package com.twitter.interaction\_graph.scio.common

import com.twitter.interaction\_graph.thriftscala.FeatureName

import com.twitter.interaction\_graph.thriftscala.TimeSeriesStatistics

import com.twitter.interaction\_graph.thriftscala.Vertex

import com.twitter.interaction\_graph.thriftscala.VertexFeature

object VertexFeatureCombiner {

def apply(userId: Long): VertexFeatureCombiner = new VertexFeatureCombiner(

instanceVertex = Vertex(userId),

featureMap = Map(

(FeatureName.NumRetweets, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRetweets, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumFavorites, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumFavorites, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumMentions, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumMentions, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumTweetClicks, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumTweetClicks, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumLinkClicks, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumLinkClicks, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumProfileViews, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumProfileViews, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumFollows, true) -> new ReplacementVertexCombiner,

(FeatureName.NumFollows, false) -> new ReplacementVertexCombiner,

(FeatureName.NumUnfollows, true) -> new ReplacementVertexCombiner,

(FeatureName.NumUnfollows, false) -> new ReplacementVertexCombiner,

(FeatureName.NumMutualFollows, true) -> new ReplacementVertexCombiner,

(FeatureName.NumBlocks, true) -> new ReplacementVertexCombiner,

(FeatureName.NumBlocks, false) -> new ReplacementVertexCombiner,

(FeatureName.NumMutes, true) -> new ReplacementVertexCombiner,

(FeatureName.NumMutes, false) -> new ReplacementVertexCombiner,

(FeatureName.NumReportAsAbuses, true) -> new ReplacementVertexCombiner,

(FeatureName.NumReportAsAbuses, false) -> new ReplacementVertexCombiner,

(FeatureName.NumReportAsSpams, true) -> new ReplacementVertexCombiner,

(FeatureName.NumReportAsSpams, false) -> new ReplacementVertexCombiner,

(FeatureName.NumTweetQuotes, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumTweetQuotes, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumMutualFollows, false) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookEmail, true) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookEmail, false) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookPhone, true) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookPhone, false) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookInBoth, true) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookInBoth, false) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookMutualEdgeEmail, true) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookMutualEdgeEmail, false) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookMutualEdgePhone, true) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookMutualEdgePhone, false) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookMutualEdgeInBoth, true) -> new ReplacementVertexCombiner,

(FeatureName.AddressBookMutualEdgeInBoth, false) -> new ReplacementVertexCombiner,

(FeatureName.TotalDwellTime, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.TotalDwellTime, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumInspectedStatuses, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumInspectedStatuses, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumPhotoTags, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumPhotoTags, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumPushOpens, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumPushOpens, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumNtabClicks, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumNtabClicks, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtFavories, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtFavories, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtTweetQuotes, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtTweetQuotes, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtTweetClicks, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtTweetClicks, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtRetweets, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtRetweets, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtReplies, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtReplies, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtLinkClicks, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtLinkClicks, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtMentions, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumRtMentions, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumShares, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumShares, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumEmailOpen, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumEmailOpen, false) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumEmailClick, true) -> new WeightedAdditiveVertexCombiner,

(FeatureName.NumEmailClick, false) -> new WeightedAdditiveVertexCombiner,

)

)

}

/\*\*

\* This class can take in a number of input Vertex thrift objects (all of which are assumed to

\* contain information about a single vertex) and builds a combined Vertex protobuf object, which

\* has the union of all the input. Note that we do a weighted addition for a time-decayed value.

\* <p>

\* The input objects features must be disjoint. Also, remember that the Vertex is directed!

\*/

class VertexFeatureCombiner(

instanceVertex: Vertex,

featureMap: Map[(FeatureName, Boolean), VFeatureCombiner]) {

/\*\*

\* Adds features without any decay. To be used for the same day.

\*

\* @param vertex vertex to be added into the combiner

\*/

def addFeature(vertex: Vertex): VertexFeatureCombiner = {

val newVertex = instanceVertex.copy(weight = vertex.weight)

val newFeatures = featureMap.map {

case ((featureName, outgoing), combiner) =>

vertex.features.find(f => f.name.equals(featureName) && f.outgoing.equals(outgoing)) match {

case Some(feature) =>

val updatedCombiner =

if (combiner.isSet) combiner.updateFeature(feature) else combiner.setFeature(feature)

((featureName, outgoing), updatedCombiner)

case \_ => ((featureName, outgoing), combiner)

}

}

new VertexFeatureCombiner(newVertex, newFeatures)

}

/\*\*

\* Adds features with decays. Used for combining multiple days.

