Information Access with Apache Lucene

Metodi per il Ritrovamento dell'Informazione

Laurea Triennale in Informatica

Università degli Studi di Bari Aldo Moro

Prof. Cataldo Musto

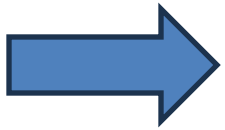
cataldo.musto@uniba.it

Credits: Marco Polignano

Code Repository & Requirements

Code repository

https://github.com/swapUniba/MRI_2024_25



Requirements

- Java SDK 1.8+ https://www.java.com/en/download/
- IDE: NetBeans, IntelliJ, Eclipse, …
- **Maven:** https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

Lucene Search

- **IndexReader:** interface for accessing an index
- **QueryParser:** parses the user query

- **IndexSearcher:** implements search over a single IndexReader
  - `search(Query query, int numDoc)`
  - TopDocs -> result of search
    - `TopDocs.scoreDocs` returns an array of retrieved documents (ScoreDoc)

Lucene Search

See the class `di.uniba.it.mri2324.lucene.TestSearch1`

```
//Open Index and create a Searcher
FSDirectory fsdir = FSDirectory.open(new File("./resources/alice").toPath());
IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));

//Single term query
//Query q = new TermQuery(new Term("chapter", "rabbit"));

//Boolean query
BooleanQuery.Builder qb = new BooleanQuery.Builder();
qb.add(new TermQuery(new Term("chapter", "rabbit")),
                                    BooleanClause.Occur.SHOULD);
qb.add(new TermQuery(new Term("chapter_text", "alice")),
                                    BooleanClause.Occur.SHOULD);
Query q = qb.build();

//Search Results
TopDocs topdocs = searcher.search(q, 10);
System.out.println("Found " + topdocs.totalHits.value + " document(s).");
```

Lucene Search

See the class `di.uniba.it.mri2324.lucene.TestSearch1`

```
//Open Index and create a Searcher
FSDirectory fsdir = FSDirectory.open(new File("./resources/alice").toPath());
IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));

//Single term query
//Query q = new TermQuery(new Term("chapter", "rabbit"));

//Boolean query
BooleanQuery.Builder qb = new BooleanQuery.Builder();
qb.add(new TermQuery(new Term("chapter", "rabbit")),
                                    BooleanClause.Occur.SHOULD);
qb.add(new TermQuery(new Term("chapter_text", "alice")),
                                    BooleanClause.Occur.SHOULD);
Query q = qb.build();

//Search Results
TopDocs topdocs = searcher.search(q, 10);
System.out.println("Found " + topdocs.totalHits.value + " document(s).");
```

We open the index we previously created

Lucene Search

See the class `di.uniba.it.mri2324.lucene.TestSearch1`

```
//Open Index and create a Searcher
FSDirectory fsdir = FSDirectory.open(new File("./resources/alice").toPath());
IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));

//Single term query
//Query q = new TermQuery(new Term("chapter", "rabbit"));

//Boolean query
BooleanQuery.Builder qb = new BooleanQuery.Builder();
qb.add(new TermQuery(new Term("chapter", "rabbit")),
                                    BooleanClause.Occur.SHOULD);
qb.add(new TermQuery(new Term("chapter_text", "alice")),
                                    BooleanClause.Occur.SHOULD);
Query q = qb.build();

//Search Results
TopDocs topdocs = searcher.search(q, 10);
System.out.println("Found " + topdocs.totalHits.value + " document(s).");
```

SHOULD=OR
MUST=AND

Lucene QueryParser

- Example:
  - `queryParser.parse("chapter:Rabbit");`

- good human entered queries, debugging
- does text analysis and constructs appropriate queries
- not all query types supported

QUERY SYNTAX 1/2

- **Wildcard Searches**
  - tes* (test - tests - tester)
  - te?t (test – text)
  - te*t (tempt)
  - tes?
- **Fuzzy Searches (Levenshtein Distance) (TERM)**
  - roam~ (foam – roams)
  - roam~0.8
- **Range Searches**
  - mod_date:[20020101 TO 20030101]
  - title:{Aida TO Carmen}
- **Proximity Searches (PHRASE)**
  - "jakarta apache"~10

QUERY SYNTAX 2/2

- **Boosting a Term**
  - jakarta^4 apache
  - "jakarta apache"^4 "Apache Lucene”
- **Boolean Operator**
  - NOT, OR, AND
  - + required operator
    title:(+return +"pink panther")
  - - prohibit operator
  - Escaping Special Characters by \

