric_ops`: Dict of metric results keyed by name. The values of the dict are the results of calling a metric function, namely a `(metric_tensor, update_op)` tuple. `metric_tensor` should be evaluated without any impact on state (typically is a pure computation results based on variables.). For example, it should not trigger the `update_op` or requires any input fetching.
- `export_outputs`: Describes the output signatures to be exported to `SavedModel` and used during serving. A dict `{name: output}` where:
  - name: An arbitrary name for this output.
  - output: an `ExportOutput` object such as `ClassificationOutput`, `RegressionOutput`, or `PredictOutput`. Single-headed models only need to specify one entry in this dictionary. Multi-headed models should specify one entry for each head, one of which must be named using signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY.
- `training_chief_hooks`: Iterable of `tf.train.SessionRunHook` objects to run on the chief worker during training.
- `training_hooks`: Iterable of `tf.train.SessionRunHook` objects to run on all workers during training.
- `scaffold`: A `tf.train.Scaffold` object that can be used to set initialization, saver, and more to be used in training.
- `evaluation_hooks`: Iterable of `tf.train.SessionRunHook` objects to run during evaluation.

Returns:

A validated `EstimatorSpec` object.

Raises:

- `ValueError`: If validation fails.
- `TypeError`: If any of the arguments is not the expected type.