"""

Utilties for constructing a metric\_fn for magic recs.

"""

from twml.contrib.metrics.metrics import (

get\_dual\_binary\_tasks\_metric\_fn,

get\_numeric\_metric\_fn,

get\_partial\_multi\_binary\_class\_metric\_fn,

get\_single\_binary\_task\_metric\_fn,

)

from .model\_utils import generate\_disliked\_mask

import tensorflow.compat.v1 as tf

METRIC\_BOOK = {

"OONC": ["OONC"],

"OONC\_Engagement": ["OONC", "Engagement"],

"Sent": ["Sent"],

"HeavyRankPosition": ["HeavyRankPosition"],

"HeavyRankProbability": ["HeavyRankProbability"],

}

USER\_AGE\_FEATURE\_NAME = "accountAge"

NEW\_USER\_AGE\_CUTOFF = 0

def remove\_padding\_and\_flatten(tensor, valid\_batch\_size):

"""Remove the padding of the input padded tensor given the valid batch size tensor,

then flatten the output with respect to the first dimension.

Args:

tensor: A tensor of size [META\_BATCH\_SIZE, BATCH\_SIZE, FEATURE\_DIM].

valid\_batch\_size: A tensor of size [META\_BATCH\_SIZE], with each element indicating

the effective batch size of the BATCH\_SIZE dimension.

Returns:

A tesnor of size [tf.reduce\_sum(valid\_batch\_size), FEATURE\_DIM].

"""

unpadded\_ragged\_tensor = tf.RaggedTensor.from\_tensor(tensor=tensor, lengths=valid\_batch\_size)

return unpadded\_ragged\_tensor.flat\_values

def safe\_mask(values, mask):

"""Mask values if possible.

Boolean mask inputed values if and only if values is a tensor of the same dimension as mask (or can be broadcasted to that dimension).

Args:

values (Any or Tensor): Input tensor to mask. Dim 0 should be size N.

mask (boolean tensor): A boolean tensor of size N.

Returns Values or Values masked.

"""

if values is None:

return values

if not tf.is\_tensor(values):

return values

values\_shape = values.get\_shape()

if not values\_shape or len(values\_shape) == 0:

return values

if not mask.get\_shape().is\_compatible\_with(values\_shape[0]):

return values

return tf.boolean\_mask(values, mask)

def add\_new\_user\_metrics(metric\_fn):

"""Will stratify the metric\_fn by adding new user metrics.

Given an input metric\_fn, double every metric: One will be the orignal and the other will only include those for new users.

Args:

metric\_fn (python function): Base twml metric\_fn.

Returns a metric\_fn with new user metrics included.

"""

def metric\_fn\_with\_new\_users(graph\_output, labels, weights):

if USER\_AGE\_FEATURE\_NAME not in graph\_output:

raise ValueError(

"In order to get metrics stratified by user age, {name} feature should be added to model graph output. However, only the following output keys were found: {keys}.".format(

name=USER\_AGE\_FEATURE\_NAME, keys=graph\_output.keys()

)

)

metric\_ops = metric\_fn(graph\_output, labels, weights)

is\_new = tf.reshape(

tf.math.less\_equal(

tf.cast(graph\_output[USER\_AGE\_FEATURE\_NAME], tf.int64),

tf.cast(NEW\_USER\_AGE\_CUTOFF, tf.int64),

),

[-1],

)

labels = safe\_mask(labels, is\_new)

weights = safe\_mask(weights, is\_new)

graph\_output = {key: safe\_mask(values, is\_new) for key, values in graph\_output.items()}

new\_user\_metric\_ops = metric\_fn(graph\_output, labels, weights)

new\_user\_metric\_ops = {name + "\_new\_users": ops for name, ops in new\_user\_metric\_ops.items()}

metric\_ops.update(new\_user\_metric\_ops)

return metric\_ops

return metric\_fn\_with\_new\_users

def get\_meta\_learn\_single\_binary\_task\_metric\_fn(

metrics, classnames, top\_k=(5, 5, 5), use\_top\_k=False

):

"""Wrapper function to use the metric\_fn with meta learning evaluation scheme.

Args:

metrics: A list of string representing metric names.

classnames: A list of string repsenting class names, In case of multiple binary class models,

the names for each class or label.

top\_k: A tuple of int to specify top K metrics.

use\_top\_k: A boolean value indicating of top K of metrics is used.

Returns:

A customized metric\_fn function.

"""

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

"""The op func of the eval\_metrics. Comparing with normal version,

the difference is we flatten the output, label, and weights.

Args:

graph\_output: A dict of tensors.

labels: A tensor of int32 be the value of either 0 or 1.

weights: A tensor of float32 to indicate the per record weight.

Returns:

A dict of metric names and values.

