namespace java com.twitter.simclusters\_v2.thriftjava

namespace py gen.twitter.simclusters\_v2.interests

#@namespace scala com.twitter.simclusters\_v2.thriftscala

#@namespace strato com.twitter.simclusters\_v2

/\*\*

\* All of the scores below assume that the knownFor vector for each cluster is already

\* of unit L2 norm i.e. sum of squares is 1.

\*\*/

struct UserToInterestedInClusterScores {

// dot product of user's binary follow vector with knownFor vector for this cluster

// TIP: By default, use this score or favScore.

1: optional double followScore(personalDataType = 'CountOfFollowersAndFollowees')

// first compute followScore as defined above

// then compute L2 norm of the vector of these scores for this cluster

// divide by that.

// essentially the more people are interested in this cluster, the lower this score gets

// TIP: Use this score if your use case needs to penalize clusters that a lot of other

// users are also interested in

2: optional double followScoreClusterNormalizedOnly(personalDataType = 'CountOfFollowersAndFollowees')

// dot product of user's producer normalized follow vector and knownFor vector for this cluster

// i.e. i^th entry in the normalized follow vector = 1.0/sqrt(number of followers of user i)

// TIP: Use this score if your use case needs to penalize clusters where the users known for

// that cluster are popular.

3: optional double followScoreProducerNormalizedOnly(personalDataType = 'CountOfFollowersAndFollowees')

// first compute followScoreProducerNormalizedOnly

// then compute L2 norm of the vector of these scores for this cluster

// divide by that.

// essentially the more people are interested in this cluster, the lower this score gets

// TIP: Use this score if your use case needs to penalize both clusters that a lot of other

// users are interested in, as well as clusters where the users known for that cluster are

// popular.

4: optional double followScoreClusterAndProducerNormalized(personalDataType = 'CountOfFollowersAndFollowees')

// dot product of user's favScoreHalfLife100Days vector with knownFor vector for this cluster

// TIP: By default, use this score or followScore.

5: optional double favScore(personalDataType = 'EngagementsPublic')

// first compute favScore as defined above

// then compute L2 norm of the vector of these scores for this cluster

// divide by that.

// essentially the more people are interested in this cluster, the lower this score gets

// TIP: Use this score if your use case needs to penalize clusters that a lot of other

// users are also interested in

6: optional double favScoreClusterNormalizedOnly(personalDataType = 'EngagementsPublic')

// dot product of user's favScoreHalfLife100DaysNormalizedByNeighborFaversL2 vector with

// knownFor vector for this cluster

// TIP: Use this score if your use case needs to penalize clusters where the users known for

// that cluster are popular.

7: optional double favScoreProducerNormalizedOnly(personalDataType = 'EngagementsPublic')

// first compute favScoreProducerNormalizedOnly as defined above

// then compute L2 norm of the vector of these scores for this cluster

// divide by that.

// essentially the more people are interested in this cluster, the lower this score gets

// TIP: Use this score if your use case needs to penalize both clusters that a lot of other

// users are interested in, as well as clusters where the users known for that cluster are

// popular.

8: optional double favScoreClusterAndProducerNormalized(personalDataType = 'EngagementsPublic')

// list of users who're known for this cluster as well as are being followed by the user.

9: optional list<i64> usersBeingFollowed(personalDataType = 'UserId')

// list of users who're known for this cluster as well as were faved at some point by the user.

10: optional list<i64> usersThatWereFaved(personalDataType = 'UserId')

// A pretty close upper bound on the number of users who are interested in this cluster.

// Useful to know if this is a niche community or a popular topic.

11: optional i32 numUsersInterestedInThisClusterUpperBound

// dot product of user's logFavScore vector with knownFor vector for this cluster

// TIP: this score is under experimentations

12: optional double logFavScore(personalDataType = 'EngagementsPublic')

// first compute logFavScore as defined above

// then compute L2 norm of the vector of these scores for this cluster

// divide by that.