\*

\* @param vertex vertex to be added into the combiner

\* @param alpha parameters for the decay calculation

\* @param day number of days from today

\*/

def addFeature(vertex: Vertex, alpha: Double, day: Int): VertexFeatureCombiner = {

val newVertex = instanceVertex.copy(weight = vertex.weight)

val newFeatures = featureMap.map {

case ((featureName, outgoing), combiner) =>

vertex.features.find(f => f.name.equals(featureName) && f.outgoing.equals(outgoing)) match {

case Some(feature) =>

val updatedCombiner =

if (combiner.isSet) combiner.updateFeature(feature, alpha, day)

else combiner.setFeature(feature, alpha, day)

((featureName, outgoing), updatedCombiner)

case \_ => ((featureName, outgoing), combiner)

}

}

new VertexFeatureCombiner(newVertex, newFeatures)

}

/\*\*

\* Generate the final combined Vertex instance

\*

\* @param totalDays total number of days to be combined together

\*/

def getCombinedVertex(totalDays: Int): Vertex = {

val moreFeatures = featureMap.values.flatMap {

case combiner => combiner.getFinalFeature(totalDays)

}

instanceVertex.copy(features = moreFeatures.toSeq)

}

}

/\*\*

\* This portion contains the actual combination logic. For now, we only implement a simple

\* additive combiner, but in future we'd like to have things like time-weighted (exponential

\* decay, maybe) values.

\*/

trait VFeatureCombiner {

val startingDay: Int

val endingDay: Int

val timeSeriesStatistics: Option[TimeSeriesStatistics]

val vertexFeature: Option[VertexFeature]

def updateTss(feature: VertexFeature, alpha: Double): VFeatureCombiner

def addToTss(feature: VertexFeature): VFeatureCombiner

def updateFeature(feature: VertexFeature, alpha: Double, day: Int): VFeatureCombiner

def updateFeature(feature: VertexFeature): VFeatureCombiner

def isSet: Boolean

def dropFeature: Boolean

def setFeature(feature: VertexFeature, alpha: Double, day: Int): VFeatureCombiner

def setFeature(feature: VertexFeature): VFeatureCombiner

def getFinalFeature(totalDays: Int): Option[VertexFeature]

}

case class WeightedAdditiveVertexCombiner(

override val vertexFeature: Option[VertexFeature] = None,

override val startingDay: Int = Integer.MAX\_VALUE,

override val endingDay: Int = Integer.MIN\_VALUE,

override val timeSeriesStatistics: Option[TimeSeriesStatistics] = None)

extends VFeatureCombiner {

override def updateTss(

feature: VertexFeature,

alpha: Double

): WeightedAdditiveVertexCombiner = copy(timeSeriesStatistics = timeSeriesStatistics.map(tss =>

InteractionGraphUtils.updateTimeSeriesStatistics(tss, feature.tss.mean, alpha)))

override def addToTss(feature: VertexFeature): WeightedAdditiveVertexCombiner =

copy(timeSeriesStatistics = timeSeriesStatistics.map(tss =>

InteractionGraphUtils.addToTimeSeriesStatistics(tss, feature.tss.mean)))

override def updateFeature(feature: VertexFeature, alpha: Double, day: Int): VFeatureCombiner = {

updateTss(feature, alpha).copy(

vertexFeature,

startingDay = startingDay,

endingDay = Math.max(endingDay, day)

)

}

override def updateFeature(feature: VertexFeature): VFeatureCombiner =

addToTss(feature)

override def setFeature(feature: VertexFeature, alpha: Double, day: Int): VFeatureCombiner = {

val newStartingDay = Math.min(startingDay, day)

val newEndingDay = Math.max(endingDay, day)

val numDaysSinceLast =

if (feature.tss.numDaysSinceLast.exists(\_ > 0))

feature.tss.numDaysSinceLast

else Some(feature.tss.numElapsedDays - feature.tss.numNonZeroDays + 1)

val tss = feature.tss.copy(numDaysSinceLast = numDaysSinceLast)

val newFeature = VertexFeature(

name = feature.name,

outgoing = feature.outgoing,

tss = tss

)