"""

metric\_op\_weighted = get\_partial\_multi\_binary\_class\_metric\_fn(

metrics, predcols=0, classes=classnames

)

classnames\_unweighted = ["unweighted\_" + classname for classname in classnames]

metric\_op\_unweighted = get\_partial\_multi\_binary\_class\_metric\_fn(

metrics, predcols=0, classes=classnames\_unweighted

)

valid\_batch\_size = graph\_output["valid\_batch\_size"]

graph\_output["output"] = remove\_padding\_and\_flatten(graph\_output["output"], valid\_batch\_size)

labels = remove\_padding\_and\_flatten(labels, valid\_batch\_size)

weights = remove\_padding\_and\_flatten(weights, valid\_batch\_size)

tf.ensure\_shape(graph\_output["output"], [None, 1])

tf.ensure\_shape(labels, [None, 1])

tf.ensure\_shape(weights, [None, 1])

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

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

metrics\_weighted.update(metrics\_unweighted)

if use\_top\_k:

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

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

metrics\_weighted.update(metrics\_numeric)

return metrics\_weighted

return get\_eval\_metric\_ops

def get\_meta\_learn\_dual\_binary\_tasks\_metric\_fn(

metrics, classnames, top\_k=(5, 5, 5), use\_top\_k=False

):

"""Wrapper function to use the metric\_fn with meta learning evaluation scheme.

Args:

metrics: A list of string representing metric names.

classnames: A list of string repsenting class names, In case of multiple binary class models,

the names for each class or label.

top\_k: A tuple of int to specify top K metrics.

use\_top\_k: A boolean value indicating of top K of metrics is used.

Returns:

A customized metric\_fn function.

"""

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

"""The op func of the eval\_metrics. Comparing with normal version,

the difference is we flatten the output, label, and weights.

Args:

graph\_output: A dict of tensors.

labels: A tensor of int32 be the value of either 0 or 1.

weights: A tensor of float32 to indicate the per record weight.

Returns:

A dict of metric names and values.

"""

metric\_op\_weighted = get\_partial\_multi\_binary\_class\_metric\_fn(

metrics, predcols=[0, 1], classes=classnames

)

classnames\_unweighted = ["unweighted\_" + classname for classname in classnames]

metric\_op\_unweighted = get\_partial\_multi\_binary\_class\_metric\_fn(

metrics, predcols=[0, 1], classes=classnames\_unweighted

)

valid\_batch\_size = graph\_output["valid\_batch\_size"]

graph\_output["output"] = remove\_padding\_and\_flatten(graph\_output["output"], valid\_batch\_size)

labels = remove\_padding\_and\_flatten(labels, valid\_batch\_size)

weights = remove\_padding\_and\_flatten(weights, valid\_batch\_size)

tf.ensure\_shape(graph\_output["output"], [None, 2])

tf.ensure\_shape(labels, [None, 2])

tf.ensure\_shape(weights, [None, 1])

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

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

metrics\_weighted.update(metrics\_unweighted)

if use\_top\_k:

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

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

metrics\_weighted.update(metrics\_numeric)

return metrics\_weighted

return get\_eval\_metric\_ops

def get\_metric\_fn(task\_name, use\_stratify\_metrics, use\_meta\_batch=False):

"""Will retrieve the metric\_fn for magic recs.

Args:

task\_name (string): Which task is being used for this model.

use\_stratify\_metrics (boolean): Should we add stratified metrics (new user metrics).

use\_meta\_batch (boolean): If the output/label/weights are passed in 3D shape instead of

2D shape.

Returns:

A metric\_fn function to pass in twml Trainer.

"""

if task\_name not in METRIC\_BOOK:

raise ValueError(

"Task name of {task\_name} not recognized. Unable to retrieve metrics.".format(

task\_name=task\_name

)

)

class\_names = METRIC\_BOOK[task\_name]

if use\_meta\_batch:

get\_n\_binary\_task\_metric\_fn = (

get\_meta\_learn\_single\_binary\_task\_metric\_fn

if len(class\_names) == 1

else get\_meta\_learn\_dual\_binary\_tasks\_metric\_fn

)

else:

get\_n\_binary\_task\_metric\_fn = (

get\_single\_binary\_task\_metric\_fn if len(class\_names) == 1 else get\_dual\_binary\_tasks\_metric\_fn

)

metric\_fn = get\_n\_binary\_task\_metric\_fn(metrics=None, classnames=METRIC\_BOOK[task\_name])

if use\_stratify\_metrics:

metric\_fn = add\_new\_user\_metrics(metric\_fn)

return metric\_fn

def flip\_disliked\_labels(metric\_fn):

"""This function returns an adapted metric\_fn which flips the labels of the OONCed evaluation data to 0 if it is disliked.

Args:

metric\_fn: A metric\_fn function to pass in twml Trainer.

Returns:

\_adapted\_metric\_fn: A customized metric\_fn function with disliked OONC labels flipped.

"""

def \_adapted\_metric\_fn(graph\_output, labels, weights):

"""A customized metric\_fn function with disliked OONC labels flipped.

Args:

graph\_output: A dict of tensors.

labels: labels of training samples, which is a 2D tensor of shape batch\_size x 3: [OONCs, engagements, dislikes]

weights: A tensor of float32 to indicate the per record weight.

Returns:

A dict of metric names and values.

"""

# We want to multiply the label of the observation by 0 only when it is disliked

disliked\_mask = generate\_disliked\_mask(labels)

# Extract OONC and engagement labels only.

labels = tf.reshape(labels[:, 0:2], shape=[-1, 2])

# Labels will be set to 0 if it is disliked.

adapted\_labels = labels \* tf.cast(tf.logical\_not(disliked\_mask), dtype=labels.dtype)

return metric\_fn(graph\_output, adapted\_labels, weights)

return \_adapted\_metric\_fn