package com.twitter.search.common.relevance.classifiers;

import com.google.common.base.Preconditions;

import com.twitter.search.common.relevance.entities.TwitterMessage;

/\*\*

\* Interface to perform feature classification for a single

\* @TwitterMessage object, or a group of them.

\*

\* Classification includes two steps: feature extraction, and

\* quality evaluation. During feature extraction, any interesting

\* feature that is deemed useful for subsequent quality analysis

\* is extracted from the @TwitterMessage object. Quality evaluation

\* is then done by a group of @TweetEvaluator objects associated

\* with the classifier, by using the various features extracted in the

\* previous step.

\*

\* Feature extraction and quality evaluation results are stored in

\* @TweetFeatures field of the @TwitterMessage object, which is defined

\* in src/main/thrift/classifier.thrift.

\*/

public abstract class TweetClassifier {

/\*\*

\* A list of TweetQualityEvaluators which are invoked after

\* feature extraction is done. If null, no quality evaluation

\* is done.

\*/

protected Iterable<TweetEvaluator> qualityEvaluators = null;

/\*\*

\* Passed in TwitterMessage is examined and any extractable

\* features are saved in TweetFeatures field of TwitterMessage.

\* Then TweetQualityEvaluators are applied to compute various

\* quality values.

\*

\* @param tweet TwitterMessage to perform classification on.

\*/

public void classifyTweet(final TwitterMessage tweet) {

Preconditions.checkNotNull(tweet);

// extract features

extractFeatures(tweet);

// compute quality

evaluate(tweet);

}

/\*\*

\* Classify a group of TwitterMessages and store features in their corresponding

\* TweetFeatures fields.

\*

\* This default implementation just iterates through the map and classifies each

\* individual tweet. Batching for better performance, if applicable, can be implemented by

\* concrete subclasses.

\*

\* @param tweets TwitterMessages to perform classification on.

\*/

public void classifyTweets(final Iterable<TwitterMessage> tweets) {

extractFeatures(tweets);

evaluate(tweets);

}

/\*\*

\* Use the specified list of TweetQualityEvaluators for this classifier.

\*

\* @param evaluators list of TweetQualityEvaluators to be used with this classifier.

\*/

protected void setQualityEvaluators(final Iterable<TweetEvaluator> qualityEvaluators) {

Preconditions.checkNotNull(qualityEvaluators);

this.qualityEvaluators = qualityEvaluators;

}

/\*\*

\* Extract interesting features from a single TwitterMessage for classification.

\*

\* @param tweet TwitterMessage to extract interesting features for

\*/

protected abstract void extractFeatures(final TwitterMessage tweet);

/\*\*

\* Extract interesting features from a list of TwitterMessages for classification.

\* @param tweets list of TwitterMessages to extract interesting features for

\*/

protected void extractFeatures(final Iterable<TwitterMessage> tweets) {

for (TwitterMessage tweet: tweets) {

extractFeatures(tweet);

}

}

/\*\*

\* Given a TwitterMessage which already has its features extracted,

\* perform quality evaluation.

\*

\* @param tweet TwitterMessage to perform quality evaluation for

\*/

protected void evaluate(final TwitterMessage tweet) {

if (qualityEvaluators == null) {

return;

}

for (TweetEvaluator evaluator : qualityEvaluators) {

evaluator.evaluate(tweet);

}

}

/\*\*

\* Given a list of TwitterMessages which already have their features extracted,

\* perform quality evaluation.

\*

\* @param tweets list of TwitterMessages to perform quality evaluation for

\*/

protected void evaluate(final Iterable<TwitterMessage> tweets) {

for (TwitterMessage tweet: tweets) {

evaluate(tweet);

}

}

}