package com.twitter.search.earlybird.search.relevance;

import java.io.IOException;

import org.apache.lucene.index.LeafReader;

import org.apache.lucene.index.LeafReaderContext;

import org.apache.lucene.search.BooleanClause;

import org.apache.lucene.search.BooleanQuery;

import org.apache.lucene.search.DocIdSetIterator;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.ScoreMode;

import org.apache.lucene.search.Weight;

import com.twitter.search.common.query.DefaultFilterWeight;

import com.twitter.search.common.schema.base.ImmutableSchemaInterface;

import com.twitter.search.core.earlybird.index.EarlybirdIndexSegmentAtomicReader;

import com.twitter.search.core.earlybird.index.util.RangeFilterDISI;

import com.twitter.search.earlybird.search.relevance.scoring.ScoringFunction;

import com.twitter.search.earlybird.search.relevance.scoring.ScoringFunctionProvider;

import com.twitter.search.earlybird.search.relevance.scoring.ScoringFunctionProvider.NamedScoringFunctionProvider;

/\*\*

\* This filter only accepts documents for which the provided

\* {@link com.twitter.search.earlybird.search.relevance.scoring.ScoringFunction}

\* returns a score that's greater or equal to the passed-in minScore and smaller or equal

\* to maxScore.

\*/

public final class ScoreFilterQuery extends Query {

private static final float DEFAULT\_LUCENE\_SCORE = 1.0F;

private final float minScore;

private final float maxScore;

private final NamedScoringFunctionProvider scoringFunctionProvider;

private final ImmutableSchemaInterface schema;

/\*\*

\* Returns a score filter.

\*

\* @param schema The schema to use to extract the feature scores.

\* @param scoringFunctionProvider The scoring function provider.

\* @param minScore The minimum score threshold.

\* @param maxScore The maximum score threshold.

\* @return A score filter with the given configuration.

\*/

public static Query getScoreFilterQuery(

ImmutableSchemaInterface schema,

NamedScoringFunctionProvider scoringFunctionProvider,

float minScore,

float maxScore) {

return new BooleanQuery.Builder()

.add(new ScoreFilterQuery(schema, scoringFunctionProvider, minScore, maxScore),

BooleanClause.Occur.FILTER)

.build();

}

private ScoreFilterQuery(ImmutableSchemaInterface schema,

NamedScoringFunctionProvider scoringFunctionProvider,

float minScore,

float maxScore) {

this.schema = schema;

this.scoringFunctionProvider = scoringFunctionProvider;

this.minScore = minScore;

this.maxScore = maxScore;

}

@Override

public Weight createWeight(IndexSearcher searcher, ScoreMode scoreMode, float boost)

throws IOException {

return new DefaultFilterWeight(this) {

@Override

protected DocIdSetIterator getDocIdSetIterator(LeafReaderContext context) throws IOException {

ScoringFunction scoringFunction = scoringFunctionProvider.getScoringFunction();

scoringFunction.setNextReader((EarlybirdIndexSegmentAtomicReader) context.reader());

return new ScoreFilterDocIdSetIterator(

context.reader(), scoringFunction, minScore, maxScore);

}

};

}

private static final class ScoreFilterDocIdSetIterator extends RangeFilterDISI {

private final ScoringFunction scoringFunction;

private final float minScore;

private final float maxScore;

public ScoreFilterDocIdSetIterator(LeafReader indexReader, ScoringFunction scoringFunction,

float minScore, float maxScore) throws IOException {

super(indexReader);

this.scoringFunction = scoringFunction;

this.minScore = minScore;

this.maxScore = maxScore;

}

@Override

protected boolean shouldReturnDoc() throws IOException {

float score = scoringFunction.score(docID(), DEFAULT\_LUCENE\_SCORE);

return score >= minScore && score <= maxScore;

}

}

public float getMinScoreForTest() {

return minScore;

}

public float getMaxScoreForTest() {

return maxScore;

}

public ScoringFunctionProvider getScoringFunctionProviderForTest() {

return scoringFunctionProvider;

}

@Override

public int hashCode() {

return (int) (minScore \* 29

+ maxScore \* 17

+ (scoringFunctionProvider == null ? 0 : scoringFunctionProvider.hashCode()));

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof ScoreFilterQuery)) {

return false;

}

ScoreFilterQuery filter = ScoreFilterQuery.class.cast(obj);

return (minScore == filter.minScore)

&& (maxScore == filter.maxScore)

&& (scoringFunctionProvider == null

? filter.scoringFunctionProvider == null

: scoringFunctionProvider.equals(filter.scoringFunctionProvider));

}

@Override

public String toString(String field) {

return "SCORE\_FILTER\_QUERY[minScore=" + minScore + ",maxScore=" + maxScore + "]";

}

}