Tratamiento del lenguaje natural

- **Tema 5-III: Tratamiento del lenguaje natural**
  - Comprensión de LN
    - Análisis morfo-léxico
    - Análisis sintáctico
    - Análisis semántico
    - Análisis pragmático
  - Generación de LN
  - **Métodos empíricos**
  - **Recuperación de información**

Tratamiento del lenguaje natural

- El problema de los métodos racionalistas que hemos visto es la dificultad de codificar manualmente el conocimiento lingüístico necesario (diccionarios, gramáticas, etc.)
  - Trabajar con un lenguaje reducido
  - Tenemos que almacenar todas las palabras del lenguaje
  - Los métodos racionalistas no tienen contemplada la capacidad de aprendizaje
- En cambio, los métodos empíricos aprenden a partir de datos prácticos (corpus de documentos)

Tratamiento del LN: métodos empíricos

- Métodos empíricos
  - Inspirados en el éxito de los métodos estadísticos utilizados en el reconocimiento del habla
    - HMMs
      - Hidden Markov models (modelos ocultos de Markov)
    - N-gramas

- La idea básica es utilizar las N-1 palabras anteriores para predecir la siguiente.

- Los sistemas que utilizan métodos empíricos necesitan una fase de entrenamiento en la que se les debe proporcionar un número suficiente de ejemplos
  - Corpus

Teoría de la Probabilidad

- La probabilidad de que una palabra sea (por ejemplo), ‘perro’, sabiendo que la primera palabra es ‘el’, es la fracción de veces que ‘el’ aparece seguido de ‘perro’
  - P(el | perro) = numVeces(el perro) / numVeces(el)
- Un importante teorema que relaciona las probabilidades condicionales es la regla de Bayes:
  - P(A | B) = P(B | A) * P(A) / P(B)
- Ayuda a calcular unas probabilidades a partir de otras.

Teoría de la probabilidad

- El objetivo es asignar una probabilidad a todas las secuencias posibles de palabras.
  - P(w1,n) = P(w1) P(w2 | w1) P(w3 | w1,2) …P(wn | w1,n-1)
    - w1,n es la cadena de n palabras w1, w2, w3, …wn-1, wn
- De esta forma, el problema de asignar probabilidades a las secuencias se reduce a la tarea de asignar probabilidades a la siguiente palabra de un texto.

Modelos de lenguaje

- Modelos Trigram
- El modelo n-gram es uno de los modelos estadísticos del lenguaje más simples pero más útiles.
- Se basa en el supuesto de que sólo las n-1 palabras anteriores tienen efecto sobre las probabilidades de la siguiente palabra.
- Modelos tri-gram (n = 3):
  - P(wn | w1,n-1) = P(wn | wn-2, wn-1)

Ejemplo

- Probabilidades de bigramas obtenidos de un corpus
  - <s>Yo = 0.25, Yo quiero = 0.32, Quiero tomar = 0.05, Comer a = 0.26, Buen comer = 0.06, <s>Opino = 0.06, Yo podria 0.29, …
    - (el símbolo <s> indica “inicio de oración”)
- Dada la oración:
  - a) *Yo quiero tomar alguna bebida Nacional.*
  - P(*Yo quiero tomar alguna bebida Nacional*) = P(Yo | <s>) P(quiero | Yo) P(tomar | quiero) P(alguna | tomar) P(bebida | alguna) P(nacional| bebida)
    - = .25 * .32 * .05 * .14 * .02 *.01 =.112 X 10-6
- Ejemplo
  - Sugerencia de palabras al escribir un mensaje
  - Sugerencias de palabras al escribir una consulta

PCFGs

- Gramáticas de Contexto Libre Probabilísticas (PCFGs)
  - Permiten asignar probabilidades a los árboles de análisis.
  - Cada regla de la gramática tiene una cierta probabilidad
  - Ejemplo
    - O -> S P (0.80), S -> Det N (0.40), S -> Det N Adj (0.25) ….
  - P(frase) = π P(reglas aplicadas en el análisis)
    - P(el perro ladra) =
      - P(O -> S P) P(S -> Det N) P(P->V) P(Det->el) P(N->perro) P(V->ladra)
  - Si hay dos posibles análisis (ambigüedad) => se calcula la probabilidad de cada posible análisis para determinar el más probable

Áreas de uso

- Las áreas de uso más importantes son:
  - Etiquetado sintáctico (POS tagging)
    - asignación de categoría sintáctica (nombre, verbo, …) a cada palabra) > 95%
  - Traducción automática mediante alineamiento de textos (dados los mismos textos en dos idiomas distintos)
    - Ejemplo: Sesiones bilingües del parlamento canadiense
  - Desambiguación léxica
    - Anotamos a mano varios significados
    - El sistema estadísticamente determinará el más probable en la siguiente aparición
  - Análisis sintáctico (bancos de árboles)
    - Ej.: 50.000 frases del *Wall Street Journal*
    - Gramáticas probabilísticas independientes del contexto (PCFGs)

