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

import java.util.Map;

import com.google.common.collect.ImmutableMap;

import com.google.common.collect.Maps;

import com.twitter.common.base.Function;

/\*\*

\* Class to keep String-Double of term vectors

\* It can calculate magnitude, dot product, and cosine similarity

\*/

public class TermVector {

private static final double MIN\_MAGNITUDE = 0.00001;

private final double magnitude;

private final ImmutableMap<String, Double> termWeights;

/\*\* Creates a new TermVector instance. \*/

public TermVector(Map<String, Double> termWeights) {

this.termWeights = ImmutableMap.copyOf(termWeights);

double sum = 0.0;

for (Map.Entry<String, Double> entry : termWeights.entrySet()) {

double value = entry.getValue();

sum += value \* value;

}

magnitude = Math.sqrt(sum);

}

public ImmutableMap<String, Double> getTermWeights() {

return termWeights;

}

public double getMagnitude() {

return magnitude;

}

/\*\*

\* Normalize term vector into unit magnitude

\* @return the unit normalized TermVector with magnitude equals 1

\* return null if magnitude is very low

\*/

public TermVector getUnitNormalized() {

if (magnitude < MIN\_MAGNITUDE) {

return null;

}

return new TermVector(

Maps.transformValues(termWeights, (Function<Double, Double>) weight -> weight / magnitude));

}

/\*\*

\* Calculate the dot product with another term vector

\* @param other the other term vector

\* @return the dot product of the two vectors

\*/

public double getDotProduct(TermVector other) {

double sum = 0.0;

for (Map.Entry<String, Double> entry : termWeights.entrySet()) {

Double value2 = other.termWeights.get(entry.getKey());

if (value2 != null) {

sum += entry.getValue() \* value2;

}

}

return sum;

}

/\*\*

\* Calculate the cosine similarity of with another term vector

\* @param other the other term vector

\* @return the cosine similarity.

\* if either has very small magnitude, it returns 0 (dotProduct close to 0)

\*/

public double getCosineSimilarity(TermVector other) {

if (magnitude < MIN\_MAGNITUDE || other.magnitude < MIN\_MAGNITUDE) {

return 0;

}

return getDotProduct(other) / (magnitude \* other.magnitude);

}

}