University of Applied Sciences and Arts Northwestern Switzerland
School of Business

Topics in Business Information Technology

Methods for Determining the Similarity of Documents

Christian Häusler

Autumn term 2012/13

Instructor: Prof. Dr. Anke Dreiling

Supervisor: Prof. Dr. Thomas Hanne

Contents

Abstract

Nowadays we are confronted with rapidly increasing number of documents. Maintaining an overview seems to be impossible. Indexing this documents with search engines allows us to group similar documents (clustering) or find the documents we are interested in with a search query. This paper provides an overview on what is involved when one wants to index digital text documents using the open source search engine library *Apache Lucene*.

In the first part this paper will discuss the terms document and similarity in the given context. The second part is about Apache Lucene and shows what tools Lucene provides in a more general perspective. While as the third part have a more detailed look at the moving part involved to build a search index with Lucene.

1 Introduction

The title of this writing contains two general words which I will clarify before we go into the details. In this first chapter the terms *document* and *similarity* will be analyzed.

1.1 Documents

A *document* is as a piece of written, printed, or electronic matter that provides information or evidence or that serves as an official record (*"document". Oxford Dictionaries* 2013).

This definition gives us at least two clues. First, documents can exist in many different forms both analog and digital and they contain information. The following list are a few examples of different document kinds.

- Book
- Online article
- Newspaper article
- Photography
- Letter
- Movie

Going further in describing documents we can find a whole bunch of attributes like title, author, type, creation date or date of last change just to name a few. But when we need to decide if we want to read a document, mostly we want to know what the content of a document is all about.

1.2 Similarity

Similarity is the state or fact of being similar while similar is referring to a resemblance in appearance, character, or quantity, without being identical (*"similar". Oxford Dictionaries* 2013; *"similarity". Oxford Dictionaries* 2013).

So in order to describe the similarity of two objects, we need to agree on the aspects we want to compare. The classic example in this context are geometric shapes. While all rectangles are similar they do not necessarily have the same size.

1.3 Document similarity

In order to bring the term *document* and *similarity* together we need to agree on a set of attributes which then can be used to compare documents.

But how can we compare those attributes? Having a closer look on those attributes we can see that they have a certain set of characteristics and we can use this characteristics to group them together and at the end we have a few types that we can compare in their own way.

Probably the easiest ones are the *numerical values* and as special case, *date values*. One of those values can be greater than, less than or equal than another or they can be within a certain range.

Another type is what we can refer to as an *entity*. Such an entity could be an Author or a category. So we are dealing with a set of values that can contain a predefined list or a list that differs depending on how many documents we compare.

This leaves us with the last type that is the content itself. Assuming we are dealing with textual documents, comparing the length of the content does not help us any further. We need to find a way how to describe this content in a way that will make it comparable.

For that librarians uses fixed dictionaries of terms and assign one or several of those keywords to a document depending on its content. Those keywords are maintained in a index for easy look up. This once again have the characteristics of list of entities which we are able to compare.

2 Apache Lucene

In order to compare a huge amount of documents we need an index where we can look up documents by certain keywords. In the past it was the job of librarians to maintain such an index by hand. Doing this by hand is a tedious and error prone process and when dealing with several hundred thousands of documents one might want to have this automated. This is where Apache Lucene comes in.

But what is Apache Lucene? Lucene is a library written in Java that allows the programmer to build and maintain an index of documents and allows to query this index to find those documents one is interested in. It also allows to build clusters, which is the process of grouping similar documents.

In the following I will look at how Lucene handles index building and searching.

Apache Lucene is only capable of dealing with plain text and thus does not support parsing of documents. Depending on the format of the documents that should be indexed some preparation work needs to be done before Lucene can do its magic. For more details see Section 3.1.

2.1 Building an index

Like librarians, Lucene stores a list of keywords for each indexed documents. But instead of using a fixed vocabulary, Lucene extracts this keywords out of the documents text. This process is called tokenization. The challenge here is to filter out rubbish and only keep what really matters for the given document. In order to do so Lucene allow to apply filters both before and after the tokenization.

Lucene brings along a whole bunch of analyzers both optimized for a specific language or make use of a specific tokenization algorithm. Those analyzers are build with general purpose in mind, so one may want to bring in optimizations specific to the given documents. Section 3.2 will have a closer look at the tokenization techniques.

2.2 Index persistence

Once we know the keywords of a document we need to store this information for later usage. In Lucene terminology this is called a *Directory*. This is an abstract concept allowing Lucene to make no assumption on how the index actually is stored. Lucene brings along an implementation for storing the index in the memory (*RAMDirectory*) which is great for speed but only on a small amount of data or at the file system

(*FSDirectory*) which is not that fast but can handle more date than the other. If neither of those does fit ones need, a custom Directory implementation can be written.

