package com.twitter.follow\_recommendations.common.models

import com.twitter.follow\_recommendations.common.rankers.common.RankerId

import com.twitter.follow\_recommendations.common.rankers.common.RankerId.RankerId

import com.twitter.follow\_recommendations.logging.{thriftscala => offline}

import com.twitter.follow\_recommendations.{thriftscala => t}

/\*\*

\* Type of Score. This is used to differentiate scores.

\*

\* Define it as a trait so it is possible to add more information for different score types.

\*/

sealed trait ScoreType {

def getName: String

}

/\*\*

\* Existing Score Types

\*/

object ScoreType {

/\*\*

\* the score is calculated based on heuristics and most likely not normalized

\*/

case object HeuristicBasedScore extends ScoreType {

override def getName: String = "HeuristicBasedScore"

}

/\*\*

\* probability of follow after the candidate is recommended to the user

\*/

case object PFollowGivenReco extends ScoreType {

override def getName: String = "PFollowGivenReco"

}

/\*\*

\* probability of engage after the user follows the candidate

\*/

case object PEngagementGivenFollow extends ScoreType {

override def getName: String = "PEngagementGivenFollow"

}

/\*\*

\* probability of engage per tweet impression

\*/

case object PEngagementPerImpression extends ScoreType {

override def getName: String = "PEngagementPerImpression"

}

/\*\*

\* probability of engage per tweet impression

\*/

case object PEngagementGivenReco extends ScoreType {

override def getName: String = "PEngagementGivenReco"

}

def fromScoreTypeString(scoreTypeName: String): ScoreType = scoreTypeName match {

case "HeuristicBasedScore" => HeuristicBasedScore

case "PFollowGivenReco" => PFollowGivenReco

case "PEngagementGivenFollow" => PEngagementGivenFollow

case "PEngagementPerImpression" => PEngagementPerImpression

case "PEngagementGivenReco" => PEngagementGivenReco

}

}

/\*\*

\* Represent the output from a certain ranker or scorer. All the fields are optional

\*

\* @param value value of the score

\* @param rankerId ranker id

\* @param scoreType score type

\*/

final case class Score(

value: Double,

rankerId: Option[RankerId] = None,

scoreType: Option[ScoreType] = None) {

def toThrift: t.Score = t.Score(

value = value,

rankerId = rankerId.map(\_.toString),

scoreType = scoreType.map(\_.getName)

)

def toOfflineThrift: offline.Score =

offline.Score(

value = value,

rankerId = rankerId.map(\_.toString),

scoreType = scoreType.map(\_.getName)

)

}

object Score {

val RandomScore = Score(0.0d, Some(RankerId.RandomRanker))

def optimusScore(score: Double, scoreType: ScoreType): Score = {

Score(value = score, scoreType = Some(scoreType))

}

def predictionScore(score: Double, rankerId: RankerId): Score = {

Score(value = score, rankerId = Some(rankerId))

}

def fromThrift(thriftScore: t.Score): Score =

Score(

value = thriftScore.value,

rankerId = thriftScore.rankerId.flatMap(RankerId.getRankerByName),

scoreType = thriftScore.scoreType.map(ScoreType.fromScoreTypeString)

)

}

/\*\*

\* a list of scores

\*/

final case class Scores(

scores: Seq[Score],

selectedRankerId: Option[RankerId] = None,

isInProducerScoringExperiment: Boolean = false) {

def toThrift: t.Scores =

t.Scores(

scores = scores.map(\_.toThrift),

selectedRankerId = selectedRankerId.map(\_.toString),

isInProducerScoringExperiment = isInProducerScoringExperiment

)

def toOfflineThrift: offline.Scores =

offline.Scores(

scores = scores.map(\_.toOfflineThrift),

selectedRankerId = selectedRankerId.map(\_.toString),

isInProducerScoringExperiment = isInProducerScoringExperiment

)

}

object Scores {

val Empty: Scores = Scores(Nil)

def fromThrift(thriftScores: t.Scores): Scores =

Scores(

scores = thriftScores.scores.map(Score.fromThrift),

selectedRankerId = thriftScores.selectedRankerId.flatMap(RankerId.getRankerByName),

isInProducerScoringExperiment = thriftScores.isInProducerScoringExperiment

)

}