package com.twitter.timelines.prediction.features.two\_hop\_features

import com.twitter.graph\_feature\_service.thriftscala.EdgeType

import com.twitter.ml.api.Feature.\_

import scala.collection.JavaConverters.\_

import TwoHopFeaturesConfig.personalDataTypesMap

object TwoHopFeaturesDescriptor {

val prefix = "two\_hop"

val normalizedPostfix = "normalized"

val leftNodeDegreePostfix = "left\_degree"

val rightNodeDegreePostfix = "right\_degree"

type TwoHopFeatureMap = Map[(EdgeType, EdgeType), Continuous]

type TwoHopFeatureNodeDegreeMap = Map[EdgeType, Continuous]

def apply(edgeTypePairs: Seq[(EdgeType, EdgeType)]): TwoHopFeaturesDescriptor = {

new TwoHopFeaturesDescriptor(edgeTypePairs)

}

}

class TwoHopFeaturesDescriptor(edgeTypePairs: Seq[(EdgeType, EdgeType)]) {

import TwoHopFeaturesDescriptor.\_

def getLeftEdge(edgeTypePair: (EdgeType, EdgeType)): EdgeType = {

edgeTypePair.\_1

}

def getLeftEdgeName(edgeTypePair: (EdgeType, EdgeType)): String = {

getLeftEdge(edgeTypePair).originalName.toLowerCase

}

def getRightEdge(edgeTypePair: (EdgeType, EdgeType)): EdgeType = {

edgeTypePair.\_2

}

def getRightEdgeName(edgeTypePair: (EdgeType, EdgeType)): String = {

getRightEdge(edgeTypePair).originalName.toLowerCase

}

val rawFeaturesMap: TwoHopFeatureMap = edgeTypePairs.map(edgeTypePair => {

val leftEdgeType = getLeftEdge(edgeTypePair)

val leftEdgeName = getLeftEdgeName(edgeTypePair)

val rightEdgeType = getRightEdge(edgeTypePair)

val rightEdgeName = getRightEdgeName(edgeTypePair)

val personalDataTypes = (

personalDataTypesMap.getOrElse(leftEdgeType, Set.empty) ++

personalDataTypesMap.getOrElse(rightEdgeType, Set.empty)

).asJava

val rawFeature = new Continuous(s"$prefix.$leftEdgeName.$rightEdgeName", personalDataTypes)

edgeTypePair -> rawFeature

})(collection.breakOut)

val leftNodeDegreeFeaturesMap: TwoHopFeatureNodeDegreeMap = edgeTypePairs.map(edgeTypePair => {

val leftEdgeType = getLeftEdge(edgeTypePair)

val leftEdgeName = getLeftEdgeName(edgeTypePair)

val personalDataTypes = personalDataTypesMap.getOrElse(leftEdgeType, Set.empty).asJava

val leftNodeDegreeFeature =

new Continuous(s"$prefix.$leftEdgeName.$leftNodeDegreePostfix", personalDataTypes)

leftEdgeType -> leftNodeDegreeFeature

})(collection.breakOut)

val rightNodeDegreeFeaturesMap: TwoHopFeatureNodeDegreeMap = edgeTypePairs.map(edgeTypePair => {

val rightEdgeType = getRightEdge(edgeTypePair)

val rightEdgeName = getRightEdgeName(edgeTypePair)

val personalDataTypes = personalDataTypesMap.getOrElse(rightEdgeType, Set.empty).asJava

val rightNodeDegreeFeature =

new Continuous(s"$prefix.$rightEdgeName.$rightNodeDegreePostfix", personalDataTypes)

rightEdgeType -> rightNodeDegreeFeature

})(collection.breakOut)

val normalizedFeaturesMap: TwoHopFeatureMap = edgeTypePairs.map(edgeTypePair => {

val leftEdgeType = getLeftEdge(edgeTypePair)

val leftEdgeName = getLeftEdgeName(edgeTypePair)

val rightEdgeType = getRightEdge(edgeTypePair)

val rightEdgeName = getRightEdgeName(edgeTypePair)

val personalDataTypes = (

personalDataTypesMap.getOrElse(leftEdgeType, Set.empty) ++

personalDataTypesMap.getOrElse(rightEdgeType, Set.empty)

).asJava

val normalizedFeature =

new Continuous(s"$prefix.$leftEdgeName.$rightEdgeName.$normalizedPostfix", personalDataTypes)

edgeTypePair -> normalizedFeature

})(collection.breakOut)

private val rawFeaturesSeq: Seq[Continuous] = rawFeaturesMap.values.toSeq

private val leftNodeDegreeFeaturesSeq: Seq[Continuous] = leftNodeDegreeFeaturesMap.values.toSeq

private val rightNodeDegreeFeaturesSeq: Seq[Continuous] = rightNodeDegreeFeaturesMap.values.toSeq

private val normalizedFeaturesSeq: Seq[Continuous] = normalizedFeaturesMap.values.toSeq

val featuresSeq: Seq[Continuous] =

rawFeaturesSeq ++ leftNodeDegreeFeaturesSeq ++ rightNodeDegreeFeaturesSeq ++ normalizedFeaturesSeq

}