2.3 Searching in index

In order to find documents that match our informational need, Lucene uses a *Query* object. This query object can be constructed by a programmer in code or a *query builder* can be used to parse search phrase like one would enter in a search field.

The terms inside of a search query are applied with the same analyzing concepts as described in 2.1. The documentation of Lucene states that it is a good start to use the same configuration yet this might not apply in all cases and different analyzers may be used.

Just a short example: Given we had indexing documents that uses the Mediawiki syntax for formating. The analyzer for indexing probably contains a filter to deal with this noise that is introduced by this formatting syntax. On the other side, queries are build by using a query builder and we can expect that the user does not enter any formatting characters. So the analyzer for the search query does not need to deal with this formatting and therefore does not contain this filter.

2.4 Lucenes understanding of a Document

For Lucene a document is a simple object that have an undefined number of fields. Each field can have a specific data type that controls the behaviour of the analyzer and allows for different search approaches as we have seen in section 1.3. It also can be marked as storable or not. A storable field will be persisted in the index and therefore be available on retrieval otherwise it will only be part of the keywords.

Determining which part of the document will end up in which field is not covered by Lucene. This is part of the document parsing process and is explained further in section 3.1.

3 In detail

3.1 Parsing: Extracting content from documents

For Apache Lucene can only deal with plain text, we need to somehow extract it content from the document which might contain also formating and oder non textual elements. We also might want to have document attributes indexed in their own fields so we need to extract them too allowing us to search by them separately. This process is called information extraction which is the process of filling the fields and records of a database from unstructured or loosely formatted text (McCallum, 2005).

There are numerous of different document formats out there. An implementation of the techniques described in the following will look different depending on the formats the documents have we want to extract information form.

Depending on the diversity of the document collection and the degree of structuring we need to take more or less effort in order to extract what ever information we are interested in.

Given a online article that makes use of Hyper Text Markup Language 5 (HTML5) in combination with Resource Description Framework (RDF) we can consider ourself lucky. The same holds true for offline documents like Portable Document Format (PDF) or the file formats from word processors like Word or OpenOffice if and only if the meta data fields are set. But in most of the cases we are confronted with documents that does not provide any structured information and we are challenged with the need of parsing native language.

If our document collection does have a homogeneous structure we will get pretty good results using Regular Expression (RegEx). The more diversity and subtlety in language we are facing, the more sophisticated methods we will need.

The simple approach with RegEx and be improved by using rules which itself can make use of state machines.

One can imaging that this approach can easily get out of hand. This is because the structure of our document collection tends to change often or we do have so many rules which are interlinked with each other that this collection of rules will not be maintainable any more.

This leads us to a techniques called Native language Processing (NLP) as they are described in Merlo et al. (2006).

3.2 Analyzing: Tokenization and Filtering

Tokenization refer to the process of splitting up a text in to small pieces (terms) which then will be used to match search query and documents against each other. The most common techniques in this area are explained in the following.

3.2.1 Stemming

In native language one term can appear in different form e.g. singular and plural. Stemming uses an algorithmic approach to reduce such words to their stem. Stemming is highly language specific and needs to be fine tuned. Lucene has implementations of the Porter, Hunspell and Snowflake stemmer. Each of the language specific analyzers provided by Lucene make use of language specific stemming. An overview on stemming and stemming algorithms is given in Smirnov (2008).

3.2.2 Stop words filtering

Stop words are small words that appears often inside a text and therefore does not have any document specific relevance. Filtering out those words will lead to a term list that contains proportionally more relevant terms. Depending on the language other words should be removed got achieve good results. If a document collection belongs to a common domain improvements can be made by using a domain specific list of stop words.

Apache Lucene contains a list of stop words withing each of its language specific analyzer. While this lists are relatively short (below hundred), it is highly recommended to apply custom stop word lists.

Basics on stop words can be found in Manning, Raghaban, and Schütze (2009, p. 27) while Dolamic and Savoy (2009) shows in which extend stop words can improve document retrieval.

3.2.3 Synonym injection

In some occasions a subject does not have any clear naming. So it may whelp to add synonyms of a term so we will get a match for every possible naming. Lucene provides a filter allowing to do exactly this but there is no default implementation of it. This technique is mentioned in Cutting, Pedersen, and Halvorsen (1991, p. 5).