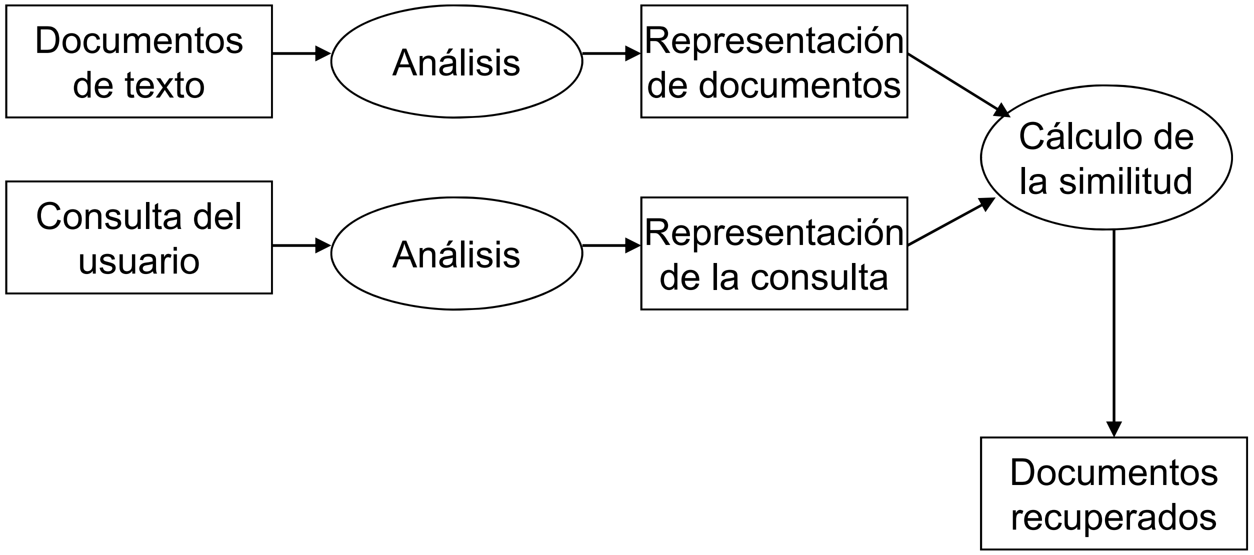
Recuperación de información

- Recuperación de información (IR, information retrieval) trata con la representación, almacenamiento, organización y acceso a elementos de información.
- La representación y la organización de la información deben facilitar al usuario el acceso a la información en la que está interesado
  - Pero, la caracterización de la necesidad de información de un usuario no es un problema sencillo
    - Traducción a una consulta
  - La información relevante para el usuario es aquella que satisface su necesidad de información
    - Juicios de relevancia del usuario sobre necesidad de información, no sobre consulta

Modelo de IR

- Representaciones de documentos
- Representaciones de consultas
- Marco para modelar representaciones de documentos, consultas y sus relaciones
- Función de similitud entre la representación de un documento y una consulta
  - Genera un número real
  - Define un orden de los documentos con respecto a una consulta

Modelo de IR


Documentos de texto
Análisis
Representación de documentos
Consulta del usuario
Análisis
Representación de la consulta
Cálculo de la similitud
Documentos recuperados

Modelo booleano

- Sencillo, basado en teoría de conjuntos y álgebra de Boole.
- Consulta expresada en forma de expresión booleana
  - Semántica precisa
- Desventajas
  - Decisión binaria de relevancia
    - Relevante o no relevante
  - Consulta difícil de generar a partir de necesidad de información

Term-document incidence

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

1 if play contains term, 0 otherwise.
So we have a 0/1 vector for each term.

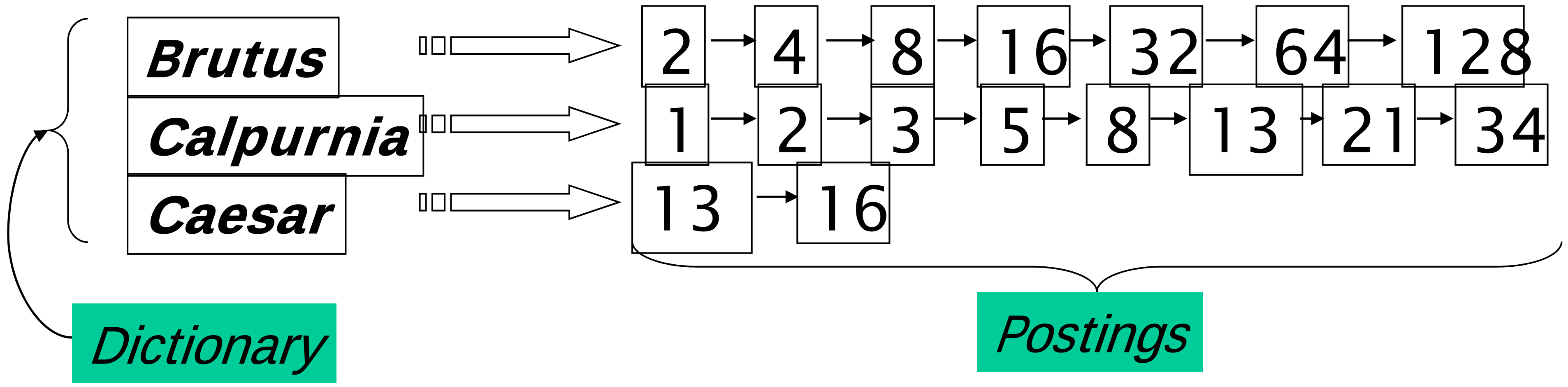
- Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?
  - To answer query: take the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) □ bitwise *AND*.
  - 110100 *AND* 110111 *AND* 101111 = 100100.

