from twml.trainers import DataRecordTrainer

# checkstyle: noqa

def get\_arg\_parser():

parser = DataRecordTrainer.add\_parser\_arguments()

parser.add\_argument(

"--input\_size\_bits",

type=int,

default=18,

help="number of bits allocated to the input size",

)

parser.add\_argument(

"--model\_trainer\_name",

default="magic\_recs\_mlp\_calibration\_MTL\_OONC\_Engagement",

type=str,

help="specify the model trainer name.",

)

parser.add\_argument(

"--model\_type",

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

type=str,

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

)

parser.add\_argument(

"--feat\_config\_type",

default="get\_feature\_config\_with\_sparse\_continuous",

type=str,

help="specify the feature configure function to use.",

)

parser.add\_argument(

"--directly\_export\_best",

default=False,

action="store\_true",

help="whether to directly\_export best\_checkpoint",

)

parser.add\_argument(

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

default="none",

type=str,

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

)

parser.add\_argument(

"--feature\_list",

default="none",

type=str,

help="Which features to use for training",

)

parser.add\_argument(

"--warm\_start\_from", default=None, type=str, help="model dir to warm start from"

)

parser.add\_argument(

"--momentum", default=0.99999, type=float, help="Momentum term for batch normalization"

)

parser.add\_argument(

"--dropout",

default=0.2,

type=float,

help="input\_dropout\_rate to rescale output by (1 - input\_dropout\_rate)",

)

parser.add\_argument(

"--out\_layer\_1\_size", default=256, type=int, help="Size of MLP\_branch layer 1"

)

parser.add\_argument(

"--out\_layer\_2\_size", default=128, type=int, help="Size of MLP\_branch layer 2"

)

parser.add\_argument("--out\_layer\_3\_size", default=64, type=int, help="Size of MLP\_branch layer 3")

parser.add\_argument(

"--sparse\_embedding\_size", default=50, type=int, help="Dimensionality of sparse embedding layer"

)

parser.add\_argument(

"--dense\_embedding\_size", default=128, type=int, help="Dimensionality of dense embedding layer"

)

parser.add\_argument(

"--use\_uam\_label",

default=False,

type=str,

help="Whether to use uam\_label or not",

)

parser.add\_argument(

"--task\_name",

default="OONC\_Engagement",

type=str,

help="specify the task name to use: OONC or OONC\_Engagement.",

)

parser.add\_argument(

"--init\_weight",

default=0.9,

type=float,

help="Initial OONC Task Weight MTL: OONC+Engagement.",

)

parser.add\_argument(

"--use\_engagement\_weight",

default=False,

action="store\_true",

help="whether to use engagement weight for base model.",

)

parser.add\_argument(

"--mtl\_num\_extra\_layers",

type=int,

default=1,

help="Number of Hidden Layers for each TaskBranch.",

)

parser.add\_argument(

"--mtl\_neuron\_scale", type=int, default=4, help="Scaling Factor of Neurons in MTL Extra Layers."

)

parser.add\_argument(

"--use\_oonc\_score",

default=False,

action="store\_true",

help="whether to use oonc score only or combined score.",

)

parser.add\_argument(

"--use\_stratified\_metrics",

default=False,

action="store\_true",

help="Use stratified metrics: Break out new-user metrics.",

)

parser.add\_argument(

"--run\_group\_metrics",

default=False,

action="store\_true",

help="Will run evaluation metrics grouped by user.",

)

parser.add\_argument(

"--use\_full\_scope",

default=False,

action="store\_true",

help="Will add extra scope and naming to graph.",

)

parser.add\_argument(

"--trainable\_regexes",

default=None,

nargs="\*",

help="The union of variables specified by the list of regexes will be considered trainable.",

)

parser.add\_argument(

"--fine\_tuning.ckpt\_to\_initialize\_from",

dest="fine\_tuning\_ckpt\_to\_initialize\_from",

type=str,

default=None,

help="Checkpoint path from which to warm start. Indicates the pre-trained model.",

)

parser.add\_argument(

"--fine\_tuning.warm\_start\_scope\_regex",

dest="fine\_tuning\_warm\_start\_scope\_regex",

type=str,

default=None,

help="All variables matching this will be restored.",

)

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 get\_arg\_parser\_light\_ranking():

parser = get\_arg\_parser()

parser.add\_argument(

"--use\_record\_weight",

default=False,

action="store\_true",

help="whether to use record weight for base model.",

)

parser.add\_argument(

"--min\_record\_weight", default=0.0, type=float, help="Minimum record weight to use."

)

parser.add\_argument(

"--smooth\_weight", default=0.0, type=float, help="Factor to smooth Rank Position Weight."

)

parser.add\_argument(

"--num\_mlp\_layers", type=int, default=3, help="Number of Hidden Layers for MLP model."

)

parser.add\_argument(

"--mlp\_neuron\_scale", type=int, default=4, help="Scaling Factor of Neurons in MLP Layers."

)

parser.add\_argument(

"--run\_light\_ranking\_group\_metrics",

default=False,

action="store\_true",

help="Will run evaluation metrics grouped by user for Light Ranking.",

)

parser.add\_argument(

"--use\_missing\_sub\_branch",

default=False,

action="store\_true",

help="Whether to use missing value sub-branch for Light Ranking.",

)

parser.add\_argument(

"--use\_gbdt\_features",

default=False,

action="store\_true",

help="Whether to use GBDT features for Light Ranking.",

)

parser.add\_argument(

"--run\_light\_ranking\_group\_metrics\_in\_bq",

default=False,

action="store\_true",

help="Whether to get\_predictions for Light Ranking to compute group metrics in BigQuery.",

)

parser.add\_argument(

"--pred\_file\_path",

default=None,

type=str,

help="path",

)

parser.add\_argument(

"--pred\_file\_name",

default=None,

type=str,

help="path",

)

return parser