DELETING DOCUMENTS

IndexWriter

- `deleteDocuments(Term… terms)`
- `deleteDocuments(Query… queries)`
- `updateDocument(Term term, Iterable<? extends IndexableField> doc)`
  Updates a document by first deleting the document(s) containing term and then adding the new document.
- `updateDocuments(Term delTerm, Iterable<? extends Iterable<? extends IndexableField>> docs)`
  Deletes and adds a block of documents with sequentially assigned document IDs, such that an external reader will see all or none of the documents.
- deleting does not immediately reclaim space

Search Engine - Home Work

- Create your own search engine
- Collect (or crawl) enough text documents containing information you want to index (i.e., sports news)
- **Optional:** define the fields and index them separately
- Index the content and make some queries

Develop a “little” search engine

FIRST PROJECT

Exercise – Search Engine

Build a “little” search engine that indexes and searches text files into a folder

- main class `IndexSE` takes the folder name and the index directory name as arguments
  - the class indexes all the “.txt” into the folder
- main class `SearchSE` takes the index directory name and the query as arguments
  - the class searches the index

Exercise – Search Engine

See classes:

- `di.uniba.it.mri2324.lucene.se.IndexSE`
- `di.uniba.it.mri2324.lucene.se.SearchSE`

IndexSE

```
FSDirectory fsdir = FSDirectory.open(new File(args[1]).toPath());
//creiamo l'indice
IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer());
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
IndexWriter writer = new IndexWriter(fsdir, iwc);
File[] files = dir.listFiles();

for (File file : files) {
   if (file.isFile() && file.getName().endsWith(".txt")) {
       Document doc = new Document();
       doc.add(new StringField("id", file.getAbsolutePath(),
                   Field.Store.YES));
       doc.add(new TextField("text", new FileReader(file)));
       writer.addDocument(doc);
    }
}
writer.close();
```

IndexSE

Passiamo come parametro la directory

```
FSDirectory fsdir = FSDirectory.open(new File(args[1]).toPath());
//creiamo l'indice
IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer());
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
IndexWriter writer = new IndexWriter(fsdir, iwc);
File[] files = dir.listFiles();

for (File file : files) {
   if (file.isFile() && file.getName().endsWith(".text"))
       { Document doc = new Document();
       doc.add(new StringField("id", file.getAbsolutePath(),
                   Field.Store.YES));
       doc.add(new TextField("text", new FileReader(file)));
       writer.addDocument(doc);
    }
}
writer.close();
```

IndexSE

```
FSDirectory fsdir = FSDirectory.open(new File(args[1]).toPath());
//creiamo l'indice
IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer());
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
IndexWriter writer = new IndexWriter(fsdir, iwc);
File[] files = dir.listFiles();

for (File file : files) {
   if (file.isFile() && file.getName().endsWith(".text"))
       { Document doc = new Document();
       doc.add(new StringField("id", file.getAbsolutePath(),
                   Field.Store.YES));
       doc.add(new TextField("text", new FileReader(file)));
       writer.addDocument(doc);
    }
}
writer.close();
```

Prendiamo tutti i .txt della directory

IndexSE

```
FSDirectory fsdir = FSDirectory.open(new File(args[1]).toPath());
//creiamo l'indice
IndexWriterConfig iwc = new IndexWriterConfig(new StandardAnalyzer());
iwc.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
IndexWriter writer = new IndexWriter(fsdir, iwc);
File[] files = dir.listFiles();

for (File file : files) {
   if (file.isFile() && file.getName().endsWith(".txt")) {
       Document doc = new Document();
       doc.add(new StringField("id", file.getAbsolutePath(),
                   Field.Store.YES));
       doc.add(new TextField("text", new FileReader(file)));
       writer.addDocument(doc);
    }
}
writer.close();
```

Mettiamo **tutto il contenuto del file** di testo in un campo.

