import uuid

from tensorflow.compat.v1 import logging

import twml

import tensorflow.compat.v1 as tf

def write\_list\_to\_hdfs\_gfile(list\_to\_write, output\_path):

"""Use tensorflow gfile to write a list to a location on hdfs"""

locname = "/tmp/{}".format(str(uuid.uuid4()))

with open(locname, "w") as f:

for row in list\_to\_write:

f.write("%s\n" % row)

tf.io.gfile.copy(locname, output\_path, overwrite=False)

def decode\_str\_or\_unicode(str\_or\_unicode):

return str\_or\_unicode.decode() if hasattr(str\_or\_unicode, 'decode') else str\_or\_unicode

def longest\_common\_prefix(strings, split\_character):

"""

Args:

string (list<str>): The list of strings to find the longest common prefix of

split\_character (str): If not None, require that the return string end in this character or

be the length of the entire string

Returns:

The string corresponding to the longest common prefix

"""

sorted\_strings = sorted(strings)

s1, s2 = sorted\_strings[0], sorted\_strings[-1]

if s1 == s2:

# If the strings are the same, just return the full string

out = s1

else:

# If the strings are not the same, return the longest common prefix optionally ending in split\_character

ix = 0

for i in range(min(len(s1), len(s2))):

if s1[i] != s2[i]:

break

if split\_character is None or s1[i] == split\_character:

ix = i + 1

out = s1[:ix]

return out

def \_expand\_prefix(fname, prefix, split\_character):

if len(fname) == len(prefix):

# If the prefix is already the full feature, just take the feature name

out = fname

elif split\_character is None:

# Advance the prefix by one character

out = fname[:len(prefix) + 1]

else:

# Advance the prefix to the next instance of split\_character or the end of the string

for ix in range(len(prefix), len(fname)):

if fname[ix] == split\_character:

break

out = fname[:ix + 1]

return out

def \_get\_feature\_types\_from\_records(records, fnames):

# This method gets the types of the features in fnames by looking at the datarecords themselves.

# The reason why we do this rather than extract the feature types from the feature\_config is

# that the feature naming conventions in the feature\_config are different from those in the

# datarecords.

fids = [twml.feature\_id(fname)[0] for fname in fnames]

feature\_to\_type = {}

for record in records:

for feature\_type, values in record.\_\_dict\_\_.items():

if values is not None:

included\_ids = set(values)

for fname, fid in zip(fnames, fids):

if fid in included\_ids:

feature\_to\_type[fname] = feature\_type

return feature\_to\_type

def \_get\_metrics\_hook(trainer):

def get\_metrics\_fn(trainer=trainer):

return {k: v[0]for k, v in trainer.current\_estimator\_spec.eval\_metric\_ops.items()}

return twml.hooks.GetMetricsHook(get\_metrics\_fn=get\_metrics\_fn)

def \_get\_feature\_name\_from\_config(feature\_config):

"""Extract the names of the features on a feature config object

"""

decoded\_feature\_names = []

for f in feature\_config.get\_feature\_spec()['features'].values():

try:

fname = decode\_str\_or\_unicode(f['featureName'])

except UnicodeEncodeError as e:

logging.error("Encountered decoding exception when decoding %s: %s" % (f, e))

decoded\_feature\_names.append(fname)

return decoded\_feature\_names