

Advanced Lucene

Grant Ingersoll
Center for Natural Language Processing
ApacheCon 2005
December 12, 2005

Overview

- CNLP Intro
- Quick Lucene Refresher
- Term Vectors
- Span Queries
- Building an IR Framework
- Case Studies from CNLP
 - Collection analysis for domain specialization
 - Crosslingual/multilingual retrieval in Arabic, English and Dutch
 - Sublanguage analysis for commercial trouble ticket analysis
 - Passage retrieval and analysis for Question Answering application
- Resources

Center for Natural Language Processing

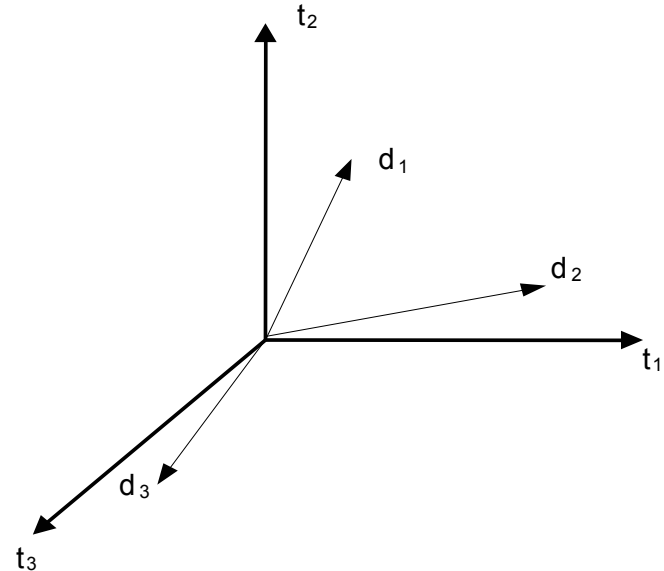
- Self-supporting university research and development center
- Experienced, multidisciplinary team of information scientists, computer scientists, and linguists
 - Many with substantial commercial software experience
- Research, development & licensing of NLP-based technology for government & industry
 - ARDA, DARPA, NSA, CIA, DHS, DOJ, NIH
 - Raytheon, SAIC, Boeing, ConEd, MySentient, Unilever
- For numerous applications:
 - Document Retrieval
 - Question-Answering
 - Information Extraction / Text Mining
 - Automatic Metadata Generation
 - Cross-Language Information Retrieval
- 60+ projects in past 6 years

Lucene Refresher

- Indexing
 - A Document is a collection of Fields
 - A Field is free text, keywords, dates, etc.
 - A Field can have several characteristics
 - indexed, tokenized, stored, term vectors
 - Apply Analyzer to alter Tokens during indexing
 - Stemming
 - Stopword removal
 - Phrase identification

Lucene Refresher

- Searching
 - Uses a modified Vector Space Model (VSM)
 - We convert a user's query into an internal representation that can be searched against the Index
 - Queries are usually analyzed in the same manner as the index
 - Get back `HITS` and use in an application



Vector Space Model

$$d_j = \langle t_{1j}, t_{2j}, t_{3j}, \dots, t_{mj} \rangle$$

Lucene Demo++

- Based on original Lucene demo
- Sample, working code for:
 - Search
 - Relevance Feedback
 - Span Queries
 - Collection Analysis
 - Candidate Identification for QA
- Requires: Lucene 1.9 RC1-dev
- Available at <http://www.cnlp.org/apachecon2005>

Term Vectors

- Relevance Feedback and “More Like This”
- Domain specialization
- Clustering
- Cosine similarity between two documents
- Highlighter
 - Needs offset info

Lucene Term Vectors (TV)

- In Lucene, a `TermFreqVector` is a representation of all of the terms and term counts in a specific `Field` of a `Document` instance
- As a tuple:

```
termFreq = <term, term countD>  
          <fieldName, <..., termFreqi, termFreqi+1, ...>>
```

- As Java:

```
public String getField();  
public String[] getTerms();  
public int[] getTermFrequencies(); } Parallel Arrays
```

Creating Term Vectors

- During indexing, create a `Field` that stores Term Vectors:

```
new Field("title", parser.getTitle(),
        Field.Store.YES,
        Field.Index.TOKENIZED,
        Field.TermVector.YES);
```

- Options are:

```
Field.TermVector.YES
```

```
Field.TermVector.NO
```

```
Field.TermVector.WITH_POSITIONS – Token Position
```

```
Field.TermVector.WITH_OFFSETS – Character offsets
```

```
Field.TermVector.WITH_POSITIONS_OFFSETS
```