Bigger document collections

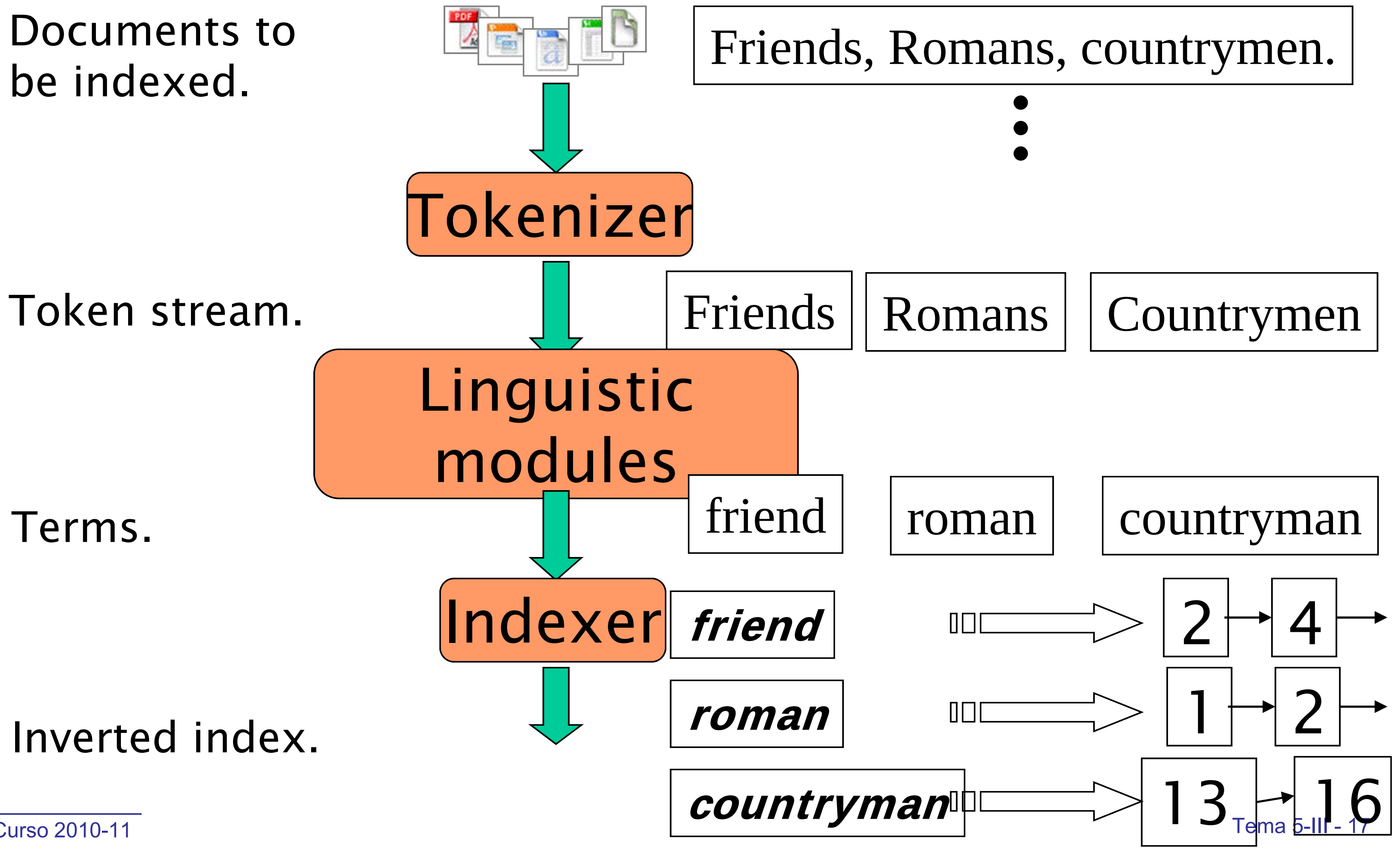
- Consider *N* = 1 million documents, each with about 1000 terms.
  - (document) collection
- Avg 6 bytes/term including spaces/punctuation
  - 6 GB of data in the document collection.
- Say there are *M* = 500000 *distinct* terms among these.
  - 500000 x 1000000 matrix has $5 \cdot 10^{11}$ 0's and 1's.
  - Too many to fit in a computer's memory
- But it has no more than 10^9 1's.
  - matrix is extremely sparse.
  - 99.8% of the cells are zero
- What's a better representation?
  - We only record the 1 positions.

Inverted index

- For each term *T*: store a list of all documents that contain *T*.
  - Linked lists generally preferred to arrays
    - Dynamic space allocation
    - Insertion of terms into documents easy



Inverted index construction



Dropping common terms: stop words

- Stop lists
  - The most frequent words, often hand-filtered for their semantic content
- The members of which are discarded during indexing
  - Significantly reduces the number of postings
  - But can affect to phrase queries
    - To be or not be
- The general trend in IR systems over time has been from standard use of quite large stops lists (200-300 terms) to very small stops lists (7-12 terms) to no stop list
  - Web search engines generally not use stop lists

Lemmatization

- Reduce inflectional/variant forms to base form
  - Use of a vocabulary and morphological analysis of words
- E.g.,
  - *am, are, is* $\rightarrow$ *be*
  - *car, cars, car's, cars'* $\rightarrow$ *car*
- *the boy's cars are different colors* $\rightarrow$ *the boy car be different color*

Stemming

- Reduce terms to their "roots"
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

$\Rightarrow$

for exampl compres and compres are both accept as equival to compres.

