tf.keras.metrics.AUC

See Stable See Nightly

TensorFlow 1 version (/versions/r1.15/api_docs/python/tf/keras/metrics/AUC) View source on GitHub (https://github.com/tensorflow/tensorflow/blob/v2.5.0/tensorflow/python/keras/metrics.py#L1940-L2410)

Approximates the AUC (Area under the curve) of the ROC or PR curves.

Inherits From: `Metric` (https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Metric), `Layer` (https://www.tensorflow.org/api_docs/python/tf/keras/layers/Layer), `Module` (https://www.tensorflow.org/api_docs/python/tf/Module)

View aliases

Main aliases

`tf.metrics.AUC` (https://www.tensorflow.org/api_docs/python/tf/keras/metrics/AUC)

Compat aliases for migration

See Migration guide (https://www.tensorflow.org/guide/migrate) for more details.

`tf.compat.v1.keras.metrics.AUC` (https://www.tensorflow.org/api_docs/python/tf/keras/metrics/AUC)

```
tf.keras.metrics.AUC(
    num_thresholds=200, curve='ROC',
    summation_method='interpolation', name=None, dtype=None,
    thresholds=None, multi_label=False, num_labels=None, label_weights=None,
    from_logits=False
)
```

Used in the notebooks

Used in the tutorials
• Classification on imbalanced data (https://www.tensorflow.org/tutorials/structured_data/imbalanced_data) • TF Lattice Premade Models (https://www.tensorflow.org/lattice/tutorials/premade_models) • Client-efficient large-model federated learning via `federated_select` and sparse aggregation (https://www.tensorflow.org/federated/tutorials/sparse_federated_learning)

The AUC (Area under the curve) of the ROC (Receiver operating characteristic; default) or PR (Precision Recall) curves are quality measures of binary classifiers. Unlike the accuracy, and like cross-entropy losses, ROC-AUC and PR-AUC evaluate all the operational points of a model.

This classes approximates AUCs using a Riemann sum: During the metric accumulation phrase, predictions are accumulated within predefined buckets by value. The AUC is then computed by interpolating per-bucket averages. These buckets define the evaluated operational points.

This metric creates four local variables, `true_positives`, `true_negatives`, `false_positives` and `false_negatives` that are used to compute the AUC. To discretize the AUC curve, a linearly spaced set of thresholds is used to compute pairs of recall and precision values. The area under the ROC-curve is therefore computed using the height of the recall values by the false positive rate, while the area under the PR-curve is the computed using the height of the precision values by the recall.

This value is ultimately returned as `auc`, an idempotent operation that computes the area under a discretized curve of precision versus recall values (computed using the aforementioned variables). The `num_thresholds` variable controls the degree of discretization with larger numbers of thresholds more closely approximating the true AUC. The quality of the approximation may vary dramatically depending on `num_thresholds`. The `thresholds` parameter can be used to manually specify thresholds which split the predictions more evenly.

For a best approximation of the real AUC, `predictions` should be distributed approximately uniformly in the range 0, 1 (/api_docs/python/tf/keras/metrics/if%20%60from_logits=False%60). The quality of the AUC approximation may be poor if this is not the case. Setting `summation_method` to 'minoring' or 'majoring' can help quantify the error in the approximation by providing lower or upper bound estimate of the AUC.

If `sample_weight` is `None`, weights default to 1. Use `sample_weight` of 0 to mask values.

Args	
`num_thresholds`	(Optional) Defaults to 200. The number of thresholds to use when discretizing the roc curve. Values must be > 1.
`curve`	(Optional) Specifies the name of the curve to be computed, 'ROC' [default] or 'PR' for the Precision-Recall-curve.
`summation_method`	(Optional) Specifies the Riemann summation method (https://en.wikipedia.org/wiki/Riemann_sum) used. 'interpolation' (default) applies mid-point summation scheme for `ROC`. For PR-AUC, interpolates (true/false) positives but not the ratio that is precision (see Davis & Goadrich 2006 for details); 'minoring' applies left summation for increasing intervals and right summation for decreasing intervals; 'majoring' does the opposite.
`name`	(Optional) string name of the metric instance.
`dtype`	(Optional) data type of the metric result.
`thresholds`	(Optional) A list of floating point values to use as the thresholds for discretizing the curve. If set, the `num_thresholds` parameter is ignored. Values should be in [0, 1]. Endpoint thresholds equal to {-epsilon, 1+epsilon} for a small positive epsilon value will be automatically included with these to correctly handle predictions equal to exactly 0 or 1.
`multi_label`	boolean indicating whether multilabel data should be treated as such, wherein AUC is computed separately for each label and then averaged across labels, or (when False) if the data should be flattened into a single label before AUC computation. In the latter case, when multilabel data is passed to AUC, each label-prediction pair is treated as an individual data point. Should be set to False for multi-class data.