SearchSE

```
FSDirectory fsdir = FSDirectory.open(new File(args[0]).toPath());

IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));
QueryParser qp = new QueryParser("text", new StandardAnalyzer());

//Parse the query
Query q = qp.parse(args[1]);
//Search
TopDocs topdocs = searcher.search(q, 10);
 for (ScoreDoc sdoc : topdocs.scoreDocs)
{
   searcher.doc(sdoc.doc).get("id")
   System.out.println("Found doc, path=" + ", score" + sdoc.score);

}
```

SearchSE

Primo Parametro, directory dell'indice

```
FSDirectory fsdir = FSDirectory.open(new File(args[0]).toPath());

IndexSearcher searcher = new IndexSearcher(DirectoryReader.open(fsdir));
QueryParser qp = new QueryParser("text", new StandardAnalyzer());

//Parse the query
Query q = qp.parse(args[1]);
//Search
TopDocs topdocs = searcher.search(q, 10);
 for (ScoreDoc sdoc : topdocs.scoreDocs)
{
   searcher.doc(sdoc.doc).get("id")
   System.out.println("Found doc, path=" + ", score" + sdoc.score);

}
```

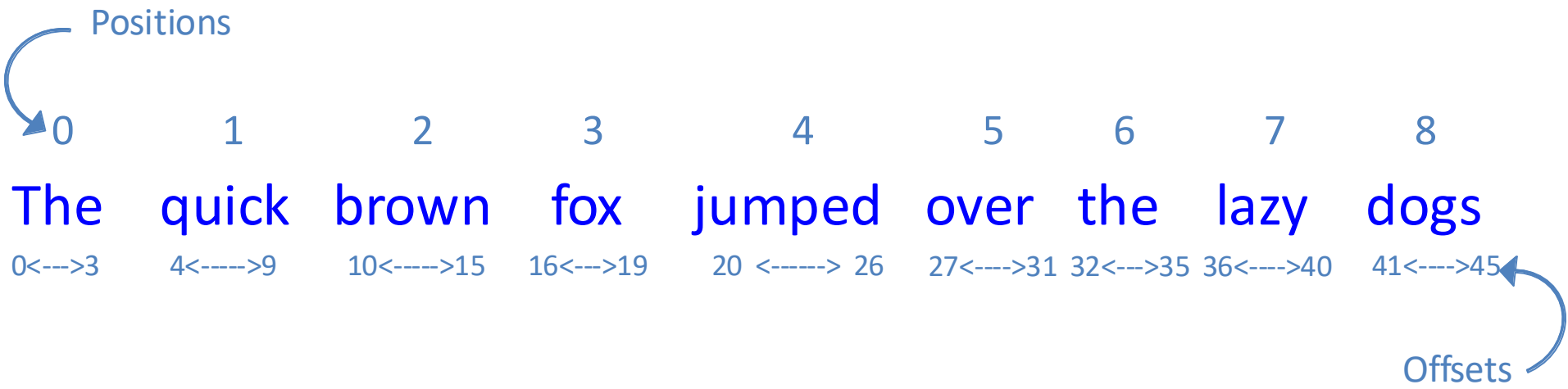
Secondo parametro, la query

POSTING API

Posting API

- Getting access to term **frequency, positions and offsets**
- Required for:
  - Relevance Feedback and “More Like This”
  - Clustering
  - Similarity between two documents
  - Highlighter
    - needs offsets info

Basics: Positions and Offsets



Posting API

- `Fields` is the initial entry point into the postings APIs

- `Terms` represents the collection of terms within a field
- `TermsEnum` provides an iterator over the list of terms within a field

- `PostingsEnum` is an extension of `DocIdSetIterator` that iterates over the list of documents for a term, along with the term frequency within that document

- `PostingsEnum` also allows iteration of the positions a term occurred within the document, and any **additional per-position information (offsets and payload)**

Store Posting 1/2

- During indexing, create a `Field` that stores TermVectors through `FieldTyp` (we used TextField as a predefined IndexableField)
  - `FieldType ft = new FieldType(TextField.TYPE_NOT_STORED);`
    - `TYPE_STORED|TYPE_NOT_STORED`
- Invoke in cascade the following methods to set the field type:
  - `ft.setTokenized(true); //done as default`
  - `ft.setStoreTermVectors(true);`
  - `ft.setStoreTermVectorPositions(true);`
  - `ft.setStoreTermVectorOffsets(true);`
  - `ft.setStoreTermVectorPayloads(true);`

Store Posting 2/2

See class: `di.uniba.it.mri2324.lucene.se.post.IndexPostExample`

```
//define a custom field type that stores post information
FieldType ft = new FieldType(TextField.TYPE_NOT_STORED);
ft.setStoreTermVectors(true);
ft.setStoreTermVectorPositions(true);
ft.setStoreTermVectorOffsets(true);
…
Document doc1 = new Document();
doc1.add(new StringField("id", "1", Field.Store.YES));
doc1.add(new Field("text", new FileReader("./resources/text/es1.txt"), ft));
writer.addDocument(doc1);
```