Accessing Term Vectors

- Term Vectors are acquired from the IndexReader using:

```
TermFreqVector getTermFreqVector(int docNumber,  
                                  String field)
```

```
TermFreqVector[] getTermFreqVectors(int docNumber)
```

- Can be cast to `TermPositionVector` if the vector was created with offset or position information

- `TermPositionVector` API:

```
int[] getTermPositions(int index);
```

```
TermVectorOffsetInfo [] getOffsets(int index);
```

Relevance Feedback

- Expand the original query using terms from documents
- Manual Feedback:
 - User selects which documents are most relevant and, optionally, non-relevant
 - Get the terms from the term vector for each of the documents and construct a new query
 - Can apply boosts based on term frequencies
- Automatic Feedback
 - Application assumes the top X documents are relevant and the bottom Y are non-relevant and constructs a new query based on the terms in those documents
- See *Modern Information Retrieval* by Baeza-Yates, et. al. for in-depth discussion of feedback

Example

- From Demo, SearchServlet.java
- Code to get the top X terms from a TV

```
protected Collection getTopTerms(TermFreqVector tfv, int
                                numTermsToReturn)
{
    String[] terms = tfv.getTerms();//get the terms
    int [] freqs = tfv.getTermFrequencies();//get the frequencies
    List result = new ArrayList(terms.length);
    for (int i = 0; i < terms.length; i++)
    {
        //create a container for the Term and Frequency information
        result.add(new TermFreq(terms[i], freqs[i]));
    }
    Collections.sort(result, comparator);//sort by frequency
    if (numTermsToReturn < result.size())
    {
        result = result.subList(0, numTermsToReturn);
    }
    return result;
}
```

Span Queries

- Provide info about where a match took place within a document
- `SpanTermQuery` is the building block for more complicated queries
- **Other `SpanQuery` classes:**
 - `SpanFirstQuery`, `SpanNearQuery`,
`SpanNotQuery`, `SpanOrQuery`,
`SpanRegexQuery`

Spans

- The `Spans` object provides document and position info about the current match
- From `SpanQuery`:

- `Spans getSpans(IndexReader reader)`

- **Interface definitions:**

- `boolean next()` //Move to the next match

- `int doc()` //the doc id of the match

- `int start()` //The match start position

- `int end()` //The match end position

- `boolean skipTo(int target)` //skip to a doc

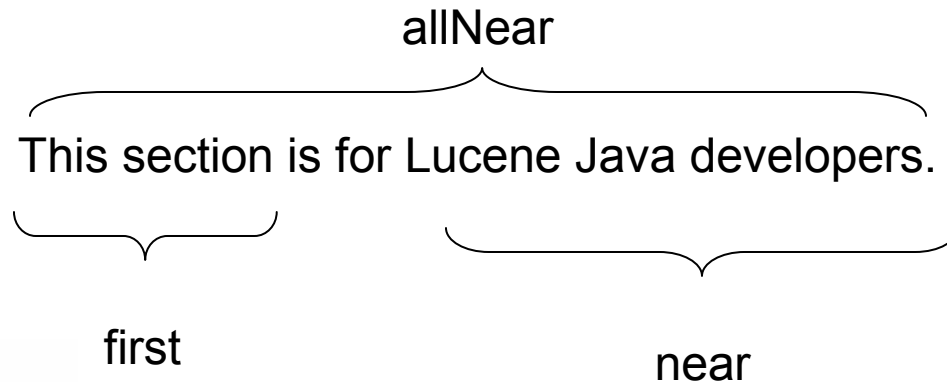
Phrase Matching

- `SpanNearQuery` provides functionality similar to `PhraseQuery`
- Use position distance instead of edit distance
- Advantages:
 - Less slop is required to match terms
 - Can be built up of other `SpanQuery` instances

Example

```
SpanTermQuery section = new SpanTermQuery(new Term("test", "section"));
SpanTermQuery lucene = new SpanTermQuery(new Term("test", "Lucene"));
SpanTermQuery dev = new SpanTermQuery(new Term("test",
                                                "developers"));

SpanFirstQuery first = new SpanFirstQuery(section, 2);
Spans spans = first.getSpans(indexReader); //do something with the spans
SpanQuery [] clauses = {lucene, dev};
SpanNearQuery near = new SpanNearQuery(clauses, 2, true);
spans = first.getSpans(indexReader); //do something with the spans
clauses = new SpanQuery[]{section, near};
SpanNearQuery allNear = new SpanNearQuery(clauses, 3, false);
spans = allNear.getSpans(indexReader); //do something with the spans
```



What is QA?