Evidence accumulation

- 1 vs. 0 occurrence of a search term
  - 2 vs. 1 occurrence
  - 3 vs. 2 occurrences, etc.
- Need term frequency information in docs.
- Used to compute a score for each document
- Matching documents *rank-ordered* by this score.

Term frequency vectors

- Consider the number of occurrences of a term *t* in a document *d*, denoted $\mathrm{tf}_{t,d}$
  - Document is a vector: a column below
  - *Bag of words* model
    - The exact ordering of the terms in a document is ignored

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

tf x idf term weights

- tf x idf measure combines:
  - term frequency (*tf*)
    - or *wf*, some measure of term density in a doc
  - inverse document frequency (*idf*)
    - measure of informativeness of a term: its rarity across the whole corpus

Weighting term frequency: *tf*

- What is the relative importance of
  - 0 vs. 1 occurrence of a term in a doc
  - 1 vs. 2 occurrences
  - 2 vs. 3 occurrences …
- Unclear: while it seems that more is better, a lot isn't proportionally better than a few
  - Can just use raw *tf*
  - Another option commonly used in practice:

$$wf_{t,d} = 0 \text{ if } tf_{t,d} = 0,\ 1 + \log tf_{t,d} \text{ otherwise}$$

Summary: tf x idf (or tf.idf)

- Assign a tf.idf weight to each term t in each document *d*

$$w_{t,d} = tf_{t,d} \times \log(N / df_t)$$

$$tf_{t,d} = \text{frequency of term } t \text{ in document } d$$

$$N = \text{total number of documents}$$

$$df_t = \text{the number of documents that contain term } t$$

- Increases with the number of occurrences *within* a doc
- Increases with the rarity of the term *across* the whole corpus

Real-valued term vectors

- Still Bag of words model
- Each is a vector in $\mathbb{R}^M$
  - Here log-scaled *tf.idf*

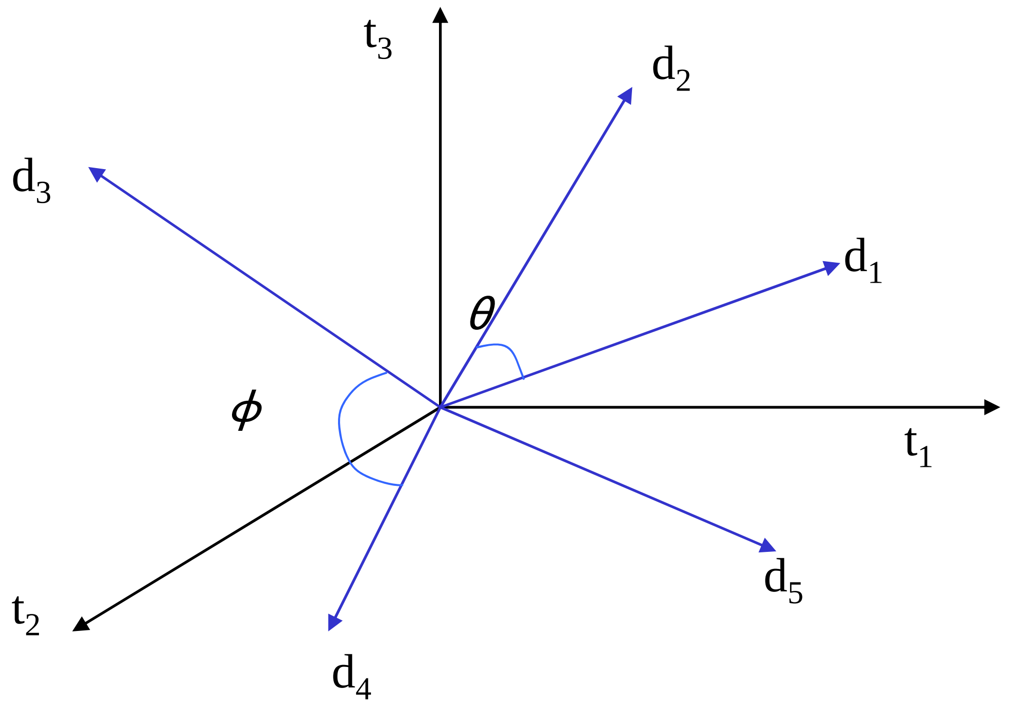
Note can be >1

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

Documents as vectors

- Each doc *j* can now be viewed as a vector of *tf*×*idf* values, one component for each term
- So we have a vector space (VSM => Vector Space Model)
  - terms are axes (dimensions)
  - docs live in this space
  - even with stemming, may have 20,000+ dimensions
- A collection can be view as a term-document matrix
  - M terms as rows
  - N documents as columns