Store Posting 2/2

See class: `di.uniba.it.mri2324.lucene.se.post.IndexPostExample`

```
//define a custom field type that stores post information
FieldType ft = new FieldType(TextField.TYPE_NOT_STORED);
ft.setStoreTermVectors(true);
ft.setStoreTermVectorPositions(true);
ft.setStoreTermVectorOffsets(true);
…
Document doc1 = new Document();
doc1.add(new StringField("id", "1", Field.Store.YES));
doc1.add(new Field("text", new FileReader("./resources/text/es1.txt"), ft));
writer.addDocument(doc1);
```

Il nuovo campo viene memorizzato **utilizzando il formato appena definito.**

Posting API

- Getting the BoW of a document: di.uniba.it.mri2324.lucene.se.post.PostExample

```
Fields fields = ireader.getTermVectors(docid);

for (String field : fields) {
      Terms terms = fields.terms(field);
      TermsEnum termsEnum = terms.iterator();
       BytesRef term = null;
       while ((term = termsEnum.next()) != null)
            {
            System.out.print(term.utf8ToString());
            PostingsEnum postings = termsEnum.postings(null,PostingsEnum.FREQS);

            while (postings.nextDoc() != DocIdSetIterator.NO_MORE_DOCS) {
                System.out.println(":" + postings.freq());
            }
      }
}
```

More info

https://lucene.apache.org/core/8_6_2/core/org/apache/lucene/index/package-summary.html#postings

Posting API

- Getting the BoW of a document:
  di.uniba.it.mri2324.lucene.se.post.PostExample

```
Fields fields = ireader.getTermVectors(docid);

for (String field : fields) {
      Terms terms = fields.terms(field);
      TermsEnum termsEnum = terms.iterator();
       BytesRef term = null;
       while ((term = termsEnum.next()) != null)
            {
            System.out.print(term.utf8ToString());
            PostingsEnum postings = termsEnum.postings(null,PostingsEnum.FREQS);

            while (postings.nextDoc() != DocIdSetIterator.NO_MORE_DOCS) {
                System.out.println(":" + postings.freq());
            }
      }
}
```

ID del documento

More info
https://lucene.apache.org/core/8_6_2/core/org/apache/lucene/index/package-summary.html#postings

Posting API

- Getting the BoW of a document: `di.uniba.it.mri2324.lucene.se.post.PostExample`

```
Fields fields = ireader.getTermVectors(docid);

for (String field : fields) {
      Terms terms = fields.terms(field);
      TermsEnum termsEnum = terms.iterator();
       BytesRef term = null;
       while ((term = termsEnum.next()) != null)
            {
            System.out.print(term.utf8ToString());
            PostingsEnum postings = termsEnum.postings(null,PostingsEnum.FREQS);

            while (postings.nextDoc() != DocIdSetIterator.NO_MORE_DOCS) {
                System.out.println(":" + postings.freq());
            }
      }
}
```

Accediamo alle frequenze dei termini dentro `termsEnum`

More info
https://lucene.apache.org/core/8_6_2/core/org/apache/lucene/index/package-summary.html#postings

Posting API

- Getting the BoW of a document: `di.uniba.it.mri2324.lucene.se.post.PostExample`

```
Fields fields = ireader.getTermVectors(docid);

for (String field : fields) {
      Terms terms = fields.terms(field);
      TermsEnum termsEnum = terms.iterator();
       BytesRef term = null;
       while ((term = termsEnum.next()) != null)
            {
            System.out.print(term.utf8ToString());
            PostingsEnum postings = termsEnum.postings(null,PostingsEnum.ALL);

            while (postings.nextDoc() != DocIdSetIterator.NO_MORE_DOCS) {
                System.out.println(":" + postings.freq());

                for (int i = 0; i < postings.freq(); i++) {
                             int position = postings.nextPosition();
                             System.out.println("\t" + position + "\t{" +
                            postings.startOffset() + ", " + postings.endOffset()
                + "}");}}}}
```

Accediamo a tutte le informazioni sui termini (es. la posizione) dentro `termsEnum`

Posting APIs - Home Work

- **Integrate** Posting APIs with the search engine previously created

- **Adapt the indexing process**, by using Posting APIs to **store offsets and positions** (create a new field, for example for the text of the chapter in Alice in Wonderland)

- **Pick the top-1 result for your query**, and look for the terms of the query in the text, and return **the position of the terms**.