- Return an answer, not just a list of hits that the user has to sift through
- Often built on top of IR Systems:
 - IR: Query usually has the terms you are interested in
 - QA: Query usually does not have the terms you are interested in
- Question ambiguity:
 - Who is the President?
- Future of search?!?

QA Needs

- Candidate Identification
- Determine Question types and focus
 - Person, place, yes/no, number, etc.
- Document Analysis
 - Determine if candidates align with the question
- Answering non-collection based questions
- Results display

Candidate Identification for QA

- A candidate is a potential answer for a question
- Can be as short as a word or as large as multiple documents, based on system goals
- Example:
 - Q: How many Lucene properties can be tuned?
 - Candidate:
Lucene has a number of properties that can be tuned.

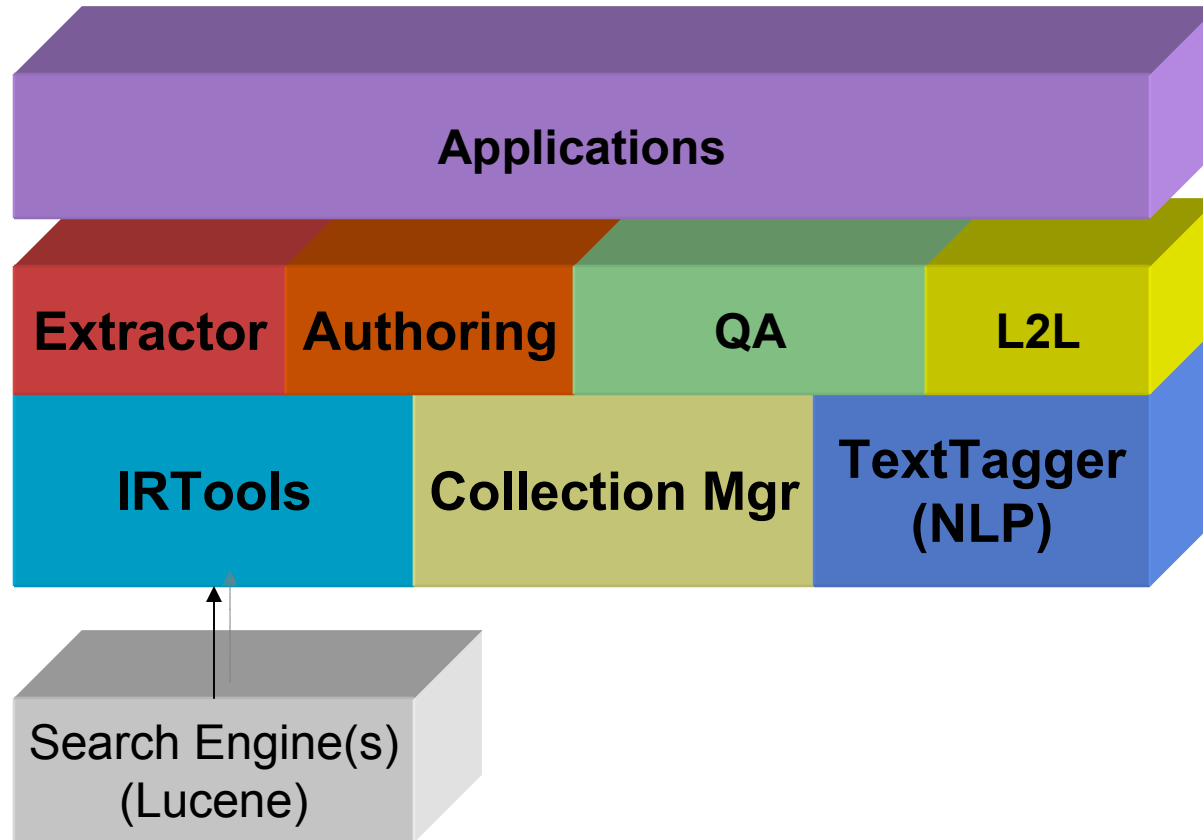
Span Queries and QA

- Retrieve candidates using `SpanNearQuery`, built out of the keywords of the user's question
 - Expand to include window larger than span
- Score the sections based on system criteria
 - Keywords, distances, ordering, others
- Sample Code in `QAService.java` in demo

