package com.twitter.product\_mixer.component\_library.scorer.tensorbuilder

import com.google.protobuf.ByteString

import com.twitter.product\_mixer.core.feature.Feature

import inference.GrpcService.InferTensorContents

import inference.GrpcService.ModelInferRequest.InferInputTensor

// This class contains most of common versions at Twitter, but in the future we can add more:

// https://github.com/kserve/kserve/blob/master/docs/predict-api/v2/required\_api.md#tensor-data-1

trait InferInputTensorBuilder[Value] {

def apply(

featureName: String,

featureValues: Seq[Value]

): Seq[InferInputTensor]

}

object InferInputTensorBuilder {

def checkTensorShapeMatchesValueLength(

featureName: String,

featureValues: Seq[Any],

tensorShape: Seq[Int]

): Unit = {

val featureValuesSize = featureValues.size

val tensorShapeSize = tensorShape.product

if (featureValuesSize != tensorShapeSize) {

throw new FeatureValuesAndShapeMismatchException(

featureName,

featureValuesSize,

tensorShapeSize)

}

}

def buildBoolInferInputTensor(

featureName: String,

featureValues: Seq[Boolean],

tensorShape: Seq[Int]

): Seq[InferInputTensor] = {

checkTensorShapeMatchesValueLength(featureName, featureValues, tensorShape)

val inputTensorBuilder = InferInputTensor.newBuilder().setName(featureName)

tensorShape.foreach { shape =>

inputTensorBuilder.addShape(shape)

}

val inputTensor = inputTensorBuilder

.setDatatype("BOOL")

.setContents {

val contents = InferTensorContents.newBuilder()

featureValues.foreach { featureValue =>

contents.addBoolContents(featureValue)

}

contents

}

.build()

Seq(inputTensor)

}

def buildBytesInferInputTensor(

featureName: String,

featureValues: Seq[String],

tensorShape: Seq[Int]

): Seq[InferInputTensor] = {

checkTensorShapeMatchesValueLength(featureName, featureValues, tensorShape)

val inputTensorBuilder = InferInputTensor.newBuilder().setName(featureName)

tensorShape.foreach { shape =>

inputTensorBuilder.addShape(shape)

}

val inputTensor = inputTensorBuilder

.setDatatype("BYTES")

.setContents {

val contents = InferTensorContents.newBuilder()

featureValues.foreach { featureValue =>

val featureValueBytes = ByteString.copyFromUtf8(featureValue)

contents.addByteContents(featureValueBytes)

}

contents

}

.build()

Seq(inputTensor)

}

def buildFloat32InferInputTensor(

featureName: String,

featureValues: Seq[Float],

tensorShape: Seq[Int]

): Seq[InferInputTensor] = {

checkTensorShapeMatchesValueLength(featureName, featureValues, tensorShape)

val inputTensorBuilder = InferInputTensor.newBuilder().setName(featureName)

tensorShape.foreach { shape =>

inputTensorBuilder.addShape(shape)

}

val inputTensor = inputTensorBuilder

.setDatatype("FP32")

.setContents {

val contents = InferTensorContents.newBuilder()

featureValues.foreach { featureValue =>

contents.addFp32Contents(featureValue.floatValue)

}

contents

}

.build()

Seq(inputTensor)

}

def buildInt64InferInputTensor(

featureName: String,

featureValues: Seq[Long],

tensorShape: Seq[Int]

): Seq[InferInputTensor] = {

checkTensorShapeMatchesValueLength(featureName, featureValues, tensorShape)

val inputTensorBuilder = InferInputTensor.newBuilder().setName(featureName)

tensorShape.foreach { shape =>

inputTensorBuilder.addShape(shape)

}

val inputTensor = inputTensorBuilder

.setDatatype("INT64")

.setContents {

val contents = InferTensorContents.newBuilder()

featureValues.foreach { featureValue =>

contents.addInt64Contents(featureValue)

}

contents

}

.build()

Seq(inputTensor)

}

}

class UnexpectedFeatureTypeException(feature: Feature[\_, \_])

extends UnsupportedOperationException(s"Unsupported Feature type passed in $feature")

class FeatureValuesAndShapeMismatchException(

featureName: String,

featureValuesSize: Int,

tensorShapeSize: Int)

extends UnsupportedOperationException(

s"Feature $featureName has mismatching FeatureValues (size: $featureValuesSize) and TensorShape (size: $tensorShapeSize)!")

class UnexpectedDataTypeException[T](value: T, builder: InferInputTensorBuilder[\_])

extends UnsupportedOperationException(

s"Unsupported data type ${value} passed in at ${builder.getClass.toString}")