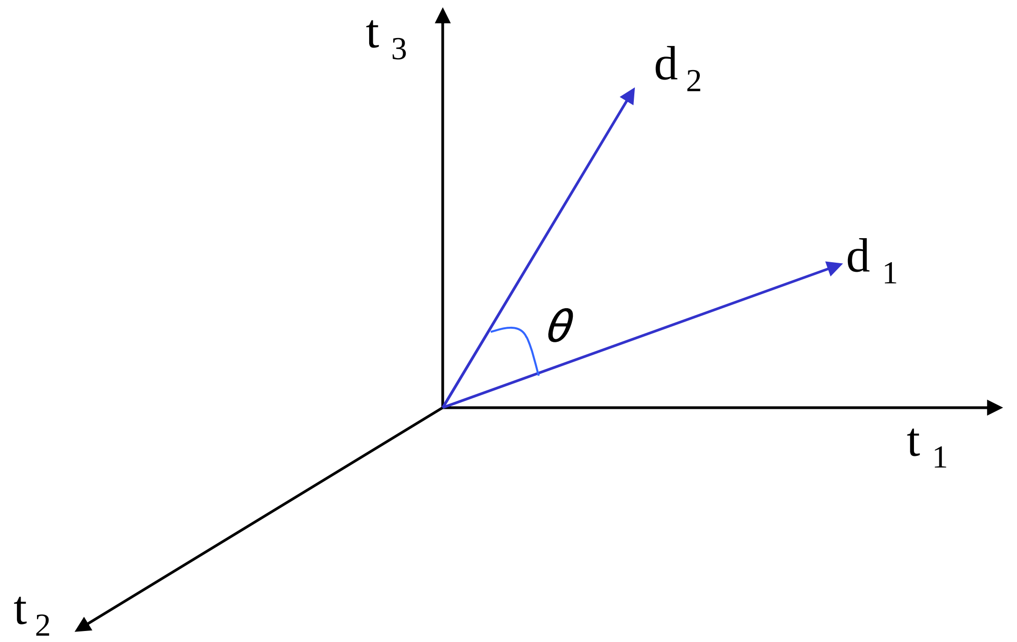
Intuition



Postulate: Documents that are “close together” in the vector space talk about the same things.

Cosine similarity

- Distance between vectors d_1 and d_2 *captured* by the cosine of the angle *x* between them.
- Note – this is *similarity*, not distance



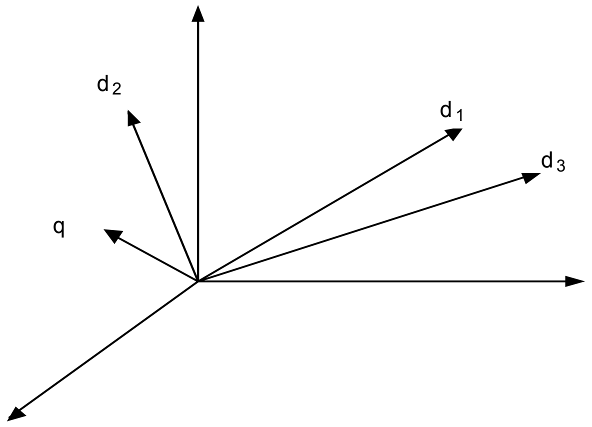
Cosine similarity

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right| \left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{M} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{M} w_{i,j}^2} \sqrt{\sum_{i=1}^{M} w_{i,k}^2}}$$

- Cosine of angle between two vectors
- The denominator involves the lengths of the vectors.

- A document may have a high cosine score for a query even if does not contain all query terms

Ejemplo



	algorithm	architecture	computer	logic	program
Documento1	2.23	5.34	2.45	0.00	0.00
Documento2	3.50	0.00	0.00	3.20	1.51
Documento3	0.00	4.76	3.23	0.00	2.31
Consulta	0.00	0.00	0.00	1.06	0.74

