Stemming

Stemming

The following steps need to be taken to implement Stemming in Nutch. Howie Wang is the person credited with doing so for version 0.7.2. I updated the process for Version 0.8. That can be found below. - Matthew Holt

*** YOU MUST DISABLE THE QUERY-BASIC PLUGIN IN ORDER FOR THIS TO WORK (this replaces all query-basic functionality)***

"I've gotten a couple of questions offlist about stemming so I thought I'd just post here with my changes. Sorry that some of the changes are in the main code and not in a plugin. It seemed that it's more efficient to put in the main analyzer. It would be nice if later releases could add support for plugging in a custom stemmer/analyzer."

(Note by AlessandroGasparini) on the 0.8.1 you could easily enable the Stemming using the multi-language support facilities and without touching the code. (Perhaps you have to write a plugin for your specific language but it's a lot more simple) see by yourself: MultiLingualSupport

Version 0.7.2

*** YOU MUST DISABLE THE QUERY-BASIC PLUGIN IN ORDER FOR THIS TO WORK (this replaces all query-basic functionality)***

The first change I made is in NutchDocumentAnalyzer.java.

Import the following classes at the top of the file:

```
import org.apache.lucene.analysis.LowerCaseTokenizer;
import org.apache.lucene.analysis.LowerCaseFilter;
import org.apache.lucene.analysis.PorterStemFilter;
```

Change tokenStream to:

```
public TokenStream tokenStream(String field, Reader reader) {
   TokenStream ts = CommonGrams.getFilter(new NutchDocumentTokenizer(reader), field);
   if (field.equals("content") || field.equals("title")) {
      ts = new LowerCaseFilter(ts);
      return new PorterStemFilter(ts);
   }
   else {
      return ts;
   }
}
```

The second change is in CommonGrams.java. Import the following classes near the top:

```
import org.apache.lucene.analysis.LowerCaseTokenizer;
import org.apache.lucene.analysis.LowerCaseFilter;
import org.apache.lucene.analysis.PorterStemFilter;
```

In optimizePhrase, after this line:

```
   TokenStream ts = getFilter(new ArrayTokens(phrase), field);
```