`num_labels`	(Optional) The number of labels, used when `multi_label' is True. Ifnum_labelsis not specified, then state variables get created on the first call toupdate_state. </td> </tr><tr> <td>label_weights</td> <td> (Optional) list, array, or tensor of non-negative weights used to compute AUCs for multilabel data. Whenmulti_labelis True, the weights are applied to the individual label AUCs when they are averaged to produce the multi-label AUC. When it's False, they are used to weight the individual label predictions in computing the confusion matrix on the flattened data. Note that this is unlike class_weights in that class_weights weights the example depending on the value of its label, whereas label_weights depends only on the index of that label before flattening; thereforelabel_weightsshould not be used for multi-class data. </td> </tr><tr> <td>from_logits</td> <td> boolean indicating whether the predictions (y_predinupdate_state) are probabilities or sigmoid logits. As a rule of thumb, when using a keras loss, thefrom_logitsconstructor argument of the loss should match the AUCfrom_logits` constructor argument.

Standalone usage:

```
>>> m = tf.keras.metrics.AUC(num_thresholds=3)
>>> m.update_state([0, 0, 1, 1], [0, 0.5, 0.3, 0.9])
>>> # threshold values are [0 - 1e-7, 0.5, 1 + 1e-7]
>>> # tp = [2, 1, 0], fp = [2, 0, 0], fn = [0, 1, 2], tn = [0, 2, 2]
>>> # recall = [1, 0.5, 0], fp_rate = [1, 0, 0]
>>> # auc = ((((1+0.5)/2)*(1-0))+ (((0.5+0)/2)*(0-0))) = 0.75
>>> m.result().numpy()
0.75
```

```
>>> m.reset_state()
>>> m.update_state([0, 0, 1, 1], [0, 0.5, 0.3, 0.9],
...                sample_weight=[1, 0, 0, 1])
>>> m.result().numpy()
1.0
```

Usage with `compile()` API:

```
# Reports the AUC of a model outputing a probability.
model.compile(optimizer='sgd',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.AUC()])

# Reports the AUC of a model outputing a logit.
model.compile(optimizer='sgd',
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=[tf.keras.metrics.AUC(from_logits=True)])
```

Attributes	
`thresholds`	The thresholds used for evaluating AUC.

Methods

`interpolate_pr_auc`

View source (https://github.com/tensorflow/tensorflow/blob/v2.5.0/tensorflow/python/keras/metrics.py#L2250-L2329)

```
interpolate_pr_auc()
```

Interpolation formula inspired by section 4 of Davis & Goadrich 2006.

https://www.biostat.wisc.edu/~page/rocpr.pdf (https://www.biostat.wisc.edu/~page/rocpr.pdf)

Note here we derive & use a closed formula not present in the paper as follows:

Precision = TP / (TP + FP) = TP / P

Modeling all of TP (true positive), FP (false positive) and their sum P = TP + FP (predicted positive) as varying linearly within each interval [A, B] between successive thresholds, we get

Precision slope = dTP / dP = (TP_B - TP_A) / (P_B - P_A) = (TP - TP_A) / (P - P_A) Precision = (TP_A + slope * (P - P_A)) / P

The area within the interval is (slope / total_pos_weight) times

int_A^B{Precision.dP} = int_A^B{(TP_A + slope * (P - P_A)) * dP / P} int_A^B{Precision.dP} = int_A^B{slope * dP + intercept * dP / P}

where intercept = TP_A - slope * P_A = TP_B - slope * P_B, resulting in

int_A^B{Precision.dP} = TP_B - TP_A + intercept * log(P_B / P_A)

Bringing back the factor (slope / total_pos_weight) we'd put aside, we get

slope * [dTP + intercept * log(P_B / P_A)] / total_pos_weight

where dTP == TP_B - TP_A.

Note that when P_A == 0 the above calculation simplifies into

int_A^B{Precision.dTP} = int_A^B{slope * dTP} = slope * (TP_B - TP_A)

which is really equivalent to imputing constant precision throughout the first bucket having >0 true positives.

Returns	
`pr_auc`	an approximation of the area under the P-R curve.

`reset_state`

View source (https://github.com/tensorflow/tensorflow/blob/v2.5.0/tensorflow/python/keras/metrics.py#L2383-L2391)

```
reset_state()
```

Resets all of the metric state variables.

This function is called between epochs/steps, when a metric is evaluated during training.

`result`

View source (https://github.com/tensorflow/tensorflow/blob/v2.5.0/tensorflow/python/keras/metrics.py#L2331-L2381)

```
result()
```

Computes and returns the metric value tensor.

Result computation is an idempotent operation that simply calculates the metric value using the state variables.

`update_state`

View source (https://github.com/tensorflow/tensorflow/blob/v2.5.0/tensorflow/python/keras/metrics.py#L2186-L2248)

```
update_state(
    y_true, y_pred, sample_weight=None
)
```

Accumulates confusion matrix statistics.

Args	
`y_true`	The ground truth values.
`y_pred`	The predicted values.
`sample_weight`	Optional weighting of each example. Defaults to 1. Can be a `Tensor` whose rank is either 0, or the same rank as `y_true`, and must be broadcastable to `y_true`.

Returns
Update op.