// essentially the more people are interested in this cluster, the lower this score gets

// TIP: this score is under experimentations

13: optional double logFavScoreClusterNormalizedOnly(personalDataType = 'EngagementsPublic')

// actual count of number of users who're known for this cluster as well as are being followed by the user.

14: optional i32 numUsersBeingFollowed

// actual count of number of users who're known for this cluster as well as were faved at some point by the user.

15: optional i32 numUsersThatWereFaved

}(persisted = 'true', hasPersonalData = 'true')

struct UserToInterestedInClusters {

1: required i64 userId(personalDataType = 'UserId')

2: required string knownForModelVersion

3: required map<i32, UserToInterestedInClusterScores> clusterIdToScores(personalDataTypeKey = 'InferredInterests')

}(persisted="true", hasPersonalData = 'true')

struct LanguageToClusters {

1: required string language

2: required string knownForModelVersion

3: required map<i32, UserToInterestedInClusterScores> clusterIdToScores(personalDataTypeKey = 'InferredInterests')

}(persisted="true", hasPersonalData = 'true')

struct ClustersUserIsInterestedIn {

1: required string knownForModelVersion

2: required map<i32, UserToInterestedInClusterScores> clusterIdToScores(personalDataTypeKey = 'InferredInterests')

}(persisted = 'true', hasPersonalData = 'true')

struct UserToKnownForClusters {

1: required i64 userId(personalDataType = 'UserId')

2: required string knownForModelVersion

3: required map<i32, UserToKnownForClusterScores> clusterIdToScores(personalDataTypeKey = 'InferredInterests')

}(persisted="true", hasPersonalData = 'true')

struct UserToKnownForClusterScores {

1: optional double knownForScore

}(persisted = 'true', hasPersonalData = 'false')

struct ClustersUserIsKnownFor {

1: required string knownForModelVersion

2: required map<i32, UserToKnownForClusterScores> clusterIdToScores(personalDataTypeKey = 'InferredInterests')

}(persisted = 'true', hasPersonalData = 'true')

/\*\* Thrift struct for storing quantile bounds output by QTreeMonoid in Algebird \*/

struct QuantileBounds {

1: required double lowerBound

2: required double upperBound

}(persisted = 'true', hasPersonalData = 'false')

/\*\* Thrift struct giving the details of the distribution of a set of doubles \*/

struct DistributionDetails {

1: required double mean

2: optional double standardDeviation

3: optional double min

4: optional QuantileBounds p25

5: optional QuantileBounds p50

6: optional QuantileBounds p75

7: optional QuantileBounds p95

8: optional double max

}(persisted = 'true', hasPersonalData = 'false')

/\*\* Note that the modelVersion here is specified somewhere outside, specifically, as part of the key \*/

struct ClusterNeighbor {

1: required i32 clusterId

/\*\* Note that followCosineSimilarity is same as dot product over followScoreClusterNormalizedOnly

\* since those scores form a unit vector \*\*/

2: optional double followCosineSimilarity

/\*\* Note that favCosineSimilarity is same as dot product over favScoreClusterNormalizedOnly

\* since those scores form a unit vector \*\*/

3: optional double favCosineSimilarity

/\*\* Note that logFavCosineSimilarity is same as dot product over logFavScoreClusterNormalizedOnly

\* since those scores form a unit vector \*\*/

4: optional double logFavCosineSimilarity

}(persisted = 'true', hasPersonalData = 'false')

/\*\* Useful for storing the list of users known for a cluster \*/

struct UserWithScore {

1: required i64 userId(personalDataType = 'UserId')

2: required double score

}(persisted="true", hasPersonalData = 'true')

// deprecated

struct EdgeCut {

1: required double cutEdges

2: required double totalVolume

}(persisted = 'true', hasPersonalData = 'false')

struct ClusterQuality {

// deprecated

1: optional EdgeCut deprecated\_unweightedEdgeCut

// deprecated

2: optional EdgeCut deprecated\_edgeWeightedCut

// deprecated

3: optional EdgeCut deprecated\_nodeAndEdgeWeightedCut

// correlation of actual weight of (u, v) with I(u & v in same cluster) \* score(u) \* score(v)

