ZHAW - Zürcher Hochschule für angewandte Wissenschaften

Seminararbeit

Information Retrieval

Evaluierung der Retrieval-Leistung einer Search Engine am Beispiel einer Media Library

Michael Kressibucher
kressmic@students.zhaw.ch

Betreuerin: Ruxandra Domenig
Abgabe: 15. Juni 2016

Zusammenfassung

In dieser Arbeit wird mit Apache Lucene ein Index von Mediadateien im MP3 und FLAC Format erstellt. Die notwendigen Metadaten werden mit der Apache Tika Library von den Dateien extrahiert. Vom erstellten Index wird appliaktorisch eine Abfrage erstellt. Für die anschliessende Analyse des Index werden die Abfragen mit Luke, einem Index Analyse Tool, ausgeführt. Da die Konfiguration und Abfragen an das Information Retrieval System nicht komplex sind, sind die Abfragen von sehr hoher Genauigkeit mit einer guten Trefferquote.

Inhaltsverzeichnis

1 Einleitung **4**

2 Metadaten Aufbereiten **5**
2.1 MP3 Format 5
2.2 FLAC Format 6
2.3 Metadaten Parsen 7

3 Information Retrieval **9**
3.1 Index erstellen 9
3.2 Abfragen ausführen 11

4 Analyse der Abfrage **13**
4.1 Test Daten 14
4.2 Precision & Recall 15

5 Fazit **17**

Abbildungsverzeichnis **18**

Tabellenverzeichnis **19**

Quellenverzeichnis **20**

1 Einleitung

Im Rahmen dieser Arbeit werden Metadaten spezifischer Mediadateien mit der Information Retrieval Bibliothek Apache Lucene [3] indiziert und durchsucht.

Dafür werden in Kapitel 2 verfügbare Mediaformate und insbesondere deren Metadaten genauer untersucht. In dieser Analyse werden die zu indizierenden Attribute der Mediaformate ausgewählt. Zusätzlich wird aufgezeigt, wie die Metadaten der Mediadateien mit der Inhalt Erkennungs und Analyse Library Apache Tika [4] aufbereitet werden.

Anschliessend werden die extrahierten Metadaten in Kapitel 3 mittels Apache Lucene indiziert. Über die indizierten Attribute wird eine erste einfache Abfrage erstellt. Primär soll der Index verwendet werden um in Kapitel 4 aus einer kleinen Menge von Dokumenten die Genauigkeit und Trefferquote von Abfragen an den Index zu untersuchen.

Der gesamte Sourcecode zu dieser Arbeit und weitere hilfreiche Beispiele sind unter `https://github.com/kressi/search-media` verfügbar. Die Beispiele wurden alle mit Scala geschrieben. Scala ist eine Funktionale Programmiersprache welche in der Java Virtual Machine (JVM) ausgeführt wird. Das heisst, alle in der Arbeit verwendeten Java Packages können ohne weiteres verwendet werden.

Ausgangslage

In der zu indizierenden, privaten Media Library befinden sich Musik- und Videodateien in unterschiedlichen Formaten. Im Rahmen dieser Arbeit werden wir daraus zwei unterschiedliche Audioformate auswählen und indizieren. Diese Library liegt auf einem Network Attached Storage (NAS) und wird im lokalen Netzwerk den Clients zur verfügung gestellt. Bei den Clients ist das NAS über das WebDAV Protokoll unter einem lokalen Laufwerk geladen, der Zugriff auf die Dateien geschieht also wie wenn die Dateien auf der lokalen Festplatte gespeichert wären. Der Index wird nur lokal für einen Client erstellt.

2 Metadaten Aufbereiten

In der Media Library sind MP3 und FLAC Dateien weitaus am häufigsten vorhanden, weshalb wir nur diese beiden Audioformate genauer betrachten werden. Aus den Metadaten dieser Audioformate interessiert uns vor allem das Tag, die Metadaten zum Audio Track, wie der Interpret, der Titel oder das Album.