3.2.4 Normalization

Humans are an individualistic species and tend to do the same thing differently. This provides us with a divers list of how times and dates are written as well as differences in word casings and orders of names and first names. The list is not final. Normalization has the aim to bring this diversity to a unified form (McCallum, 2005, p. 50).

The default implementation of Apache Lucenes analyzer applies normalization with a lower cases filter that converts everything to lower case.

3.3 Similarity Measures

In order to compute the similarity of documents we need some mathematical expression or an algorithm the computer can work with. This is called a similarity or distance measure witch maps down the similarity or difference to one single numeric value.

Huang (2008) defines four criteria such a measure must comply with.

Let x and y be any two objects in a set and $d(x, y)$ be the distance between x and y.

1. The distance between any two points must be non- negative, that is, $d(x, y) \geq 0$.
2. The distance between two objects must be zero if and only if the two objects are identical, that is, $d(x, y) = 0$ if and only if $x = y$.
3. Distance must be symmetric, that is, distance from x to y is the same as the distance from y to x, ie. $d(x, y) = d(y, x)$.
4. The measure must satisfy the triangle inequality, which is $d(x, z) \leq d(x, y) + d(y, z)$.

In the following I have a look at the measures that are part of the Lucene library.

3.3.1 Cosine Similarity

By default Lucene uses a similarity measure based on the so called vector space model as documented in *Class TFIDFSimilarity* (2012). This model describes a document as a multi dimensional vector where each occurring therm in the whole document collection represents one dimension. The cosine similarity is then simply the angle between two of this document vectors (Huang, 2008).

The cosine similarity is combined with the term frequency–inverse document frequency ($tfidf$) weighting factor. This factor reduces the relevance of common words so they do get dominant. Lucene uses the following formula to calculate the $tfidf$ factor.

$$tf(t\ in\ d) \cdot idf(t) = f^{1/2} \cdot \left(1 + log(\frac{D}{df + 1})\right) \tag{1}$$

Where f is the frequency the term appears in the given document, D is the number of documents and df the frequency the term appears in all documents.

3.3.2 BM25

BM25 as described in Robertson (2010), is an implementation of the Probability Ranking Principle (PRP). This principle assumes that each document has a binary relevance attribute stating if this document is relevant or not. The measure itself is a probabilistic value of this binary relevance. Ordering the document by their decreasing probability is assumed to give the best result the given document collection can provide.

An explanation on how the BM25 measure was implemented in Lucene is given in Pérez-Iglesias et al. (2009).

3.3.3 Divergence from randomness

Divergence from randomness (DFR) model can be considered a extension of the probabilistic approach. DFR compares the probability of a term inside a document to the probability of this term inside a randomly generated document. The relevance of the term for a document is then derived from the divergence of the probabilities of the actual and the random document (Amati and Van Rijsbergen, 2002).

3.3.4 Language models

Language modeling takes advantage of recent research in speech and language processing. As of Manning, Raghaban, and Schütze (2009, p. 248), the resulting model is mathematically precise, conceptually simple, computationally tractable, and intuitively appealing. The language model approach has a similar characteristics as the $tfidf$ but as Ponte and Croft (1998) shows, it outperforms vector space based models.

Lucene has two implementation of two language based retrieval models as they are described in Zhai and Lafferty (2004).

3.3.5 Comparison

Similarity measures are subject to often comparison against each other (Clinchant and Gaussier, 2010; Huang, 2008; Robertson, 2010). Those comparisons show a clear tendency that the newer probability and language based models performs better than the widely used vector space models. Yet the better performance does not justify to switch over a well tuned and proven vector space based system (Manning, Raghaban, and Schütze, 2009, p. 249f). But when implementing a new system they are clearly worth a look.

4 Conclusions

While confronted with more and more documents we need tools to efficiently find information we are interested in. Apache Lucene is such a tool. It provides us with a foundation to build a information retrieval system for our documents.

While Lucene is capable of creating indexes our document collections and allow us to search this index it does not help us with the task of information extraction and therefore can only be one building block of an retrieval system.

In the process of building a search index, I consider the task of information extraction the most complex, divers and important. In audio techniques we say crap in crap out. Keeping this in mind, investing a great deal pf a projects resources in analyzing the document collection, fine tuning extraction rules, defining stop word lists and so on will probably lead to more satisfying search results.

A List of Abbreviations

HTML5	Hyper Text Markup Language 5
RDF	Resource Description Framework
PDF	Portable Document Format
RegEx	Regular Expression
NLP	Native language Processing
PRP	Probability Ranking Principle
$tfidf$	term frequency–inverse document frequency
DFR	Divergence from randomness

B References

Amati, G and C J Van Rijsbergen (2002). "Probabilistic models of information retrieval based on measuring the divergence from randomness". In: *ACM Transactions on Information Systems* 20.4, pp. 357–389. URL: http://doi.acm.org/10.1145/582415.582416.