4: optional double weightAndProductOfNodeScoresCorrelation

// fraction of edges staying inside cluster divided by total edges from nodes in the cluster

5: optional double unweightedRecall

// fraction of edge weights staying inside cluster divided by total edge weights from nodes in the cluster

6: optional double weightedRecall

// total edges from nodes in the cluster

7: optional double unweightedRecallDenominator

// total edge weights from nodes in the cluster

8: optional double weightedRecallDenominator

// sum of edge weights inside cluster / { #nodes \* (#nodes - 1) }

9: optional double relativePrecisionNumerator

// above divided by the sum of edge weights in the total graph / { n \* (n - 1) }

10: optional double relativePrecision

}(persisted = 'true', hasPersonalData = 'false')

/\*\*

\* This struct is the value of the ClusterDetails key-value dataset.

\* The key is (modelVersion, clusterId)

\*\*/

struct ClusterDetails {

1: required i32 numUsersWithAnyNonZeroScore

2: required i32 numUsersWithNonZeroFollowScore

3: required i32 numUsersWithNonZeroFavScore

4: optional DistributionDetails followScoreDistributionDetails

5: optional DistributionDetails favScoreDistributionDetails

6: optional list<UserWithScore> knownForUsersAndScores

7: optional list<ClusterNeighbor> neighborClusters

// fraction of users who're known for this cluster who're marked NSFW\_User in UserSource

8: optional double fractionKnownForMarkedNSFWUser

// the major languages that this cluster's known\_fors have as their "language" field in

// UserSource, and the fractions

9: optional map<string, double> languageToFractionDeviceLanguage

// the major country codes that this cluster's known\_fors have as their "account\_country\_code"

// field in UserSource, and the fractions

10: optional map<string, double> countryCodeToFractionKnownForWithCountryCode

11: optional ClusterQuality qualityMeasuredOnSimsGraph

12: optional DistributionDetails logFavScoreDistributionDetails

// fraction of languages this cluster's known\_fors produce based on what penguin\_user\_languages dataset infers

13: optional map<string, double> languageToFractionInferredLanguage

}(persisted="true", hasPersonalData = 'true')

struct SampledEdge {

1: required i64 followerId(personalDataType = 'UserId')

2: required i64 followeeId(personalDataType = 'UserId')

3: optional double favWtIfFollowEdge

4: optional double favWtIfFavEdge

5: optional double followScoreToCluster

6: optional double favScoreToCluster

7: optional double predictedFollowScore

8: optional double predictedFavScore

}(persisted="true", hasPersonalData = 'true')

/\*\*

\* The key here is (modelVersion, clusterId)

\*\*/

struct BipartiteClusterQuality {

1: optional double inClusterFollowEdges

2: optional double inClusterFavEdges

3: optional double favWtSumOfInClusterFollowEdges

4: optional double favWtSumOfInClusterFavEdges

5: optional double outgoingFollowEdges

6: optional double outgoingFavEdges

7: optional double favWtSumOfOutgoingFollowEdges

8: optional double favWtSumOfOutgoingFavEdges

9: optional double incomingFollowEdges

10: optional double incomingFavEdges

11: optional double favWtSumOfIncomingFollowEdges

12: optional double favWtSumOfIncomingFavEdges

13: optional i32 interestedInSize

14: optional list<SampledEdge> sampledEdges

15: optional i32 knownForSize

16: optional double correlationOfFavWtIfFollowWithPredictedFollow

17: optional double correlationOfFavWtIfFavWithPredictedFav

18: optional double relativePrecisionUsingFavWtIfFav

19: optional double averagePrecisionOfWholeGraphUsingFavWtIfFav

}(persisted="true", hasPersonalData = 'true')