"""

Module containing extra tensorflow metrics used at Twitter.

This module conforms to conventions used by tf.metrics.\*.

In particular, each metric constructs two subgraphs: value\_op and update\_op:

- The value op is used to fetch the current metric value.

- The update\_op is used to accumulate into the metric.

Note: similar to tf.metrics.\*, metrics in here do not support multi-label learning.

We will have to write wrapper classes to create one metric per label.

Note: similar to tf.metrics.\*, batches added into a metric via its update\_op are cumulative!

"""

from collections import OrderedDict

import tensorflow.compat.v1 as tf

from twml.metrics import get\_multi\_binary\_class\_metric\_fn

# checkstyle: noqa

def get\_partial\_multi\_binary\_class\_metric\_fn(metrics, classes=None, class\_dim=1, predcols=None):

def get\_eval\_metric\_ops(graph\_output, labels, weights):

if predcols is None:

preds = graph\_output['output']

else:

if isinstance(predcols, int):

predcol\_list=[predcols]

else:

predcol\_list=list(predcols)

for col in predcol\_list:

assert 0 <= col < graph\_output['output'].shape[class\_dim], 'Invalid Prediction Column Index !'

preds = tf.gather(graph\_output['output'], indices=predcol\_list, axis=class\_dim) # [batchSz, num\_col]

labels = tf.gather(labels, indices=predcol\_list, axis=class\_dim) # [batchSz, num\_col]

predInfo = {'output': preds}

if 'threshold' in graph\_output:

predInfo['threshold'] = graph\_output['threshold']

if 'hard\_output' in graph\_output:

predInfo['hard\_output'] = graph\_output['hard\_output']

metrics\_op = get\_multi\_binary\_class\_metric\_fn(metrics, classes, class\_dim)

metrics\_op\_res = metrics\_op(predInfo, labels, weights)

return metrics\_op\_res

return get\_eval\_metric\_ops

# Numeric Prediction Performance among TopK Predictions

def mean\_numeric\_label\_topK(labels, predictions, weights, name, topK\_id):

top\_k\_labels = tf.gather(params=labels, indices=topK\_id, axis=0) # [topK, 1]

return tf.metrics.mean(values=top\_k\_labels, name=name)

def mean\_gated\_numeric\_label\_topK(labels, predictions, weights, name, topK\_id, bar=2.0):

assert isinstance(bar, int) or isinstance(bar, float), "bar must be int or float"

top\_k\_labels = tf.gather(params=labels, indices=topK\_id, axis=0) # [topK, 1]

gated\_top\_k\_labels = tf.cast(top\_k\_labels > bar\*1.0, tf.int32)

return tf.metrics.mean(values=gated\_top\_k\_labels, name=name)

SUPPORTED\_NUMERIC\_METRICS = {

'mean\_numeric\_label\_topk': mean\_numeric\_label\_topK,

'mean\_gated\_numeric\_label\_topk': mean\_gated\_numeric\_label\_topK

}

DEFAULT\_NUMERIC\_METRICS = ['mean\_numeric\_label\_topk', 'mean\_gated\_numeric\_label\_topk']

def get\_metric\_topK\_fn\_helper(targetMetrics, supportedMetrics\_op, metrics=None, topK=(5,5,5), predcol=None, labelcol=None):

"""

:param targetMetrics: Target Metric List

:param supportedMetrics\_op: Supported Metric Operators Dict

:param metrics: Metric Set to evaluate

:param topK: (topK\_min, topK\_max, topK\_delta) Tuple

:param predcol: Prediction Column Index

:param labelcol: Label Column Index

:return:

"""

# pylint: disable=dict-keys-not-iterating

if targetMetrics is None or supportedMetrics\_op is None:

raise ValueError("Invalid Target Metric List/op !")

targetMetrics = set([m.lower() for m in targetMetrics])

if metrics is None:

metrics = list(targetMetrics)

else:

metrics = [m.lower() for m in metrics if m.lower() in targetMetrics]

num\_k = int((topK[1]-topK[0])/topK[2]+1)

topK\_list = [topK[0]+d\*topK[2] for d in range(num\_k)]

if 1 not in topK\_list:

topK\_list = [1] + topK\_list

def get\_eval\_metric\_ops(graph\_output, labels, weights):

"""

graph\_output:

dict that is returned by build\_graph given input features.

labels:

target labels associated to batch.

weights:

weights of the samples..