Um die Metadaten der Mediadateien zu betrachten und zu editieren, verwenden wir der Einfachheit halber das Tool EasyTAG [17]. Dieses Tool ist verfügbar für Linux und weitere Platformen, es ist intuitiv zu bedienen. Neben den Attributen sehen wir darin zum Beispiel in Abb. 2.1 auch, dass die Metadaten der gewählten Datei als Vorbis Tag gespeichert sind.

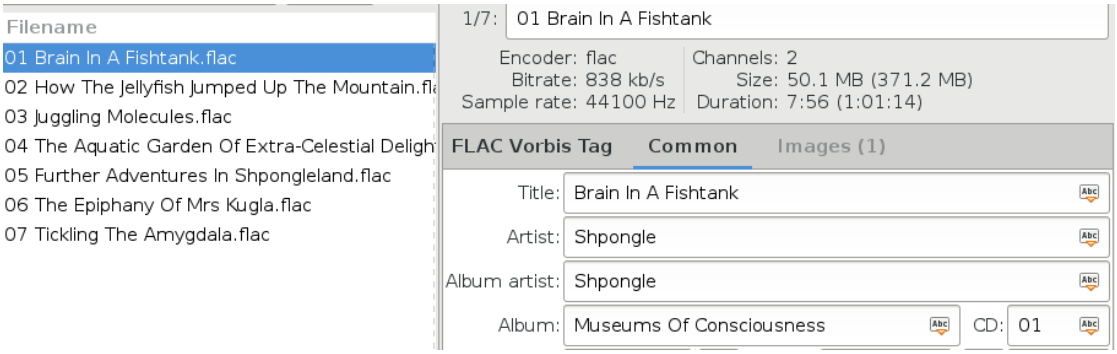

Abbildung 2.1: Metadaten mit EasyTAG editieren

2.1 MP3 Format

MPEG-1 oder MPEG-2 Audio Layer III allgemein bekannt als MP3 ist ein verlustbehafteteter Audio Codec. Die Movie Pictures Expert Group (MPEG) hat dieses Format entworfen. MP3 Dateien können aus mehreren Frames bestehen, ein solches Frame enthält jeweils einen MP3 Header und MP3 Daten. Der Header enthält Informationen wie zum Beispiel das Format, die Version des Formates oder die Bitrate. Der MP3 Standard enthält offiziell keinen Container für Tags. Üblicherweise werden Tags im ID3v1 oder ID3v2 Format an MP3 Dateien gehängt. ID3v1 ist ein 128 Byte langer Block welcher wie

in Tabelle 2.1 aufgebaut ist. Dieser Block enthält minimale Metadaten über das in der MP3 Datei gespeicherte Stück.

Feld	Bytes	Beschreibung
header	3	"TAG"
title	30	
artist	30	
album	30	
year	4	
comment	28 oder 30	Optional, letzte 2 Bytes für Track Nr.
zero-byte	1	Enthält 0, falls Track Nr. vorhanden
track	1	Track Nr. falls zero-byte gesetzt
genre	1	Index in einer Liste möglicher Genres, sonst 255

Tabelle 2.1: ID3v1 Layout

Im Gegensatz zu ID3v1 hat der ID3v2 keine fixe, sondern eine variable Länge. In einem ID3v2 können sehr viel mehr Attribute gespeichert werden, es existiert selbst für Bildcover ein Attribut. [20, 21]

2.2 FLAC Format

Free and Lossless Audio Codec (FLAC) ist ein verlustfreier Audio Codec. FLAC ist ein offenes Format und steht unter der GNU General Public License. Die grundlegende Struktur einer FLAC Datei ist wie folgt aufgebaut. [2]

- *flac* als String
- Block mit Metadaten zu Samplerate, Anzahl Channels etc.
- Keinen oder mehrere Metadaten Blocks, ein Block kann ein Bild, Vorbis Kommentar oder Track Informationen enthalten.
- Eines oder mehrere Audioframes.