Add:

```
   ts = new PorterStemFilter(new LowerCaseFilter(ts));
```

And the rest is a new QueryFilter plugin that I'm calling query-stemmer. Here's the full source for the Java file. You can copy the build.xml and plugin.xml from query-basic, and alter the names for query-stemmer.

```
/* Copyright (c) 2003 The Nutch Organization.  All rights reserved.   */
```

```
/* Use subject to the conditions in http://www.nutch.org/LICENSE.txt. */

package org.apache.nutch.searcher.stemmer;

import org.apache.lucene.search.BooleanQuery;
import org.apache.lucene.search.PhraseQuery;
import org.apache.lucene.search.TermQuery;
import org.apache.lucene.analysis.TokenFilter;
import org.apache.lucene.analysis.TokenStream;
import org.apache.lucene.analysis.Token;
import org.apache.lucene.analysis.LowerCaseTokenizer;
import org.apache.lucene.analysis.LowerCaseFilter;
import org.apache.lucene.analysis.PorterStemFilter;

import org.apache.nutch.analysis.NutchDocumentAnalyzer;
import org.apache.nutch.analysis.CommonGrams;

import org.apache.nutch.searcher.QueryFilter;
import org.apache.nutch.searcher.Query;
import org.apache.nutch.searcher.Query.*;

import java.io.IOException;
import java.util.HashSet;
import java.io.StringReader;

/** The default query filter.  Query terms in the default query field are
* expanded to search the url, anchor and content document fields.*/
public class StemmerQueryFilter implements QueryFilter {

  private static float URL_BOOST = 4.0f;
  private static float ANCHOR_BOOST = 2.0f;

  private static int SLOP = Integer.MAX_VALUE;
  private static float PHRASE_BOOST = 1.0f;

  private static final String[] FIELDS = {"url", "anchor", "content",
"title"};
  private static final float[] FIELD_BOOSTS = {URL_BOOST, ANCHOR_BOOST,
1.0f, 2.0f};

  /** Set the boost factor for url matches, relative to content and anchor
   * matches */
  public static void setUrlBoost(float boost) { URL_BOOST = boost; }

  /** Set the boost factor for title/anchor matches, relative to url and
   * content matches. */
  public static void setAnchorBoost(float boost) { ANCHOR_BOOST = boost; }

  /** Set the boost factor for sloppy phrase matches relative to unordered
term
   * matches. */
  public static void setPhraseBoost(float boost) { PHRASE_BOOST = boost; }

  /** Set the maximum number of terms permitted between matching terms in a
   * sloppy phrase match. */
  public static void setSlop(int slop) { SLOP = slop; }

  public BooleanQuery filter(Query input, BooleanQuery output) {
    addTerms(input, output);
    addSloppyPhrases(input, output);
    return output;
  }

  private static void addTerms(Query input, BooleanQuery output) {
    Clause[] clauses = input.getClauses();
    for (int i = 0; i < clauses.length; i++) {
      Clause c = clauses[i];

      if (!c.getField().equals(Clause.DEFAULT_FIELD))
        continue;                                 // skip non-default fields
```

```
      BooleanQuery out = new BooleanQuery();
      for (int f = 0; f < FIELDS.length; f++) {

        Clause o = c;
        String[] opt;

        // TODO: I'm a little nervous about stemming for all default fields.
        //       Should keep an eye on this.
        if (c.isPhrase()) {                         // optimize phrase
clauses
            opt = CommonGrams.optimizePhrase(c.getPhrase(), FIELDS[f]);
        } else {
            System.out.println("o.getTerm = " + o.getTerm().toString());
            opt = getStemmedWords(o.getTerm().toString());
        }
        if (opt.length==1) {
            o = new Clause(new Term(opt[0]), c.isRequired(),
c.isProhibited());
        } else {
            o = new Clause(new Phrase(opt), c.isRequired(),
c.isProhibited());
        }

        out.add(o.isPhrase()
                ? exactPhrase(o.getPhrase(), FIELDS[f], FIELD_BOOSTS[f])
                : termQuery(FIELDS[f], o.getTerm(), FIELD_BOOSTS[f]),
                false, false);
      }
      output.add(out, c.isRequired(), c.isProhibited());
    }
    System.out.println("query = " + output.toString());
  }

    private static String[] getStemmedWords(String value) {
          StringReader sr = new StringReader(value);
          TokenStream ts = new PorterStemFilter(new LowerCaseTokenizer(sr));

          String stemmedValue = "";
          try {
              Token token = ts.next();
              int count = 0;
              while (token != null) {
                  System.out.println("token = " + token.termText());
                  System.out.println("type = " + token.type());

                  if (count == 0)
                      stemmedValue = token.termText();
                  else
                      stemmedValue = stemmedValue + " " + token.termText();

                  token = ts.next();
                  count++;
              }
          } catch (Exception e) {
              stemmedValue = value;
          }

          if (stemmedValue.equals("")) {
              stemmedValue = value;
          }

          String[] stemmedValues = stemmedValue.split("\\s+");

          for (int j=0; j<stemmedValues.length; j++) {
              System.out.println("stemmedValues = " + stemmedValues[j]);
          }
          return stemmedValues;
    }


  private static void addSloppyPhrases(Query input, BooleanQuery output) {
```

```
    Clause[] clauses = input.getClauses();
    for (int f = 0; f < FIELDS.length; f++) {

      PhraseQuery sloppyPhrase = new PhraseQuery();
      sloppyPhrase.setBoost(FIELD_BOOSTS[f] * PHRASE_BOOST);
      sloppyPhrase.setSlop("anchor".equals(FIELDS[f])
                           ? NutchDocumentAnalyzer.INTER_ANCHOR_GAP
                           : SLOP);
      int sloppyTerms = 0;

      for (int i = 0; i < clauses.length; i++) {
        Clause c = clauses[i];

        if (!c.getField().equals(Clause.DEFAULT_FIELD))
          continue;                               // skip non-default fields

        if (c.isPhrase())                         // skip exact phrases
          continue;

        if (c.isProhibited())                     // skip prohibited terms
          continue;

        sloppyPhrase.add(luceneTerm(FIELDS[f], c.getTerm()));
        sloppyTerms++;
      }

      if (sloppyTerms > 1)
        output.add(sloppyPhrase, false, false);
    }
  }


  private static org.apache.lucene.search.Query
        termQuery(String field, Term term, float boost) {
    TermQuery result = new TermQuery(luceneTerm(field, term));
    result.setBoost(boost);
    return result;
  }

  /** Utility to construct a Lucene exact phrase query for a Nutch phrase.
*/
  private static org.apache.lucene.search.Query
       exactPhrase(Phrase nutchPhrase,
                   String field, float boost) {
    Term[] terms = nutchPhrase.getTerms();
    PhraseQuery exactPhrase = new PhraseQuery();
    for (int i = 0; i < terms.length; i++) {
      exactPhrase.add(luceneTerm(field, terms[i]));
    }
    exactPhrase.setBoost(boost);
    return exactPhrase;
  }

  /** Utility to construct a Lucene Term given a Nutch query term and field.
*/
  private static org.apache.lucene.index.Term luceneTerm(String field,
                                                         Term term) {
    return new org.apache.lucene.index.Term(field, term.toString());
  }
}
```