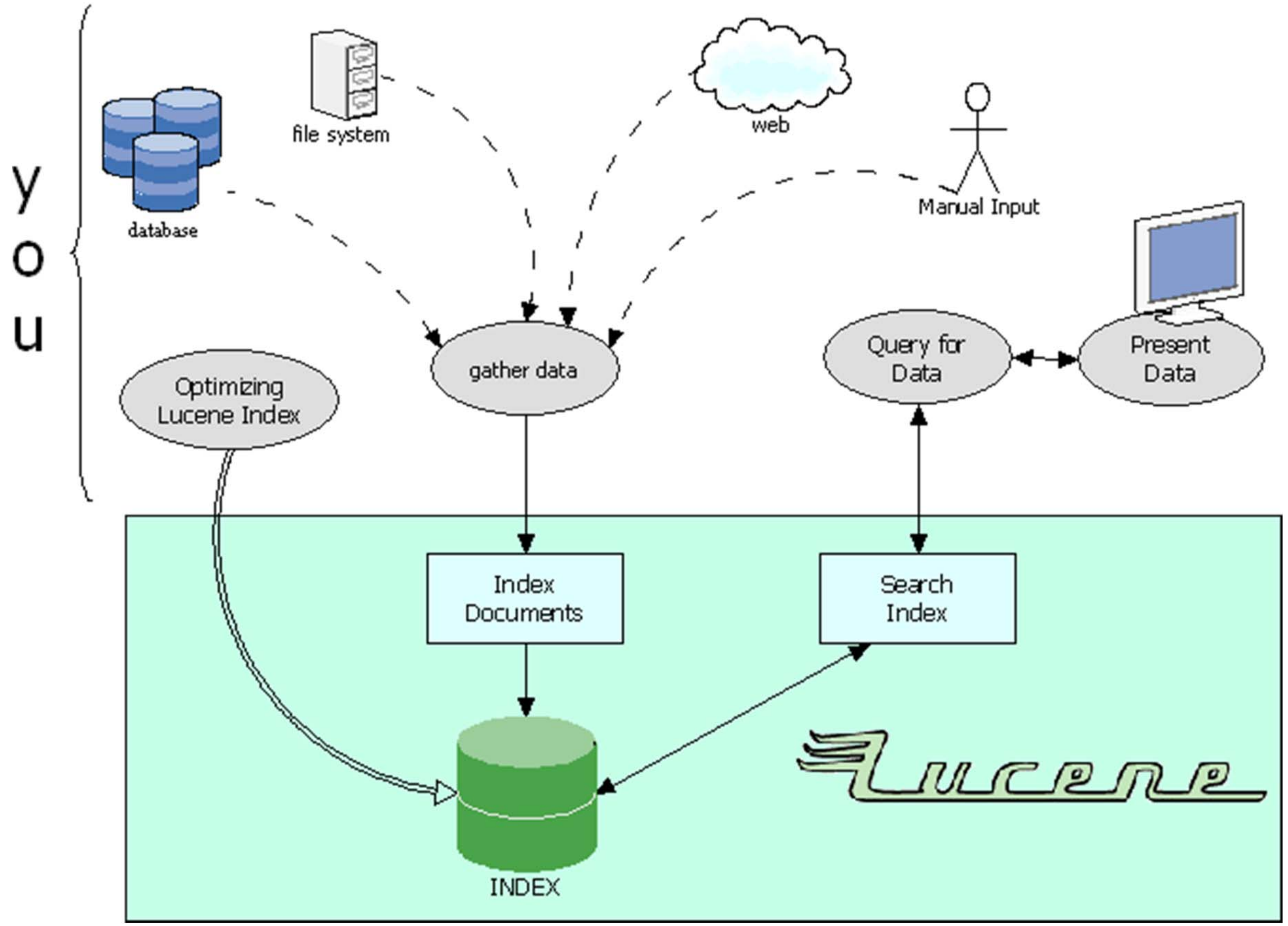
similitudes entre documentos d_i y consulta q:

$sim(d_1,q) = 0.0000$
$sim(d_2,q) = 0.7009$
$sim(d_3,q) = 0.2133$

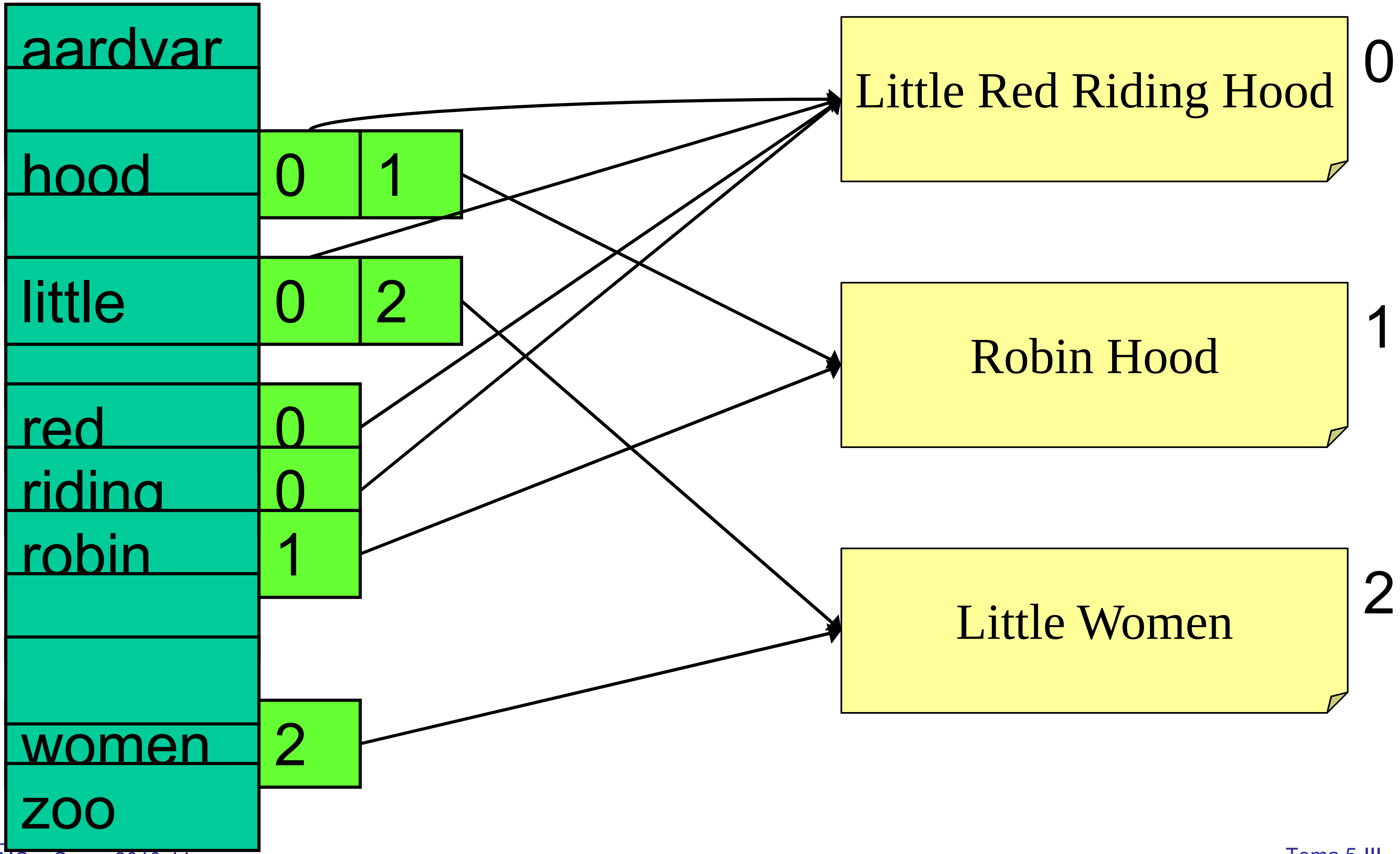
Vector space scoring and query operator interaction