Class TFIDFSimilarity. URL: http://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html (visited on 12/28/2012).

Clinchant, Stephane and Eric Gaussier (2010). "Information-Based Models for Ad Hoc IR". In: *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. SIGIR '10, pp. 234–241. URL: http://eprints.pascal-network.org/archive/00007028/.

Cutting, Doug, Jan Pedersen, and Per-kristian Halvorsen (1991). "An Object-Oriented Architecture for Text Retrieval". In: *In Conference Proceedings of RIAO'91, Intelligent Text and Image Handling*, pp. 285–298.

"document". Oxford Dictionaries. URL: http://oxforddictionaries.com/definition/english/document (visited on 01/04/2013).

Dolamic, Ljiljana and Jacques Savoy (2009). “When Stopword Lists Make the Difference”. In: *Journal of the American Society for Information Science and Technology* 61.1, pp. 200–203. DOI: `10.1002/asi.v61:1`. URL: `http://dx.doi.org/10.1002/asi.v61:1`.

Huang, Anna (2008). “Similarity measures for text document clustering”. In: *Proceedings of the Sixth New Zealand Computer Science Research Student Conference (NZCSRSC2008), Christchurch, New Zealand* April, pp. 49–56. URL: `http://nzcsrsc08.canterbury.ac.nz/site/proceedings/Individual_Papers/pg049_Similarity_Measures_for_Text_Document_Clustering.pdf`.

Manning, Christopher D, Prabhakar Raghaban, and Hinrich Schütze (2009). *An Introduction to Information Retrieval*. c. Cambridge University Press New York, NY, USA, p. 544. ISBN: 0521865719 9780521865715. URL: `http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf`.

McCallum, Andrew (Nov. 2005). “Information extraction: Distilling structured data from unstructured text”. In: *Queue* 3.9, pp. 48–57. ISSN: 1542-7730. DOI: `10.1145/1105664.1105679`. URL: `http://dl.acm.org/citation.cfm?id=1105679`.

Merlo, Paola et al. (2006). *Learning Document Similarity Using Natural Language Processing*.

Pérez-Iglesias, Joaquín et al. (2009). “Integrating the Probabilistic Models BM25/BM25F into Lucene”. In: *Language* abs/0911.5.Gile 1995, pp. 173–190. URL: `http://arxiv.org/abs/0911.5046`.

Ponte, Jay M and W Bruce Croft (1998). “A language modeling approach to information retrieval”. In: *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. Ed. by W Bruce Croft et al. Vol. 98. SIGIR ’98 3. ACM. ACM Press, pp. 275–281. ISBN: 1581130155. DOI: `10.1145/290941.291008`. URL: `http://portal.acm.org/citation.cfm?doid=290941.291008`.

Robertson, Stephen (2010). “The Probabilistic Relevance Framework: BM25 and Beyond”. In: *Foundations and Trends® in Information Retrieval* 3.4, pp. 333–389. ISSN: 15540669. DOI: `10.1561/1500000019`. URL: `http://www.nowpublishers.com/product.aspx?product=INR&doi=1500000019`.

”similar”. Oxford Dictionaries. URL: `http://oxforddictionaries.com/definition/english/similar` (visited on 01/04/2013).

”similarity”. Oxford Dictionaries. URL: `http://oxforddictionaries.com/definition/english/similarity` (visited on 01/04/2013).

Smirnov, Ilia (2008). “Overview of Stemming Algorithms”. In: *Mechanical Translation*. URL: `http://the-smirnovs.org/info/stemming.pdf`.

Zhai, Chengxiang and John Lafferty (2004). “A study of smoothing methods for language models applied to information retrieval”. In: *ACM Transactions on Information Systems*. SIGIR ’01 22.2. Ed. by W Bruce Croft et al., pp. 179–214. ISSN: 10468188. DOI: `10.1145/984321.984322`. URL: `http://portal.acm.org/citation.cfm?doid=984321.984322`.

C Declaration of Authenticity

I the undersigned declare that all material presented in this paper is my own work or fully and specifically acknowledged wherever adapted from other sources.

I understand that if at any time it is shown that I have significantly misrepresented material presented here, any degree or credits awarded to me on the basis of that material may be revoked.

I declare that all statements and information contained herein are true, correct and accurate to the best of my knowledge and belief.

Name	Christian Häusler
Date	2013-01-04
Signature	[illegible]