Version 0.8

(Note by AlessandroGasparini) on the 0.8.1 you could easily enable the Stemming using the multi-language support facilities and without touching the code. (Perhaps you have to write a plugin for your specific language but it's a lot more simple) see by yourself: MultiLingualSupport

*** YOU MUST DISABLE THE QUERY-BASIC PLUGIN IN ORDER FOR THIS TO WORK (this replaces all query-basic functionality)***
The first change I made is in NutchDocumentAnalyzer.java.

Import the following classes at the top of the file:

```
import org.apache.lucene.analysis.LowerCaseTokenizer;
import org.apache.lucene.analysis.LowerCaseFilter;
import org.apache.lucene.analysis.PorterStemFilter;
```

Change tokenStream at the bottom of the file to:

```
public TokenStream tokenStream(String field, Reader reader) {
    Analyzer analyzer;
    if ("anchor".equals(field)) {
        analyzer = ANCHOR_ANALYZER;
    }
    else {
        analyzer = CONTENT_ANALYZER;

        TokenStream ts = analyzer.tokenStream(field, reader);
        if (field.equals("content") || field.equals("title")) {
            ts = new LowerCaseFilter(ts);
            return new PorterStemFilter(ts);
        }
        else {
            return ts;
        }
    }
}
```

The second change is in CommonGrams.java. Import the following classes near the top:

```
import org.apache.lucene.analysis.LowerCaseTokenizer;
import org.apache.lucene.analysis.LowerCaseFilter;
import org.apache.lucene.analysis.PorterStemFilter;
```

In optimizePhrase, after this line:

```
   TokenStream ts = getFilter(new ArrayTokens(phrase), field);
```

Add:

```
   ts = new PorterStemFilter(new LowerCaseFilter(ts));
```

And the rest is a new QueryFilter plugin that I'm calling query-stemmer. Here's the full source for the Java file. You can copy the build.xml and plugin.xml from query-basic, and alter the names for query-stemmer.