- Vector space scoring supports so-called free text retrieval
  - Without any query operators connecting them
- The classical interpretation was that at least one of the query terms be present in any retrieved document
- More recently, web search engines as Google have popularized the semantics of conjunctive query
  - Only retrieve documents containing all or most query terms

Lucene Overview



Inverted Index

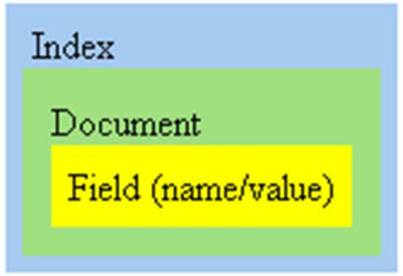


Lucene's main Goal: Indexing and Searching

- What is Indexing – the creation of a data structure that facilitates fast, random access to information stored in it.
- What is Searching – the process of looking up and retrieving information from the index for the purpose of accessing the underlying documents.

High Level

- Indexing
  - A Document is a collection of Fields
  - A Field is free text, keywords, dates, etc.
  - A Field can have several characteristics
    - Apply Analyzer to alter Tokens during indexing
      - Stemming (reducing a word to its root form)
      - Stopword removal
      - Phrase identification



Simplicity

- IndexWriter
  - Creates a new index and adds documents to an existing index
  - writer.addDocument(document)
- Analyzer
  - It is in charged of extracting tokens out of text to be indexed and eliminating the rest

Simplicity

- IndexSearcher
  - Implements search over a single IndexReader.
  - searcher.search(query)
- Query
  - The abstract base class for different types of queries.
    - TermQuery: A query with one Term
    - Term: a pair of strings: name of the field and value of that field.
- Hits
  - It is a simple container of pointers to ranked search results.

Lucene Analyzers

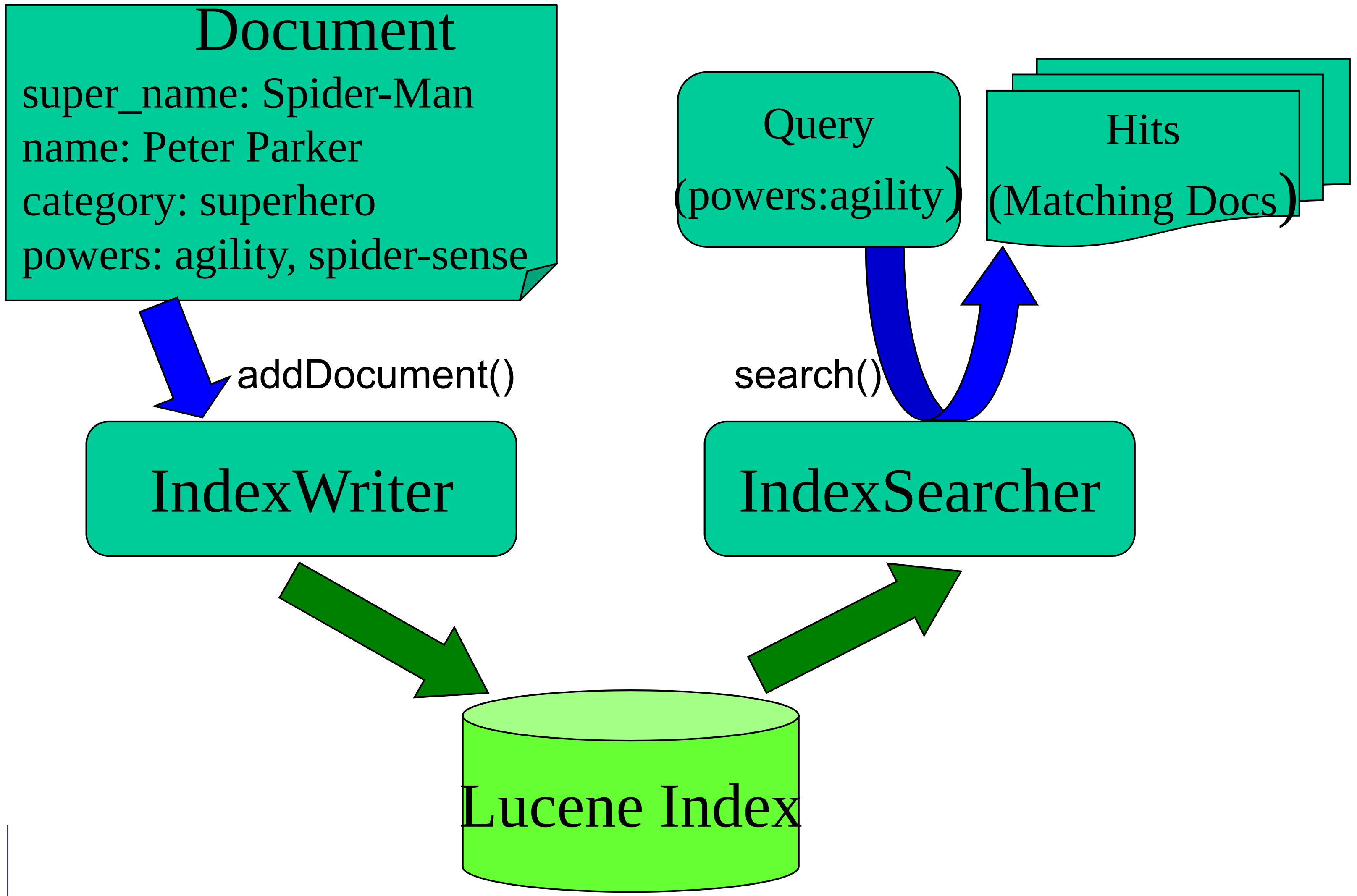
- Lucene indexes text, and part of the first step is cleaning up the text.
- You use an Analyzer to do this - it drops out punctuation and commonly occurring but meaningless words (the, a, an, etc).
- Lucene provides a couple different Analyzers, and you can make but your own, but **you must make sure you use the same sort of analyzer for indexing and for searching**.

