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

import java.util.concurrent.TimeUnit;

import com.google.common.base.Preconditions;

/\*\*

\* Utility to compute an age decay multiplier based on a sigmoid function.

\*/

public class AgeDecay {

public static final double SLOPE\_COEFF = 4.0;

public static final double LN\_HALF = Math.log(0.5);

public final double halflife;

public final double maxBoost;

public final double base;

public final double slope;

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

public AgeDecay(double base, double maxBoost, double halflife, double slope) {

this.maxBoost = maxBoost;

this.base = base;

this.halflife = halflife;

this.slope = slope;

}

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

public AgeDecay(double base, double halflife, double slope) {

this(base, 1.0, halflife, slope);

}

/\*\*

\* Compute the age decay, using the provided halflife.

\*

\* @param tweetAge The tweet age.

\* @param unit The unit of the tweetAge parameter.

\*/

public double getAgeDecayMultiplier(long tweetAge, TimeUnit unit) {

return getAgeDecayMultiplier(TimeUnit.SECONDS.convert(tweetAge, unit));

}

/\*\*

\* Compute the age decay, assuming the halflife in the constructor is in minutes.

\* @param ageInSeconds the age in seconds

\*/

public double getAgeDecayMultiplier(long ageInSeconds) {

long minutesSinceTweet = TimeUnit.MINUTES.convert(ageInSeconds, TimeUnit.SECONDS);

return compute(minutesSinceTweet);

}

/\*\*

\* Compute age decay given an age, the age has to be in the same unit as halflife, which you

\* construct the object with.

\*/

public double compute(double age) {

return compute(base, maxBoost, halflife, slope, age);

}

/\*\*

\* Compute the age decay given all parameters. Use this if you don't need to reuse an AgeDecay

\* object.

\*/

public static double compute(

double base, double maxBoost, double halflife, double slope, double age) {

return base + ((maxBoost - base) / (1 + Math.exp(slope \* (age - halflife))));

}

public static double compute(

double base, double maxBoost, double halflife, double age) {

Preconditions.checkArgument(halflife != 0);

return compute(base, maxBoost, halflife, SLOPE\_COEFF / halflife, age);

}

/\*\*

\* Another nicer exponential decay function. Returns a value in (0, 1]

\*/

public static double computeExponential(double halflife, double exp, double age) {

return Math.exp(LN\_HALF \* Math.pow(age, exp) / Math.pow(halflife, exp));

}

/\*\*

\* Exponential decay with remapping of the value from (0,1] to (min,max]

\*/

public static double computeExponential(double halflife, double exp, double age,

double minBoost, double maxBoost) {

double decay = computeExponential(halflife, exp, age); // in (0, 1]

return (maxBoost - minBoost) \* decay + minBoost;

}

}