Der Standard sieht den Vorbis Comment für die Tag Metadaten vor. Dieser Standard lässt zu, bis zu 2^{24} Bytes in einem Vorbis Comment zu speichern. Die Daten sollen in menschenlesbarem Format gespeichert werden, damit entfallen Probleme wie zum Beispiel das Genre im ID3 Tag Format, welches nur als Index gespeichert wird. Um aus diesem Index das Genre zu erhalten, muss man die überall gleiche Liste aller Genres kennen. [10]

2.3 Metadaten Parsen

Um die Metadaten zu den unterschiedlichen Audioformaten zu parsen wird die Library Apache Tika [4] verwendet. Diese Library bietet für verschiedenste Dokumente, darunter Textdokument, Source Code Dateien oder eben auch Media Dateien, Parser um aus den Dateien Metainformationen zu extrahieren.[9]

Mit dem MP3Parser können Metadaten aus MP3 Dateien extrahiert werden. Für FLAC Dateien existiert der externe FlacParser aus dem Gagravarr Package. Tika bietet mit dem AutoDetectParser zusätzlich einen Parser, welcher einem den Umstand abnimmt selbst entscheiden zu müssen, welcher Parser für eine bestimmte Datei verwendet werden soll. Wir werden nachfolgend nur mit diesem Parser arbeiten.

In nachfolgenden Listing sehen wir wie wir den Parser verwenden um damit die *metadata* aus dem *file* zu extrahieren.

```
1  val metadata = new Metadata
2  val handler = new BodyContentHandler
3  val context = new ParseContext
4  val parser  = new AutoDetectParser
5  val stream  = Try(new FileInputStream(file))
6
7  stream.map( s =>
8      catching(classOf[TikaException],
9               classOf[SAXException])
10     andFinally(s.close())
11     withTry(parser.parse(s, handler, metadata, context))
12 ) match {
13     case Success(_) =>
14        return metadata
15     case Failure(e:Throwable) => println(e.toString())
16 }
```

Die Metadaten zu einer von unseren Audiodateien sehen wir in Tabelle 2.2. Unter diesen Attributen fällt auf, dass viele mit *xmpdm* beginnen. Bezeichner vor dem Doppelpunkt sind Formate in welchen Tika dieses Attribut ablegt. *xmpdm* steht für XMP Digital Media. Extensible Metadata Platform (XMP) ist ein Standard für den Austausch von Metadaten.[19]

Im nachfolgende Kapitel werden wir die Attribute *xmpdem:genre*, *author*, *xmpdm:releasedate*, *title* und *xmpdmalbum* indizieren.

Attribut	Wert
xmpdm:genre	Hardstyle
x-parsed-by	org.apache.tika.parser.DefaultParser
creator	Showtek
xmpdm:album	Today is Tomorrow
xmpdm:tracknumber	05
xmpdm:releasedate	2007
meta:author	Showtek
xmpdm:artist	Showtek
dc:creator	Showtek
xmpdm:audiocompressor	MP3
title	FTS
xmpdm:audiochanneltype	Stereo
version	MPEG 3 Layer III Version 1
xmpdm:audiosamplerate	44100
channels	2
dc:title	FTS
author	Showtek
xmpdm:duration	165467.921875
content-type	audio/mpeg
samplerate	44100

Tabelle 2.2: Metadaten einer MP3 Datei

3 Information Retrieval

Im vorigen Kapitel haben wir uns die Metadaten der zu indizierenden Dokumente angeschaut. Diese Metadaten werden wir nun mit der Library Apache Lucene [3] indizieren und durchsuchen.