- **SimpleAnalyzer**: seems to just use a Tokenizer that converts all of the input to lower case.
- **StopAnalyzer** includes the **lower-case filter, and** also has a filter that **drops out any "stop words",** words like articles (a, an, the, etc) that occur so commonly in English..
- **StandardAnalyzer** does both **lower-case and stop-word filtering**, **and** in addition tries to do some **basic clean-up of words,** for example taking out apostrophes (') and removing periods from acronyms (i.e. "T.L.A." becomes "TLA").

Query Terms

- A query is broken up into terms and operators. There are two types of terms: Single Terms and Phrases.
- A Single Term is a single word such as "test" or "hello".
- A Phrase is a group of words surrounded by double quotes such as "hello dolly".
- Note: The analyzer used to create the index will be used on the terms and phrases in the query string. So it is important to choose an analyzer that will not interfere with the terms used in the query string.

Basic Application



Basic Application

- http://apache.rediris.es/lucene/java/archive/lucene-2.3.2.zip
  - Descomprimir
- Crear proyecto de Eclipse
  - añadir lucene-core-2.3.2.jar
- http://lucene.apache.org/java/docs/index.html

Indexing Documents

Document persona1 = *crearDocumento*(“Pepe Pérez", "UCM", "www.ucm.es/pepe","111111111");

Document persona2 = *crearDocumento*(“Pepe Gómez", "UAM", "www.uam.es/pepe","222222222");

- **public static** Document crearDocumento(String nombre, String empresa, String web, String telefono)

  Document document = **new** Document();

  document.add(**new** Field("nombre", nombre, Store.*YES*, Index.*TOKENIZED*));

  document.add(**new** Field("empresa", empresa, Store.*YES*, Index.*TOKENIZED*));

  document.add(**new** Field("web", web, Store.*YES*, Index.*UN_TOKENIZED*));

  document.add(**new** Field("telefono", telefono, Store.*YES*, Index.*UN_TOKENIZED*));

  return document;

Indexing Documents

```
Analyzer analizador = new StandardAnalyzer();
IndexWriter writer = new IndexWriter(directorioIndexacion,
    analizador, true);
writer.setUseCompoundFile(false);
writer.addDocument(persona1);
writer.addDocument(persona2);
writer.optimize();
writer.close();
```

Tokenizers

Tokenizers break field text into tokens

- StandardTokenizer
  - source string: “full-text lucene.apache.org”
  - “full”   “text”   “lucene.apache.org”
- WhitespaceTokenizer
  - “full-text”   “lucene.apache.org”
- LetterTokenizer
  - “full”   “text”   “lucene”   “apache”   “org”

Analyzers

```
class MyAnalyzer extends Analyzer {
 private Set myStopSet =
    StopFilter.makeStopSet(StopAnalyzer.ENGLISH_STOP_WORDS);

 public TokenStream tokenStream(String fieldname, Reader reader) {
   TokenStream ts = new StandardTokenizer(reader);
   ts = new StandardFilter(ts);
   ts = new LowerCaseFilter(ts);
   ts = new StopFilter(ts, myStopSet);
   return ts;
 }
}
```

Searching an Index

```
IndexSearcher searcher = new IndexSearcher(directorioIndexacion);
Analyzer analyzer = new StandardAnalyzer();

QueryParser parser = new QueryParser("nombre", analyzer);
Query query = parser.parse("Pepe");
Hits hits = searcher.search(query);
System.out.println("matches:" + hits.length());

HitIterator iterador = (HitIterator) hits.iterator();
while (iterador.hasNext())
        Hit hit = (Hit) iterador.next();
        Document doc = hit.getDocument();
        System.out.println("nombre=" + doc.get("nombre"));

searcher.close();
```

Enlaces

- Lucene
  - http://lucene.apache.org/
- Tutorial de Lucene 2.9.1
  - http://www.lucenetutorial.com/lucene-in-5-minutes.html
- http://www.slideshare.net/otisg/lucene-introduction
- LingPipe
  - http://alias-i.com/lingpipe/