Implement an evaluation pipeline using the cranfield paradigm

Evaluation

Exercise 3

- Implement an evaluation pipeline
- Given
  - A test collection
  - Relevance assessments
  - Evaluation Metrics

  Find out the best indexing/searching configurations
- Experiment with
  - Analyzers
  - Query construction
  - Field structuring
  - Term/field boosting

Test collection

- All files are in ./resources/cran
- Cranfield Collection: collection of abstracts
- **cran.all.1400.json**: documents collection in JSON
  - fields: *id, text, authors, title, biblio*
- **cran.qry.json**: topics (queries) in JSON
  - fields: *id, query*
- See `di.uniba.it.mri2324.lucene.cran.HowToUseGson` for how to read JSON files
- **cranqrel:** relevance judgments
  - a map between a query and the set of relevant documents
    <qid> <run_num> <docid> <relevance>

Pipeline

- Index the documents collection
- Implement an evaluation class that for each topic (query) stores the top 100 retrieved documents
  - see the output format in the next slide
- Try different analyzers, queries formulation, term boosting … and measure the performance of your pipeline

The output format

- Output format, six fields separated by a space char:
  - query_id
  - run_identifier: may be every number (it is used to keep reference to the experiment)
  - doc_id
  - doc_rank
  - doc_score
  - exp_name: exp_name is a short reference string to the experiment
- `example.out` contains an output example

Trec_eval

- Program for running the evaluation
- Outputs many common evaluation measures:
  - Precision/recall
  - Number of [relevant] document retrieved
  - MAP, GMAP, P@K
- How to execute
  - `trec_eval` relevance_judgment_file output_file
- See: http://trec.nist.gov/trec_eval/index.html
- Versions already compiled for Linux, Mac and Win are in ./resources/trec_eval