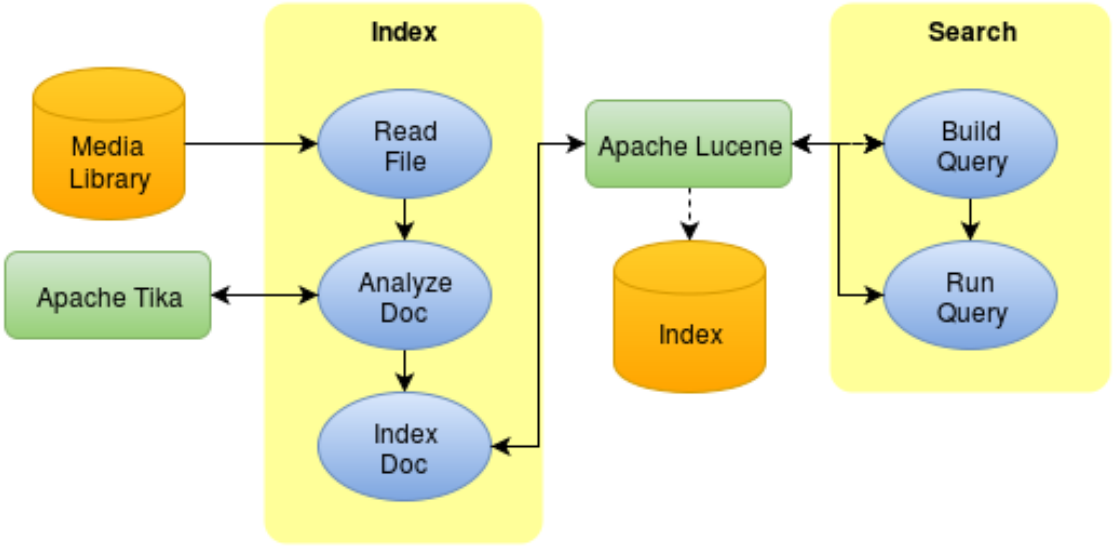

Abbildung 3.1: Architektur Information Retrieval Applikation

In Abb. 3.1 sehen wir die einzelnen Elemente welche in einer Information Retrieval Applikation mitwirken und in welchen Schritte welche Apache Libraries verwendet werden. Aus dieser Übersicht wollen wir uns nun dem letzten Schritt des Erstellens des Index zuwenden, dem Indizieren der Metadaten der Dokumente. [16]

3.1 Index erstellen

Die Metadaten, welche in Abschnitt 2.3 gelesen wurden, bilden pro gelesene Datei eine Menge von Attributen mit Werten. Aus einer solchen Menge wählen wir die gewünschten Attribute aus und fassen diese zusammen zu Dokumenten. Diese Dokumente werden dem Index hinzugefügt.

Shared Memory in Linux

Zugriffe auf den Index sollten möglichst schnell ausgeführt werden. Am schnellsten sind diese, wenn sich der Index im Arbeitsspeicher befindet. Da auf Linux Systemen standardmässig das Verzeichnis Shared Memory */dev/shm* im Arbeitsspeicher geladen wird, werden wir den Index darin erstellen. Es ist anzumerken, dass Dateien in diesem Verzeichnis verloren gehen wenn das System heruntergefahren wird.[11]

In folgendem Listing betrachten wir die Funktion, mit welcher wir aus dem *fileName* und den *metadata* ein Dokument erstellen. Allenfalls vorhandene *keywords* fügen wir dem Dokument als einzelne *StringFields* hinzu. Ein *StringField* ist ein String welcher als ganzes indiziert wird. Das *TextField* wird bei Leerzeichen oder anderen Trennzeichen gesplittet und als mehrere *Strings* indiziert. Mit dem *boolean* Wert *Store.YES* sagen wir, dass der Wert des indizierten Attributes auch abgelegt werden soll. Alle Attribute welche leer sind werden nicht indiziert. Weiter werden alle Attribute welche nicht explizit indiziert werden verworfen.

```
1  def mkDocument (fileName:String, metadata:Metadata): Document = {
2
3      val fieldPat = "([a-z0-9]+):([a-z0-9]+)".r
4      val xmpdmAttrs = Array("album", "releasedate", "genre")
5      val doc = new Document()
6      doc.add(new StoredField("file", fileName))
7
8      for (key <- metadata.names()) {
9          val name = key.toLowerCase()
10         val value = metadata.get(key)
11         if (!StringUtils.isBlank(value)) {
12             name match {
13                 case "keywords" =>
14                     for (keyword <- value.split(",?(\\s+)")) {
15                         doc.add(new StringField(name, keyword, Store.YES))
16                     }
17                 case "title" | "author" | "composer" =>
18                     doc.add(new TextField(name, value, Store.YES))
19                 case fieldPat("xmpdm", attr) if xmpdmAttrs contains attr
                       =>
20                     doc.add(new TextField(attr, value, Store.YES))
21                 case _ => ()
22             }
23         }
24     }
25     doc
26 }
```