Example

```
QueryTermVector queryVec = new QueryTermVector(question,
                                                analyzer);
String [] terms = queryVec.getTerms();
int [] freqs = queryVec.getTermFrequencies();
SpanQuery [] clauses = new SpanQuery[terms.length];
for (int i = 0; i < terms.length; i++)
{
    String term = terms[i];
    SpanTermQuery termQuery = new SpanTermQuery(new
        Term("contents", term));
    termQuery.setBoost(freqs[i]);
    clauses[i] = termQuery;
}
SpanQuery query = new SpanNearQuery(closures, 15, false);
Spans spans = query.getSpans(indexReader);
//Now loop through the spans and analyze the documents
```

Tying It All Together



Building an IR Framework

- Goals:
 - Support rapid experimentation and deployment
 - Configurable by non-programmers
 - Pluggable search engines (Lucene, Lucene LM, others)
 - Indexing and searching
 - Relevance Feedback (manual and automatic)
 - Multilingual support
 - Common search API across projects

Lucene IR Tools

- Write implementations for Interfaces
 - Indexing
 - Searching
 - Analysis
 - Wrap Filters and Tokenizers, providing Bean properties for configuration
 - Explanations
- Use Digester to create
 - Uses reflection, but only during startup

Sample Configuration

- **Declare a Tokenizer:**

```
<tokenizer name="standardTokenizer"  
           class="StandardTokenizerWrapper"/>
```

- **Declare a Token Filter:**

```
<filter name="stop" class="StopFilterWrapper"  
        ignoreCase="true" stopFile="stopwords.dat"/>
```

- **Declare an Analyzer:**

```
<analyzer class="ConfigurableAnalyzer">  
  <name>test</name>  
  <tokenizer>standardTokenizer</tokenizer>  
  <filter>stop</filter>  
</analyzer>
```

- **Can also use existing Lucene Analyzers**

Case Studies

- Domain Specialization
- AIR (Arabic Information Retrieval)
- Commercial QA
- Trouble Ticket Analysis

Domain Specialization

- At CNLP, we often work within specific domains
 - Terrorism
 - Insurance
 - CRM
 - Scientific
 - Trouble Tickets
 - Banking
- Users are Subject Matter Experts (SMEs) who require advanced search and discovery capabilities
- We often cannot see the documents when developing solutions for customer

Domain Specialization

- We provide the SME with tools to tailor our out of the box processing for their domain
- Users often want to see NLP features like:
 - Phrase and entity identification
 - Phrases and entities in context of other terms
 - Entity categorizations
 - Occurrences and co-occurrences
- Upon examining, users can choose to alter (or remove) how terms were processed
- Benefits
 - Less ambiguity yields better results for searches and extractions
 - SME has a better understanding of collection

KBB

File Collection Edit Options Help

Collection Manager Phrases Terms Context Document

Phrase Type: NP # NPs: 1044 Average NPs/doc.: 65.2 # Unique NPs: 342 Average Unique NPs/doc.: 21.4

Frequency: # unique occurrences of cat. # occurrences of cat. # docs containing cat. # occurrences of NP # docs containing NP