Relevance feedback

Documents similarity

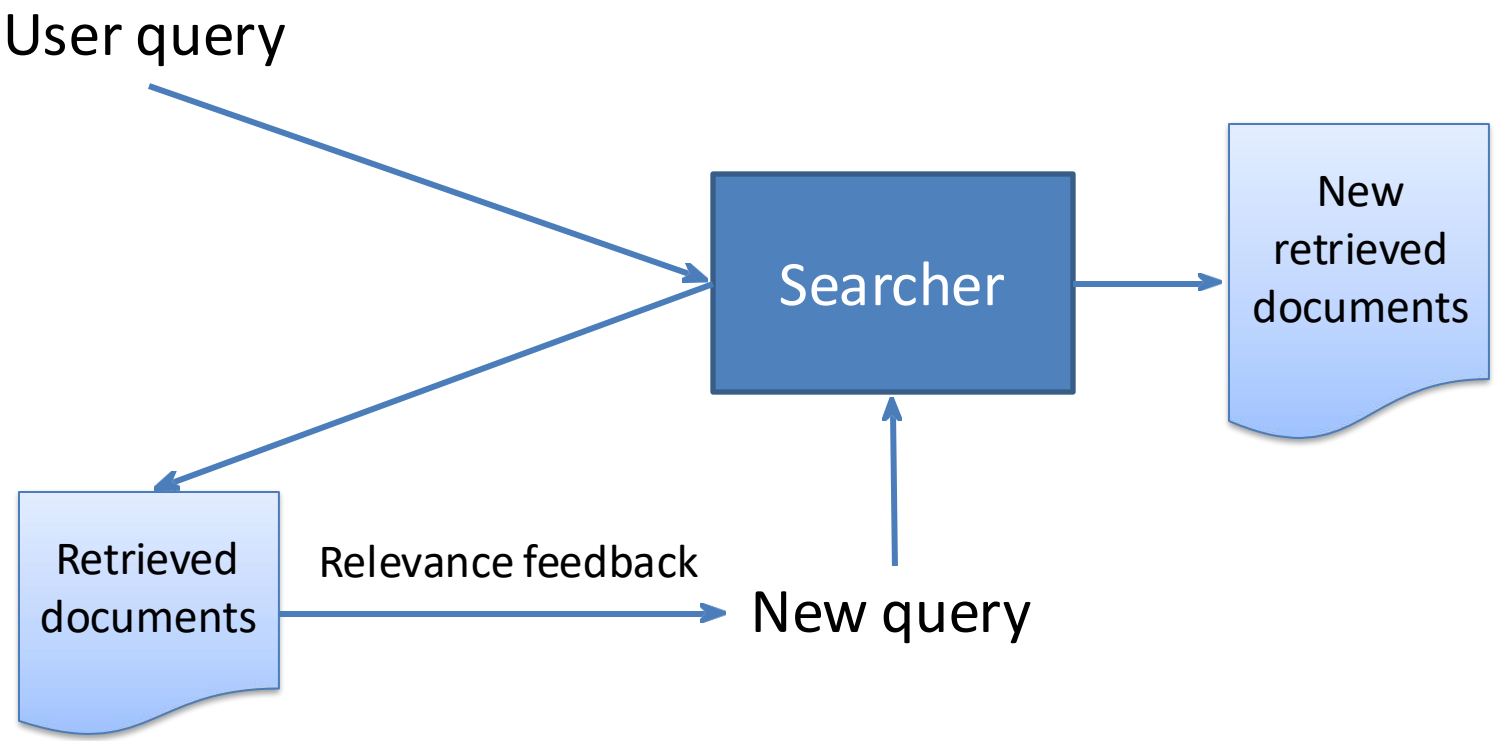
Apache Tika

ADVANCED TOPICS

Relevance feedback

- Improve retrieval performance using information about document relevance
  - **Explicit:** relevance indicated by the user
  - **Implicit:** from user behavior
  - **Blind (or pseudo):** using information about the top *k* retrieved documents

Relevance feedback



Rocchio Algorithm

$$Q_{new} = \alpha \cdot Q + \beta \cdot \frac{1}{|D_{rel}|} \cdot \sum_{D_j \in D_{rel}} D_j - \delta \cdot \frac{1}{|D_{norel}|} \cdot \sum_{D_k \in D_{norel}} D_k$$

Original query

Relevant document

Non-relevant document

Q_{new}, Q, D_j, D_i are vectors: BoW of a query or a document

Rocchio Algorithm (blind)

$$Q_{new} = \alpha \cdot Q + \beta \cdot \frac{1}{|D_{rel}|} \cdot \sum_{D_j \in D_{rel}} D_j - \delta \cdot \frac{1}{|D_{norel}|} \cdot \sum_{D_k \in D_{norel}} D_k$$

In blind (pseudo) relevance feedback we know only relevant documents (supposed to be the top *K* documents)

(generally adopted by search engine)

Relevance feedback

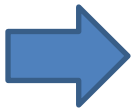
Q = Alice

It's a friend of mine — a Cheshire Cat,' said Alice: 'allow me to introduce it.

It's the oldest rule in the book,' said the King. 'Then it ought to be Number One,' said Alice

Alice looked round eagerly, and found that it was the Red Queen. 'She's grown a good deal!' was her first remark.

Alice lay back, and closed her eyes. There was the Red Queen again, with that incessant grin. Or was it the Cheshire cat's grin?



Q_{new} = Alice **Cheshire Cat King book**

It's a friend of mine — a Cheshire Cat,' said Alice: 'allow me to introduce it.

It's the oldest rule in the book,' said the King. 'Then it ought to be Number One,' said Alice

Alice book was reprinted and published in 1866.

…

(pseudo) relevance feedback in Lucene

Excercise 4

Relevance feedback in Lucene

- Possible using term vector (e.g. TermsEnum)
  - Retrieve the top *K* documents
  - Build the Q_{new} using term vector from the *K* documents
  - Re-query using Q_{new}
- Suggestions:
  - Map<String, Integer> for representing BoW as sparse vectors

Document similarity

Excercise 5

Document similarity

- Documents are represented by vectors
  - Build the document vector using TermsEnum
  - Compute the similarity using cosine similarity
- Implement a functionality like “similar to this document”
- Suggestions:
  - Map<String, Integer> for representing BoW as sparse vectors
  - Computes cosine similarity between the BoW of two Documents

CONCLUSIONS

Conclusions

- A popular IR model
  - Vector Space Model
  - Lucene supports other IR models: BM25, Language Modeling, ...
- Lucene
  - provides API to build search engine
- Apace Tika
  - extracts metadata and text from files and URLs
- **LET'S GO TO BUILD YOUR SEARCH ENGINE!**

Lucene API

- core: lucene core library
- analyzers-common: analyzers for indexing content in different languages and domains
  - Arabic, Chinese, Italian, …
- queryparser: query parser and parsing framework
- highlighter: a set of classes for highlighting matching terms in search results
- suggest: auto-suggest and spellchecking support

Lucene related project

- Apache Solr: open source enterprise search platform from the Apache Lucene
  - Full-Text Search Capabilities, High Volume Web Traffic, HTML Administration Interfaces

- Apache Nutch: web-search engine based on Solr and Lucene
  - crawler, link-graph database, HTML