{

```
/* Copyright (c) 2003 The Nutch Organization.  All rights reserved.   */
/* Use subject to the conditions in http://www.nutch.org/LICENSE.txt. */

package org.apache.nutch.searcher.stemmer;

import org.apache.hadoop.conf.Configuration;
import org.apache.lucene.search.BooleanClause;
import org.apache.lucene.search.BooleanQuery;
import org.apache.lucene.search.PhraseQuery;
import org.apache.lucene.search.TermQuery;
import org.apache.lucene.analysis.TokenFilter;
import org.apache.lucene.analysis.TokenStream;
import org.apache.lucene.analysis.Token;
import org.apache.lucene.analysis.LowerCaseTokenizer;
import org.apache.lucene.analysis.LowerCaseFilter;
import org.apache.lucene.analysis.PorterStemFilter;

import org.apache.nutch.analysis.NutchDocumentAnalyzer;
import org.apache.nutch.analysis.CommonGrams;
```

```
import org.apache.nutch.searcher.QueryFilter;
import org.apache.nutch.searcher.Query;
import org.apache.nutch.searcher.Query.*;

import java.io.IOException;
import java.util.HashSet;
import java.io.StringReader;

/**
 * The default query filter. Query terms in the default query field are expanded
 * to search the url, anchor and content document fields.
 */
public class StemmerQueryFilter implements QueryFilter {
        private static int SLOP = Integer.MAX_VALUE;

        private float PHRASE_BOOST = 1.0f;

        private static final String[] FIELDS = { "url", "anchor", "content",
                        "title" , "host" };

        private final float[] FIELD_BOOSTS = { 4.0f, 2.0f, 1.0f, 1.5f, 2.0f };

        private Configuration conf;

        /**
         * Set the boost factor for url matches, relative to content and anchor
         * matches
         */
        public void setUrlBoost(float boost) {
                FIELD_BOOSTS[0] = boost;
        }

        /**
         * Set the boost factor for title/anchor matches, relative to url and
         * content matches.
         */
        public void setAnchorBoost(float boost) {
                FIELD_BOOSTS[1] = boost;
        }

        /**
         * Set the boost factor for sloppy phrase matches relative to unordered term
         * matches.
         */
        public void setPhraseBoost(float boost) {
                PHRASE_BOOST = boost;
        }

        /**
         * Set the maximum number of terms permitted between matching terms in a
         * sloppy phrase match.
         */
        public static void setSlop(int slop) {
                SLOP = slop;
        }

        public BooleanQuery filter(Query input, BooleanQuery output) {
                addTerms(input, output);
                addSloppyPhrases(input, output);
                return output;
        }

        private void addTerms(Query input, BooleanQuery output) {
                Clause[] clauses = input.getClauses();
                for (int i = 0; i < clauses.length; i++) {
                        Clause c = clauses[i];

                        if (!c.getField().equals(Clause.DEFAULT_FIELD))
                                continue; // skip non-default fields
```

```
                BooleanQuery out = new BooleanQuery();
                for (int f = 0; f < FIELDS.length; f++) {

                        Clause o = c;
                        String[] opt;

                        // TODO: I'm a little nervous about stemming for all default
                        // fields.
                        // Should keep an eye on this.
                        if (c.isPhrase()) { // optimize phrase clauses
                                opt = new CommonGrams(getConf()).optimizePhrase(c
                                                .getPhrase(), FIELDS[f]);
                        } else {
                                System.out.println("o.getTerm = " + o.getTerm().toString());
                                opt = getStemmedWords(o.getTerm().toString());
                        }
                        if (opt.length == 1) {
                                o = new Clause(new Term(opt[0]), c.isRequired(), c
                                                .isProhibited(), getConf());
                        } else {
                                o = new Clause(new Phrase(opt), c.isRequired(), c
                                                .isProhibited(), getConf());
                        }

                        out.add(o.isPhrase() ? exactPhrase(o.getPhrase(), FIELDS[f],
                                        FIELD_BOOSTS[f]) : termQuery(FIELDS[f], o.getTerm(),
                                        FIELD_BOOSTS[f]), BooleanClause.Occur.SHOULD);
                }
                output.add(out, (c.isProhibited() ? BooleanClause.Occur.MUST_NOT
                                : (c.isRequired() ? BooleanClause.Occur.MUST