"""

eval\_metric\_ops = OrderedDict()

if predcol is None:

pred = graph\_output['output']

else:

assert 0 <= predcol < graph\_output['output'].shape[1], 'Invalid Prediction Column Index !'

assert labelcol is not None

pred = tf.reshape(graph\_output['output'][:, predcol], shape=[-1, 1])

labels = tf.reshape(labels[:, labelcol], shape=[-1, 1])

numOut = graph\_output['output'].shape[1]

pred\_score = tf.reshape(graph\_output['output'][:, numOut-1], shape=[-1, 1])

# add metrics to eval\_metric\_ops dict

for metric\_name in metrics:

metric\_name = metric\_name.lower() # metric name are case insensitive.

if metric\_name in supportedMetrics\_op:

metric\_factory = supportedMetrics\_op.get(metric\_name)

if 'topk' not in metric\_name:

value\_op, update\_op = metric\_factory(

labels=labels,

predictions=pred,

weights=weights,

name=metric\_name)

eval\_metric\_ops[metric\_name] = (value\_op, update\_op)

else:

for K in topK\_list:

K\_min = tf.minimum(K, tf.shape(pred\_score)[0])

topK\_id = tf.nn.top\_k(tf.reshape(pred\_score, shape=[-1]), k=K\_min)[1] # [topK]

value\_op, update\_op = metric\_factory(

labels=labels,

predictions=pred,

weights=weights,

name=metric\_name+'\_\_k\_'+str(K),

topK\_id=topK\_id)

eval\_metric\_ops[metric\_name+'\_\_k\_'+str(K)] = (value\_op, update\_op)

else:

raise ValueError('Cannot find the metric named ' + metric\_name)

return eval\_metric\_ops

return get\_eval\_metric\_ops

def get\_numeric\_metric\_fn(metrics=None, topK=(5,5,5), predcol=None, labelcol=None):

if metrics is None:

metrics = list(DEFAULT\_NUMERIC\_METRICS)

metrics = list(set(metrics))

metric\_op = get\_metric\_topK\_fn\_helper(targetMetrics=list(DEFAULT\_NUMERIC\_METRICS),

supportedMetrics\_op=SUPPORTED\_NUMERIC\_METRICS,

metrics=metrics, topK=topK, predcol=predcol, labelcol=labelcol)

return metric\_op

def get\_single\_binary\_task\_metric\_fn(metrics, classnames, topK=(5,5,5), use\_topK=False):

"""

graph\_output['output']: [BatchSz, 1] [pred\_Task1]

labels: [BatchSz, 2] [Task1, NumericLabel]

"""

def get\_eval\_metric\_ops(graph\_output, labels, weights):

metric\_op\_base = get\_partial\_multi\_binary\_class\_metric\_fn(metrics, predcols=0, classes=classnames)

classnames\_unw = ['unweighted\_'+cs for cs in classnames]

metric\_op\_unw = get\_partial\_multi\_binary\_class\_metric\_fn(metrics, predcols=0, classes=classnames\_unw)

metrics\_base\_res = metric\_op\_base(graph\_output, labels, weights)

metrics\_unw\_res = metric\_op\_unw(graph\_output, labels, None)

metrics\_base\_res.update(metrics\_unw\_res)

if use\_topK:

metric\_op\_numeric = get\_numeric\_metric\_fn(metrics=None, topK=topK, predcol=0, labelcol=1)

metrics\_numeric\_res = metric\_op\_numeric(graph\_output, labels, weights)

metrics\_base\_res.update(metrics\_numeric\_res)

return metrics\_base\_res

return get\_eval\_metric\_ops

def get\_dual\_binary\_tasks\_metric\_fn(metrics, classnames, topK=(5,5,5), use\_topK=False):

"""

graph\_output['output']: [BatchSz, 3] [pred\_Task1, pred\_Task2, Score]

labels: [BatchSz, 3] [Task1, Task2, NumericLabel]

"""

def get\_eval\_metric\_ops(graph\_output, labels, weights):

metric\_op\_base = get\_partial\_multi\_binary\_class\_metric\_fn(metrics, predcols=[0, 1], classes=classnames)

classnames\_unw = ['unweighted\_'+cs for cs in classnames]

metric\_op\_unw = get\_partial\_multi\_binary\_class\_metric\_fn(metrics, predcols=[0, 1], classes=classnames\_unw)

metrics\_base\_res = metric\_op\_base(graph\_output, labels, weights)

metrics\_unw\_res = metric\_op\_unw(graph\_output, labels, None)

metrics\_base\_res.update(metrics\_unw\_res)

if use\_topK:

metric\_op\_numeric = get\_numeric\_metric\_fn(metrics=None, topK=topK, predcol=2, labelcol=2)

metrics\_numeric\_res = metric\_op\_numeric(graph\_output, labels, weights)

metrics\_base\_res.update(metrics\_numeric\_res)

return metrics\_base\_res

return get\_eval\_metric\_ops