**Code:**

String titleTemp = *title*.get(i);

String descTemp = *description*.get(i);

String narrTemp = *parseNarr*(*narrative*.get(i));

Term termTitle = **new** Term("title", titleTemp);

TermQuery termQueryTitle = **new** TermQuery(termTitle);

SpanFirstQuery spanFirstQueryTitle = **new** SpanFirstQuery(**new** SpanTermQuery(termTitle), 1);

Term termDesc = **new** Term("description", descTemp);

TermQuery termQueryDesc = **new** TermQuery(termDesc);

SpanFirstQuery spanFirstQueryDesc = **new** SpanFirstQuery(**new** SpanTermQuery(termDesc), 1);

Term termNarr = **new** Term("narrative", narrTemp);

TermQuery termQueryNarr = **new** TermQuery(termNarr);

SpanFirstQuery spanFirstQueryNarr = **new** SpanFirstQuery(**new** SpanTermQuery(termNarr), 1);

//Boolean query builder

BooleanQuery.Builder booleanQuery = **new** BooleanQuery.Builder();

booleanQuery.add(**new** BooleanClause(termQueryTitle, BooleanClause.Occur.***MUST***));

booleanQuery.add(**new** BooleanClause(spanFirstQueryTitle, BooleanClause.Occur.***SHOULD***));

booleanQuery.add(**new** BooleanClause(termQueryDesc, BooleanClause.Occur.***MUST***));

booleanQuery.add(**new** BooleanClause(spanFirstQueryDesc, BooleanClause.Occur.***SHOULD***));

booleanQuery.add(**new** BooleanClause(termQueryNarr, BooleanClause.Occur.***MUST***));

booleanQuery.add(**new** BooleanClause(spanFirstQueryNarr, BooleanClause.Occur.***SHOULD***));

// System.out.println(result);

BooleanQuery query = **null**;

query=booleanQuery.build();

// query=parser.parse(result);

*queries*.add(query);

}

My Analyser code:

**package** assignment2;

**import** java.util.\*;

**import** org.apache.lucene.analysis.Analyzer;

**import** org.apache.lucene.analysis.standard.ClassicFilter;

**import** org.apache.lucene.analysis.standard.StandardTokenizer;

**import** org.tartarus.snowball.ext.EnglishStemmer;

**import** org.apache.lucene.analysis.TokenStream;

**import** org.apache.lucene.analysis.core.FlattenGraphFilter;

**import** org.apache.lucene.analysis.core.LowerCaseFilter;

**import** org.apache.lucene.analysis.StopFilter;

**import** org.apache.lucene.analysis.CharArraySet;

**import** org.apache.lucene.analysis.en.PorterStemFilter;

**import** org.apache.lucene.analysis.miscellaneous.TrimFilter;

**import** org.apache.lucene.analysis.miscellaneous.WordDelimiterGraphFilter;

**import** org.apache.lucene.analysis.snowball.SnowballFilter;

**import** org.apache.lucene.analysis.ngram.NGramTokenFilter;

**public** **class** MyAnalyzer **extends** Analyzer{

@Override

**protected** TokenStreamComponents createComponents(String fieldName) {

StandardTokenizer standard = **new** StandardTokenizer();

TokenStream result = **new** ClassicFilter(standard);

// Stop words

List<String> stopWordList = Arrays.*asList*("a", "about", "above", "after", "again", "against", "all", "am", "an", "and", "any", "are", "as", "at", "be", "because", "been", "before", "being", "below", "between", "both", "but", "by", "could", "did", "do", "does", "doing", "down", "during", "each", "few", "for", "from", "further", "had", "has", "have", "having", "he", "he'd", "he'll", "he's", "her", "here", "here's", "hers", "herself", "him", "himself", "his", "how", "how's", "i", "i'd", "i'll", "i'm", "i've", "if", "in", "into", "is", "it", "it's", "its", "itself", "let's", "me", "more", "most", "my", "myself", "nor", "of", "on", "once", "only", "or", "other", "ought", "our", "ours", "ourselves", "out", "over", "own", "same", "she", "she'd", "she'll", "she's", "should", "so", "some", "such", "than", "that", "that's", "the", "their", "theirs", "them", "themselves", "then", "there", "there's", "these", "they", "they'd", "they'll", "they're", "they've", "this", "those", "through", "to", "too", "under", "until", "up", "very", "was", "we", "we'd", "we'll", "we're", "we've", "were", "what", "what's", "when", "when's", "where", "where's", "which", "while", "who", "who's", "whom", "why", "why's", "with", "would", "you", "you'd", "you'll", "you're", "you've", "your", "yours", "yourself", "yourselves");

CharArraySet stopWordSet = **new** CharArraySet( stopWordList, **true**);

// Converting all tokens to lower case

result = **new** LowerCaseFilter(result);

// Trims leading and trailing whitespace from Tokens in the stream.

result = **new** TrimFilter(result);

// Removes stop words from the stream

result = **new** StopFilter(result, stopWordSet);

// Porter Stem filtering for converting words with common stems as they

// tend to have similar meanings.

result = **new** PorterStemFilter(result);

result= **new** NGramTokenFilter(result, 1, 5, **true**);

**return** **new** TokenStreamComponents(standard, result);

}

}