                                                : BooleanClause.Occur.SHOULD)));
        }
        System.out.println("query = " + output.toString());
}

private static String[] getStemmedWords(String value) {
        StringReader sr = new StringReader(value);
        TokenStream ts = new PorterStemFilter(new LowerCaseTokenizer(sr));

        String stemmedValue = "";
        try {
                Token token = ts.next();
                int count = 0;
                while (token != null) {
                        System.out.println("token = " + token.termText());
                        System.out.println("type = " + token.type());

                        if (count == 0)
                                stemmedValue = token.termText();
                        else
                                stemmedValue = stemmedValue + " " + token.termText();

                        token = ts.next();
                        count++;
                }
        } catch (Exception e) {
                stemmedValue = value;
        }

        if (stemmedValue.equals("")) {
                stemmedValue = value;
        }

        String[] stemmedValues = stemmedValue.split("\\s+");

        for (int j = 0; j < stemmedValues.length; j++) {
                System.out.println("stemmedValues = " + stemmedValues[j]);
        }
        return stemmedValues;
}
```

```
        private void addSloppyPhrases(Query input, BooleanQuery output) {
                Clause[] clauses = input.getClauses();
                for (int f = 0; f < FIELDS.length; f++) {

                        PhraseQuery sloppyPhrase = new PhraseQuery();
                        sloppyPhrase.setBoost(FIELD_BOOSTS[f] * PHRASE_BOOST);
                        sloppyPhrase
                                        .setSlop("anchor".equals(FIELDS[f]) ? NutchDocumentAnalyzer.
INTER_ANCHOR_GAP
                                                        : SLOP);
                        int sloppyTerms = 0;

                        for (int i = 0; i < clauses.length; i++) {
                                Clause c = clauses[i];

                                if (!c.getField().equals(Clause.DEFAULT_FIELD))
                                        continue; // skip non-default fields

                                if (c.isPhrase()) // skip exact phrases
                                        continue;

                                if (c.isProhibited()) // skip prohibited terms
                                        continue;

                                sloppyPhrase.add(luceneTerm(FIELDS[f], c.getTerm()));
                                sloppyTerms++;
                        }

                        if (sloppyTerms > 1)
                                output.add(sloppyPhrase, BooleanClause.Occur.SHOULD);
                }
        }

        private static org.apache.lucene.search.Query termQuery(String field,
                        Term term, float boost) {
                TermQuery result = new TermQuery(luceneTerm(field, term));
                result.setBoost(boost);
                return result;
        }

        /**
         * Utility to construct a Lucene exact phrase query for a Nutch phrase.
         */
        private static org.apache.lucene.search.Query exactPhrase(
                        Phrase nutchPhrase, String field, float boost) {
                Term[] terms = nutchPhrase.getTerms();
                PhraseQuery exactPhrase = new PhraseQuery();
                for (int i = 0; i < terms.length; i++) {
                        exactPhrase.add(luceneTerm(field, terms[i]));
                }
                exactPhrase.setBoost(boost);
                return exactPhrase;
        }

        /**
         * Utility to construct a Lucene Term given a Nutch query term and field.
         */
        private static org.apache.lucene.index.Term luceneTerm(String field,
                        Term term) {
                return new org.apache.lucene.index.Term(field, term.toString());
        }

        public void setConf(Configuration conf) {
                this.conf = conf;
                this.FIELD_BOOSTS[0] = conf.getFloat("query.url.boost", 4.0f);
                this.FIELD_BOOSTS[1] = conf.getFloat("query.anchor.boost", 2.0f);
                this.FIELD_BOOSTS[2] = conf.getFloat("query.content.boost", 1.0f);
                this.FIELD_BOOSTS[3] = conf.getFloat("query.title.boost", 1.5f);
                this.FIELD_BOOSTS[4] = conf.getFloat("query.host.boost", 2.0f);
                this.PHRASE_BOOST = conf.getFloat("query.phrase.boost", 1.0f);
        }
```

```
        public Configuration getConf() {
                return this.conf;
        }
}
```