WeightedAdditiveVertexCombiner(

Some(newFeature),

newStartingDay,

newEndingDay,

Some(tss)

)

}

def getFinalFeature(totalDays: Int): Option[VertexFeature] = {

if (vertexFeature.isEmpty || dropFeature) return None

val newTss = if (totalDays > 0) {

val elapsed =

timeSeriesStatistics.map(tss => tss.numElapsedDays + totalDays - 1 - startingDay)

val latest =

if (endingDay > 0) Some(totalDays - endingDay)

else timeSeriesStatistics.map(tss => tss.numDaysSinceLast.get + totalDays - 1)

timeSeriesStatistics.map(tss =>

tss.copy(

numElapsedDays = elapsed.get,

numDaysSinceLast = latest

))

} else timeSeriesStatistics

vertexFeature.map(vf => vf.copy(tss = newTss.get))

}

override def setFeature(feature: VertexFeature): VFeatureCombiner = setFeature(feature, 1.0, 0)

override def isSet: Boolean = vertexFeature.isDefined

override def dropFeature: Boolean =

timeSeriesStatistics.exists(tss =>

tss.numDaysSinceLast.exists(\_ > InteractionGraphUtils.MAX\_DAYS\_RETENTION) &&

tss.ewma < InteractionGraphUtils.MIN\_FEATURE\_VALUE)

}

/\*\*

\* This combiner always replaces the old value with the current. Ignores time-decays.

\*/

case class ReplacementVertexCombiner(

override val vertexFeature: Option[VertexFeature] = None,

override val startingDay: Int = Integer.MAX\_VALUE,

override val endingDay: Int = Integer.MIN\_VALUE,

override val timeSeriesStatistics: Option[TimeSeriesStatistics] = None)

extends VFeatureCombiner {

override def updateTss(

feature: VertexFeature,

alpha: Double

): ReplacementVertexCombiner = setFeature(feature, 1.0, 0)

override def addToTss(feature: VertexFeature): ReplacementVertexCombiner =

setFeature(feature, 1.0, 0)

override def updateFeature(

feature: VertexFeature,

alpha: Double,

day: Int

): ReplacementVertexCombiner = updateTss(feature, alpha).copy(

vertexFeature,

startingDay = startingDay,

endingDay = Math.max(endingDay, day)

)

override def updateFeature(feature: VertexFeature): ReplacementVertexCombiner =

addToTss(feature)

override def setFeature(

feature: VertexFeature,

alpha: Double,

day: Int

): ReplacementVertexCombiner = {

val newStartingDay = Math.min(startingDay, day)

val newEndingDay = Math.max(endingDay, day)

val numDaysSinceLast =

if (feature.tss.numDaysSinceLast.exists(\_ > 0))

feature.tss.numDaysSinceLast

else Some(feature.tss.numElapsedDays - feature.tss.numNonZeroDays + 1)

val tss = feature.tss.copy(numDaysSinceLast = numDaysSinceLast)

val newFeature = VertexFeature(

name = feature.name,

outgoing = feature.outgoing,

tss = tss

)

ReplacementVertexCombiner(

Some(newFeature),

newStartingDay,

newEndingDay,

Some(tss)

)

}

override def getFinalFeature(totalDays: Int): Option[VertexFeature] = {

if (vertexFeature.isEmpty || dropFeature) return None

if (timeSeriesStatistics.exists(tss => tss.ewma < 1.0)) return None

val newTss = if (totalDays > 0) {

val latest =

if (endingDay > 0) totalDays - endingDay

else timeSeriesStatistics.get.numDaysSinceLast.get + totalDays - 1

timeSeriesStatistics.map(tss =>

tss.copy(

numElapsedDays = 1,

numDaysSinceLast = Some(latest)

))

} else timeSeriesStatistics

vertexFeature.map(vf => vf.copy(tss = newTss.get))

}

override def setFeature(feature: VertexFeature): VFeatureCombiner = setFeature(feature, 1.0, 0)

override def isSet: Boolean = vertexFeature.isDefined

override def dropFeature: Boolean =

timeSeriesStatistics.exists(tss =>

tss.numDaysSinceLast.exists(\_ > InteractionGraphUtils.MAX\_DAYS\_RETENTION) &&

tss.ewma < InteractionGraphUtils.MIN\_FEATURE\_VALUE)

}