"""Utility functions to create FeatureConfig objects from feature\_spec.yaml files"""

import os

import re

import tensorflow.compat.v1 as tf

import yaml

from twml.feature\_config import FeatureConfigBuilder

from twml.contrib.feature\_config import FeatureConfigBuilder as FeatureConfigBuilderV2

def \_get\_config\_version(config\_dict):

doc = config\_dict

supported\_classes = {

"twml.FeatureConfig": "v1",

"twml.contrib.FeatureConfig": "v2"

}

if "class" not in doc:

raise ValueError("'class' key not found")

if doc["class"] not in supported\_classes.keys():

raise ValueError("Class %s not supported. Supported clases are %s"

% (doc["class"], supported\_classes.keys()))

return supported\_classes[doc["class"]]

def \_validate\_config\_dict\_v1(config\_dict):

"""

Validate spec exported by twml.FeatureConfig

"""

doc = config\_dict

def malformed\_error(msg):

raise ValueError("twml.FeatureConfig: Malformed feature\_spec. %s" % msg)

if doc["class"] != "twml.FeatureConfig":

malformed\_error("'class' is not twml.FeatureConfig")

if "format" not in doc:

malformed\_error("'format' key not found")

# validate spec exported by twml.FeatureConfig

if doc["format"] == "exported":

dict\_keys = ["features", "labels", "weight", "tensors", "sparse\_tensors"]

for key in dict\_keys:

if key not in doc:

malformed\_error("'%s' key not found" % key)

if type(doc[key]) != dict:

malformed\_error("'%s' is not a dict" % key)

if "filters" not in doc:

malformed\_error("'filters' key not found")

elif type(doc["filters"]) != list:

malformed\_error("'filters' is not a list")

# validate spec provided by modeler

elif doc["format"] == "manual":

raise NotImplementedError("Manual config support not yet implemented")

else:

malformed\_error("'format' must be 'exported' or 'manual'")

def \_validate\_config\_dict\_v2(config\_dict):

"""

Validate spec exported by twml.contrib.FeatureConfig

"""

doc = config\_dict

def malformed\_error(msg):

raise ValueError("twml.contrib.FeatureConfig: Malformed feature\_spec. %s" % msg)

if doc["class"] != "twml.contrib.FeatureConfig":

malformed\_error("'class' is not twml.contrib.FeatureConfig")

if "format" not in doc:

malformed\_error("'format key not found'")

# validate spec exported by twml.contrib.FeatureConfig (basic validation only)

if doc["format"] == "exported":

dict\_keys = ["features", "labels", "weight", "tensors", "sparseTensors", "discretizeConfig"]

for key in dict\_keys:

if key not in doc:

malformed\_error("'%s' key not found" % key)

if type(doc[key]) != dict:

malformed\_error("'%s' is not a dict" % key)

list\_keys = ["sparseFeatureGroups", "denseFeatureGroups", "denseFeatures", "images", "filters"]

for key in list\_keys:

if key not in doc:

malformed\_error("'%s' key not found" % key)

if type(doc[key]) != list:

malformed\_error("'%s' is not a list" % key)

# validate spec provided by modeler

elif doc["format"] == "manual":

raise NotImplementedError("Manual config support not yet implemented")

else:

malformed\_error("'format' must be 'exported' or 'manual'")

def \_create\_feature\_config\_v1(config\_dict, data\_spec\_path):

fc\_builder = FeatureConfigBuilder(data\_spec\_path)

if config\_dict["format"] == "exported":

# add features

for feature\_info in config\_dict["features"].values():

feature\_name = re.escape(feature\_info["featureName"])

feature\_group = feature\_info["featureGroup"]

fc\_builder.add\_feature(feature\_name, feature\_group)

# add labels

labels = []

for label\_info in config\_dict["labels"].values():

labels.append(label\_info["featureName"])

fc\_builder.add\_labels(labels)

# feature filters

for feature\_name in config\_dict["filters"]:

fc\_builder.add\_filter(feature\_name)

# weight

if config\_dict["weight"]:

weight\_feature = list(config\_dict["weight"].values())[0]["featureName"]

fc\_builder.define\_weight(weight\_feature)

else:

raise ValueError("Format '%s' not implemented" % config\_dict["format"])

return fc\_builder.build()

def \_create\_feature\_config\_v2(config\_dict, data\_spec\_path):

fc\_builder = FeatureConfigBuilderV2(data\_spec\_path)

if config\_dict["format"] == "exported":

