"""

Model for modifying the checkpoints of the magic recs cnn Model with addition, deletion, and reordering

of continuous and binary features.

"""

import os

from twitter.deepbird.projects.magic\_recs.libs.get\_feat\_config import FEATURE\_LIST\_DEFAULT\_PATH

from twitter.deepbird.projects.magic\_recs.libs.warm\_start\_utils\_v11 import (

get\_feature\_list\_for\_heavy\_ranking,

mkdirp,

rename\_dir,

rmdir,

warm\_start\_checkpoint,

)

import twml

from twml.trainers import DataRecordTrainer

import tensorflow.compat.v1 as tf

from tensorflow.compat.v1 import logging

def get\_arg\_parser():

parser = DataRecordTrainer.add\_parser\_arguments()

parser.add\_argument(

"--model\_type",

default="deepnorm\_gbdt\_inputdrop2\_rescale",

type=str,

help="specify the model type to use.",

)

parser.add\_argument(

"--model\_trainer\_name",

default="None",

type=str,

help="deprecated, added here just for api compatibility.",

)

parser.add\_argument(

"--warm\_start\_base\_dir",

default="none",

type=str,

help="latest ckpt in this folder will be used.",

)

parser.add\_argument(

"--output\_checkpoint\_dir",

default="none",

type=str,

help="Output folder for warm started ckpt. If none, it will move warm\_start\_base\_dir to backup, and overwrite it",

)

parser.add\_argument(

"--feature\_list",

default="none",

type=str,

help="Which features to use for training",

)

parser.add\_argument(

"--old\_feature\_list",

default="none",

type=str,

help="Which features to use for training",

)

return parser

def get\_params(args=None):

parser = get\_arg\_parser()

if args is None:

return parser.parse\_args()

else:

return parser.parse\_args(args)

def \_main():

opt = get\_params()

logging.info("parse is: ")

logging.info(opt)

if opt.feature\_list == "none":

feature\_list\_path = FEATURE\_LIST\_DEFAULT\_PATH

else:

feature\_list\_path = opt.feature\_list

if opt.warm\_start\_base\_dir != "none" and tf.io.gfile.exists(opt.warm\_start\_base\_dir):

if opt.output\_checkpoint\_dir == "none" or opt.output\_checkpoint\_dir == opt.warm\_start\_base\_dir:

\_warm\_start\_base\_dir = os.path.normpath(opt.warm\_start\_base\_dir) + "\_backup\_warm\_start"

\_output\_folder\_dir = opt.warm\_start\_base\_dir

rename\_dir(opt.warm\_start\_base\_dir, \_warm\_start\_base\_dir)

tf.logging.info(f"moved {opt.warm\_start\_base\_dir} to {\_warm\_start\_base\_dir}")

else:

\_warm\_start\_base\_dir = opt.warm\_start\_base\_dir

\_output\_folder\_dir = opt.output\_checkpoint\_dir

continuous\_binary\_feat\_list\_save\_path = os.path.join(

\_warm\_start\_base\_dir, "continuous\_binary\_feat\_list.json"

)

if opt.old\_feature\_list != "none":

tf.logging.info("getting old continuous\_binary\_feat\_list")

continuous\_binary\_feat\_list = get\_feature\_list\_for\_heavy\_ranking(

opt.old\_feature\_list, opt.data\_spec

)

rmdir(continuous\_binary\_feat\_list\_save\_path)

twml.util.write\_file(

continuous\_binary\_feat\_list\_save\_path, continuous\_binary\_feat\_list, encode="json"

)

tf.logging.info(f"Finish writting files to {continuous\_binary\_feat\_list\_save\_path}")

warm\_start\_folder = os.path.join(\_warm\_start\_base\_dir, "best\_checkpoint")

if not tf.io.gfile.exists(warm\_start\_folder):

warm\_start\_folder = \_warm\_start\_base\_dir

rmdir(\_output\_folder\_dir)

mkdirp(\_output\_folder\_dir)

new\_ckpt = warm\_start\_checkpoint(

warm\_start\_folder,

continuous\_binary\_feat\_list\_save\_path,

feature\_list\_path,

opt.data\_spec,

\_output\_folder\_dir,

opt.model\_type,

)

logging.info(f"Created new ckpt {new\_ckpt} from {warm\_start\_folder}")

tf.logging.info("getting new continuous\_binary\_feat\_list")

new\_continuous\_binary\_feat\_list\_save\_path = os.path.join(

\_output\_folder\_dir, "continuous\_binary\_feat\_list.json"

)

continuous\_binary\_feat\_list = get\_feature\_list\_for\_heavy\_ranking(

feature\_list\_path, opt.data\_spec

)

rmdir(new\_continuous\_binary\_feat\_list\_save\_path)

twml.util.write\_file(

new\_continuous\_binary\_feat\_list\_save\_path, continuous\_binary\_feat\_list, encode="json"

)

tf.logging.info(f"Finish writting files to {new\_continuous\_binary\_feat\_list\_save\_path}")

if \_\_name\_\_ == "\_\_main\_\_":

\_main()