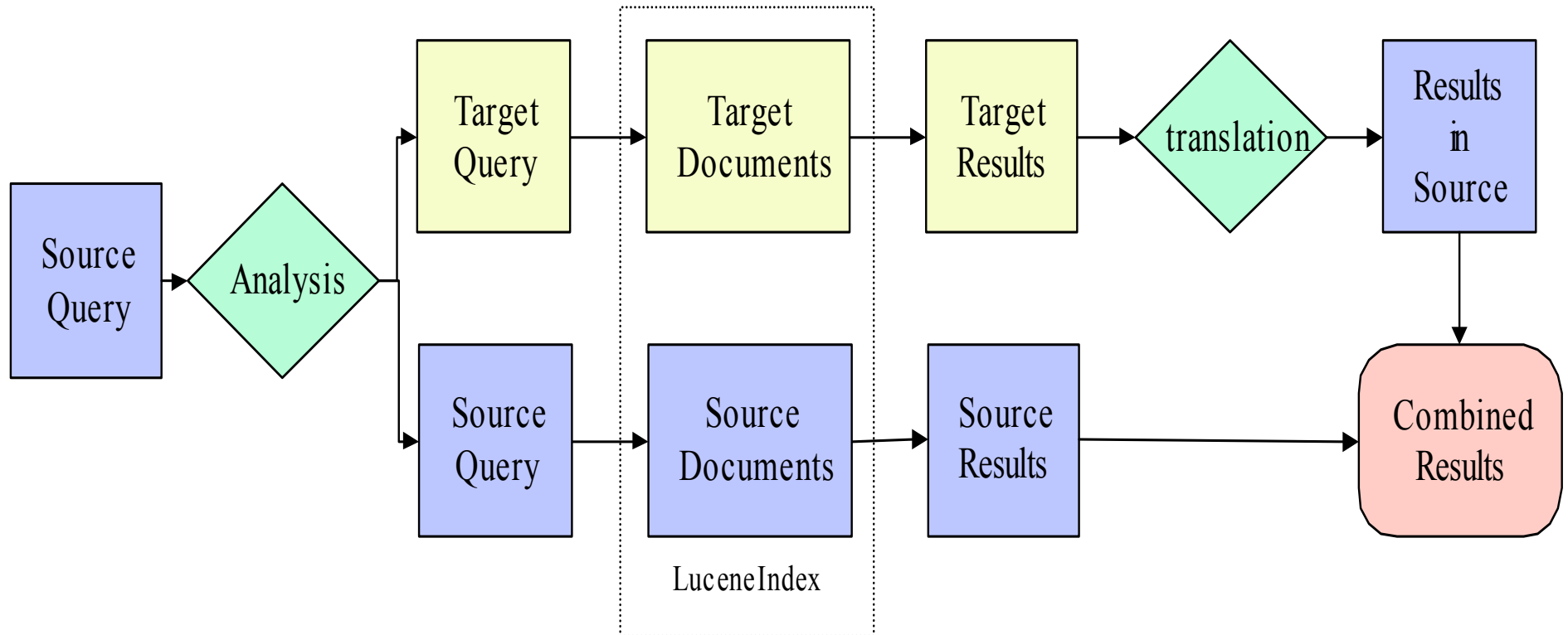
Category	Frequency	NP	Frequency
Person/Individual	36	Carmen	2
Companies	2	Aida	2
Newspapers, Journals and ...	2	Peter Carlson	2
Specialized Facility	2	Doug	2
Tests and Measures	2	Andy Scholz	1
Address	1	Aida TO Carmen	1
City, Town, Village	1	Brian Goetz	1
Documents	1	Andrew C. Oliver	1
Historical period	1	Cory Hubert	1
Internet	1	Ben Litchfield	1
Island	1	Daniel	1
		Daniel Armbrust	1

- Provides several layers of co-occurrence data to the SME for custom processing
 - Taxonomy -> Phrases
 - Terms -> Context

Lucene and Domains

- Build an index of NLP features
- Several Alternatives for domain analysis:
 - Store Term Vectors (including position and offset) and loop through selected documents
 - Custom Analyzer/filter produces Lucene tokens and calculates counts, etc.
 - Span Queries for certain features such as term co-occurrence

Arabic Info. Retrieval (AIR)



AIR and Lucene

- AIR uses IR Tools
- Wrote several different filters for Arabic and English to provide stemming and phrase identification
- One index per language
- Do query analysis on both the Source and Target language
- Easily extendable to other languages

QA for CRM

- Provide NLP based QA for use in CRM applications
- Index knowledge based documents, plus FAQs and PAQs
- Uses Lucene to identify passages which are then processed by QA system to determine answers
- Index contains rich NLP features
- Set boosts on Query according to importance of term in query. Boosts determined by our query analysis system (L2L)
- We use the score from Lucene as a factor in scoring answers

Trouble Ticket Analysis

- Client has Trouble Tickets (TT) which are a combination of:
 - Fixed domain fields such as checkboxes
 - Free text, often written using shorthand
- Objective:
 - Use NLP to discover patterns and trends across TT collection enabling better categorization

Using Lucene for Trouble Tickets

- Write custom Filters and Tokenizers to process shorthand
 - Tokenizer:
 - Certain Symbols and whitespace delineate tokens
 - Filters
 - Standardize shorthand
 - Addresses, dates, equipment and structure tags
 - Regular expression stopword removal
- Use high frequency terms to identify lexical clues for use in trend identification by TextTagger
- Browsing collection to gain an understanding of domain

Resources

- Demos available
- <http://lucene.apache.org>
- Mailing List
 - java-user@lucene.apache.org
- CNLP
 - <http://www.cnlp.org>
 - <http://www.cnlp.org/apachecon2005>
- *Lucene In Action*
 - <http://www.lucenebook.com>