# add sparse group extraction configs

for sparse\_group in config\_dict["sparseFeatureGroups"]:

fids = sparse\_group["features"].keys()

fnames = [sparse\_group["features"][fid]["featureName"] for fid in fids]

fc\_builder.extract\_features\_as\_hashed\_sparse(

feature\_regexes=[re.escape(fname) for fname in fnames],

output\_tensor\_name=sparse\_group["outputName"],

hash\_space\_size\_bits=sparse\_group["hashSpaceBits"],

discretize\_num\_bins=sparse\_group["discretize"]["numBins"],

discretize\_output\_size\_bits=sparse\_group["discretize"]["outputSizeBits"],

discretize\_type=sparse\_group["discretize"]["type"],

type\_filter=sparse\_group["filterType"])

# add dense group extraction configs

for dense\_group in config\_dict["denseFeatureGroups"]:

fids = dense\_group["features"].keys()

fnames = [dense\_group["features"][fid]["featureName"] for fid in fids]

fc\_builder.extract\_feature\_group(

feature\_regexes=[re.escape(fname) for fname in fnames],

group\_name=dense\_group["outputName"],

type\_filter=dense\_group["filterType"],

default\_value=dense\_group["defaultValue"])

# add dense feature configs

for dense\_features in config\_dict["denseFeatures"]:

fids = dense\_features["features"].keys()

fnames = [dense\_features["features"][fid]["featureName"] for fid in fids]

default\_value = dense\_features["defaultValue"]

if len(fnames) == 1 and type(default\_value) != dict:

fc\_builder.extract\_feature(

feature\_name=re.escape(fnames[0]),

expected\_shape=dense\_features["expectedShape"],

default\_value=dense\_features["defaultValue"])

else:

fc\_builder.extract\_features(

feature\_regexes=[re.escape(fname) for fname in fnames],

default\_value\_map=dense\_features["defaultValue"])

# add image feature configs

for image in config\_dict["images"]:

fc\_builder.extract\_image(

feature\_name=image["featureName"],

preprocess=image["preprocess"],

out\_type=tf.as\_dtype(image["outType"].lower()),

channels=image["channels"],

default\_image=image["defaultImage"],

)

# add other tensor features (non-image)

tensor\_fnames = []

image\_fnames = [img["featureName"] for img in config\_dict["images"]]

for tensor\_fname in config\_dict["tensors"]:

if tensor\_fname not in image\_fnames:

tensor\_fnames.append(tensor\_fname)

for sparse\_tensor\_fname in config\_dict["sparseTensors"]:

tensor\_fnames.append(sparse\_tensor\_fname)

fc\_builder.extract\_tensors(tensor\_fnames)

# add labels

labels = []

for label\_info in config\_dict["labels"].values():

labels.append(label\_info["featureName"])

fc\_builder.add\_labels(labels)

else:

raise ValueError("Format '%s' not implemented" % config\_dict["format"])

return fc\_builder.build()

def create\_feature\_config\_from\_dict(config\_dict, data\_spec\_path):

"""

Create a FeatureConfig object from a feature spec dict.

"""

config\_version = \_get\_config\_version(config\_dict)

if config\_version == "v1":

\_validate\_config\_dict\_v1(config\_dict)

feature\_config = \_create\_feature\_config\_v1(config\_dict, data\_spec\_path)

elif config\_version == "v2":

\_validate\_config\_dict\_v2(config\_dict)

feature\_config = \_create\_feature\_config\_v2(config\_dict, data\_spec\_path)

else:

raise ValueError("version not supported")

return feature\_config

def create\_feature\_config(config\_path, data\_spec\_path):

"""

Create a FeatureConfig object from a feature\_spec.yaml file.

"""

\_, ext = os.path.splitext(config\_path)

if ext not in ['.yaml', '.yml']:

raise ValueError("create\_feature\_config\_from\_yaml: Only .yaml/.yml supported")

with tf.io.gfile.GFile(config\_path, mode='r') as fs:

config\_dict = yaml.safe\_load(fs)

return create\_feature\_config\_from\_dict(config\_dict, data\_spec\_path)