import bisect

import os

import random as python\_random

import subprocess

from toxicity\_ml\_pipeline.settings.default\_settings\_tox import LOCAL\_DIR

import numpy as np

from sklearn.metrics import precision\_recall\_curve

try:

import tensorflow as tf

except ModuleNotFoundError:

pass

def upload\_model(full\_gcs\_model\_path):

folder\_name = full\_gcs\_model\_path

if folder\_name[:5] != "gs://":

folder\_name = "gs://" + folder\_name

dirname = os.path.dirname(folder\_name)

epoch = os.path.basename(folder\_name)

model\_dir = os.path.join(LOCAL\_DIR, "models")

cmd = f"mkdir {model\_dir}"

try:

execute\_command(cmd)

except subprocess.CalledProcessError:

pass

model\_dir = os.path.join(model\_dir, os.path.basename(dirname))

cmd = f"mkdir {model\_dir}"

try:

execute\_command(cmd)

except subprocess.CalledProcessError:

pass

try:

\_ = int(epoch)

except ValueError:

cmd = f"gsutil rsync -r '{folder\_name}' {model\_dir}"

weights\_dir = model\_dir

else:

cmd = f"gsutil cp '{dirname}/checkpoint' {model\_dir}/"

execute\_command(cmd)

cmd = f"gsutil cp '{os.path.join(dirname, epoch)}\*' {model\_dir}/"

weights\_dir = f"{model\_dir}/{epoch}"

execute\_command(cmd)

return weights\_dir

def compute\_precision\_fixed\_recall(labels, preds, fixed\_recall):

precision\_values, recall\_values, thresholds = precision\_recall\_curve(y\_true=labels, probas\_pred=preds)

index\_recall = bisect.bisect\_left(-recall\_values, -1 \* fixed\_recall)

result = precision\_values[index\_recall - 1]

print(f"Precision at {recall\_values[index\_recall-1]} recall: {result}")

return result, thresholds[index\_recall - 1]

def load\_inference\_func(model\_folder):

model = tf.saved\_model.load(model\_folder, ["serve"])

inference\_func = model.signatures["serving\_default"]

return inference\_func

def execute\_query(client, query):

job = client.query(query)

df = job.result().to\_dataframe()

return df

def execute\_command(cmd, print\_=True):

s = subprocess.run(cmd, shell=True, capture\_output=print\_, check=True)

if print\_:

print(s.stderr.decode("utf-8"))

print(s.stdout.decode("utf-8"))

def check\_gpu():

try:

execute\_command("nvidia-smi")

except subprocess.CalledProcessError:

print("There is no GPU when there should be one.")

raise AttributeError

l = tf.config.list\_physical\_devices("GPU")

if len(l) == 0:

raise ModuleNotFoundError("Tensorflow has not found the GPU. Check your installation")

print(l)

def set\_seeds(seed):

np.random.seed(seed)

python\_random.seed(seed)

tf.random.set\_seed(seed)