import os

from twitter.deepbird.projects.magic\_recs.libs.metric\_fn\_utils import USER\_AGE\_FEATURE\_NAME

from twitter.deepbird.projects.magic\_recs.libs.model\_utils import read\_config

from twml.contrib import feature\_config as contrib\_feature\_config

# checkstyle: noqa

FEAT\_CONFIG\_DEFAULT\_VAL = -1.23456789

DEFAULT\_INPUT\_SIZE\_BITS = 18

DEFAULT\_FEATURE\_LIST\_PATH = "./feature\_list\_default.yaml"

FEATURE\_LIST\_DEFAULT\_PATH = os.path.join(

os.path.dirname(os.path.realpath(\_\_file\_\_)), DEFAULT\_FEATURE\_LIST\_PATH

)

DEFAULT\_FEATURE\_LIST\_LIGHT\_RANKING\_PATH = "./feature\_list\_light\_ranking.yaml"

FEATURE\_LIST\_DEFAULT\_LIGHT\_RANKING\_PATH = os.path.join(

os.path.dirname(os.path.realpath(\_\_file\_\_)), DEFAULT\_FEATURE\_LIST\_LIGHT\_RANKING\_PATH

)

FEATURE\_LIST\_DEFAULT = read\_config(FEATURE\_LIST\_DEFAULT\_PATH).items()

FEATURE\_LIST\_LIGHT\_RANKING\_DEFAULT = read\_config(FEATURE\_LIST\_DEFAULT\_LIGHT\_RANKING\_PATH).items()

LABELS = ["label"]

LABELS\_MTL = {"OONC": ["label"], "OONC\_Engagement": ["label", "label.engagement"]}

LABELS\_LR = {

"Sent": ["label.sent"],

"HeavyRankPosition": ["meta.ranking.is\_top3"],

"HeavyRankProbability": ["meta.ranking.weighted\_oonc\_model\_score"],

}

def \_get\_new\_feature\_config\_base(

data\_spec\_path,

labels,

add\_sparse\_continous=True,

add\_gbdt=True,

add\_user\_id=False,

add\_timestamp=False,

add\_user\_age=False,

feature\_list\_provided=[],

opt=None,

run\_light\_ranking\_group\_metrics\_in\_bq=False,

):

"""

Getter of the feature config based on specification.

Args:

data\_spec\_path: A string indicating the path of the data\_spec.json file, which could be

either a local path or a hdfs path.

labels: A list of strings indicating the name of the label in the data spec.

add\_sparse\_continous: A bool indicating if sparse\_continuous feature needs to be included.

add\_gbdt: A bool indicating if gbdt feature needs to be included.

add\_user\_id: A bool indicating if user\_id feature needs to be included.

add\_timestamp: A bool indicating if timestamp feature needs to be included. This will be useful

for sequential models and meta learning models.

add\_user\_age: A bool indicating if the user age feature needs to be included.

feature\_list\_provided: A list of features thats need to be included. If not specified, will use

FEATURE\_LIST\_DEFAULT by default.

opt: A namespace of arguments indicating the hyparameters.

run\_light\_ranking\_group\_metrics\_in\_bq: A bool indicating if heavy ranker score info needs to be included to compute group metrics in BigQuery.

Returns:

A twml feature config object.

"""

input\_size\_bits = DEFAULT\_INPUT\_SIZE\_BITS if opt is None else opt.input\_size\_bits

feature\_list = feature\_list\_provided if feature\_list\_provided != [] else FEATURE\_LIST\_DEFAULT

a\_string\_feat\_list = [f[0] for f in feature\_list if f[1] != "S"]

builder = contrib\_feature\_config.FeatureConfigBuilder(data\_spec\_path=data\_spec\_path)

builder = builder.extract\_feature\_group(

feature\_regexes=a\_string\_feat\_list,

group\_name="continuous",

default\_value=FEAT\_CONFIG\_DEFAULT\_VAL,

type\_filter=["CONTINUOUS"],

)

builder = builder.extract\_features\_as\_hashed\_sparse(

feature\_regexes=a\_string\_feat\_list,

output\_tensor\_name="sparse\_no\_continuous",

hash\_space\_size\_bits=input\_size\_bits,

type\_filter=["BINARY", "DISCRETE", "STRING", "SPARSE\_BINARY"],

)

if add\_gbdt:

builder = builder.extract\_features\_as\_hashed\_sparse(

feature\_regexes=["ads\..\*"],

output\_tensor\_name="gbdt\_sparse",

hash\_space\_size\_bits=input\_size\_bits,

)

if add\_sparse\_continous:

s\_string\_feat\_list = [f[0] for f in feature\_list if f[1] == "S"]

builder = builder.extract\_features\_as\_hashed\_sparse(

feature\_regexes=s\_string\_feat\_list,

output\_tensor\_name="sparse\_continuous",

hash\_space\_size\_bits=input\_size\_bits,

type\_filter=["SPARSE\_CONTINUOUS"],

)

if add\_user\_id:

builder = builder.extract\_feature("meta.user\_id")

if add\_timestamp:

builder = builder.extract\_feature("meta.timestamp")

if add\_user\_age:

builder = builder.extract\_feature(USER\_AGE\_FEATURE\_NAME)

if run\_light\_ranking\_group\_metrics\_in\_bq:

builder = builder.extract\_feature("meta.trace\_id")

builder = builder.extract\_feature("meta.ranking.weighted\_oonc\_model\_score")

builder = builder.add\_labels(labels).define\_weight("meta.weight")

return builder.build()

def get\_feature\_config\_with\_sparse\_continuous(

data\_spec\_path,

feature\_list\_provided=[],

opt=None,

add\_user\_id=False,

add\_timestamp=False,

add\_user\_age=False,

):

task\_name = opt.task\_name if getattr(opt, "task\_name", None) is not None else "OONC"

if task\_name not in LABELS\_MTL:

raise ValueError("Invalid Task Name !")

return \_get\_new\_feature\_config\_base(

data\_spec\_path=data\_spec\_path,

labels=LABELS\_MTL[task\_name],

add\_sparse\_continous=True,

add\_user\_id=add\_user\_id,

add\_timestamp=add\_timestamp,

add\_user\_age=add\_user\_age,

feature\_list\_provided=feature\_list\_provided,

opt=opt,

)

def get\_feature\_config\_light\_ranking(

data\_spec\_path,

feature\_list\_provided=[],

opt=None,

add\_user\_id=True,

add\_timestamp=False,

add\_user\_age=False,

add\_gbdt=False,

run\_light\_ranking\_group\_metrics\_in\_bq=False,

):

task\_name = opt.task\_name if getattr(opt, "task\_name", None) is not None else "HeavyRankPosition"

if task\_name not in LABELS\_LR:

raise ValueError("Invalid Task Name !")

if not feature\_list\_provided:

feature\_list\_provided = FEATURE\_LIST\_LIGHT\_RANKING\_DEFAULT

return \_get\_new\_feature\_config\_base(

data\_spec\_path=data\_spec\_path,

labels=LABELS\_LR[task\_name],

add\_sparse\_continous=False,

add\_gbdt=add\_gbdt,

add\_user\_id=add\_user\_id,

add\_timestamp=add\_timestamp,

add\_user\_age=add\_user\_age,

feature\_list\_provided=feature\_list\_provided,

opt=opt,

run\_light\_ranking\_group\_metrics\_in\_bq=run\_light\_ranking\_group\_metrics\_in\_bq,

)