In einem Loop iterieren wir nun rekursiv über alle Dateien in der Media Library, erstellen zu jeder FLAC und MP3 Datei ein Dokument und fügen dieses Dokument zum Index

hinzu.

Rekursiv über Dateien Iterieren wird mit der *listFiles* Methode aus dem Apache Package *org.apache.commons.io.FileUtils* ausgeführt. Diese Methode erlaubt auf einfache Weise rekursiv über alle Dateien eines Verzeichnisses zu iterieren. [7]

3.2 Abfragen ausführen

Auf dem soeben erstellten Index führen wir nun eine Abfrage aus. Die Abfrage soll über mehrere Attribute ausgeführt werden.

Abfragen müssen der Lucene Query Parser Syntax entsprechen. Damit können unter anderem Proximity Searches, Range Searches oder Fuzzy Searches ausgeführt werden, es können zu durchsuchende Attribute festgelegt werden oder auch einzelne Terme miteinander verknüpft werden. [8]

Für unsere Abfrage wollen wir für das Erste nur nach einem Begriff in mehreren Attributen suchen. Dazu müssen wir den *MultiFieldQueryParser* verwenden. Nachfolgend parsed der *parser* einen Abfrage String und gibt eine *Query* zurück. Diese Query *q* übergeben wir dem *IndexSearcher*. Von dem *collector* werden danach die gefundendenen Resultate geholt. [6, 18]

```
1  def apply() {
2
3      val indexDir = Paths.get(Config.indexPath)
4      val index = FSDirectory.open(indexDir)
5
6      // Build a Query object
7      val fields = Array("album", "author", "composer", "album", "title")
8      val analyzer = new StandardAnalyzer()
9      val parser = new MultiFieldQueryParser(fields, analyzer)
10
11     Try(parser.parse("Shpongle")) match {
12         case Success(q) =>
13             val hitsPerPage = 10
14             val reader = DirectoryReader.open(index)
15             val searcher = new IndexSearcher(reader)
16             val collector = TopScoreDocCollector.create(hitsPerPage)
17             searcher.search(q, collector)
18
19             println("total hits: " + collector.getTotalHits())
20             val hits = collector.topDocs().scoreDocs
21             for (hit <- hits) {
```

```
22                  val doc = reader.document(hit.doc)
23                  println(doc.get("file") + " (" + hit.score + ")")
24              }
25          case Failure(e) => println(e.toString())
26      }
27  }
```

Wie wir nun gesehen haben, erlaubt einem die Apache Lucene Library auf einfache Weise, aus Dokumenten einen Index zu erstellen und an diesen Abfragen zu stellen. Im nächsten Kapitel wollen wir den so erstellten Index der Media Library etwas genauer betrachten.

4 Analyse der Abfrage

Um dieses Information Retrieval System zu bewerten, möchten wir die Precision (Genaugkeit) und den Recall (Trefferquote) für typische Abfragen an das System bestimmen. Um die dazu notwendigen Daten zu erheben verwenden wir das Entwicklungs und Analyse Tool für Lucene Indizes Luke.

Luke kann unter anderem verwendet werden um Dokumente im Index zu betrachten (Abb. 4.1) oder um Abfragen zu erstellen (Abb. 4.2). [13] Luke [1] scheint vom ursprünglichen Entwickler Bialecki nicht mehr maintained zu werden. Von Kan ist eine Version verfügbar in welcher auch Indizes von Lucene in der aktuellen Version gelesen werden können. [14]

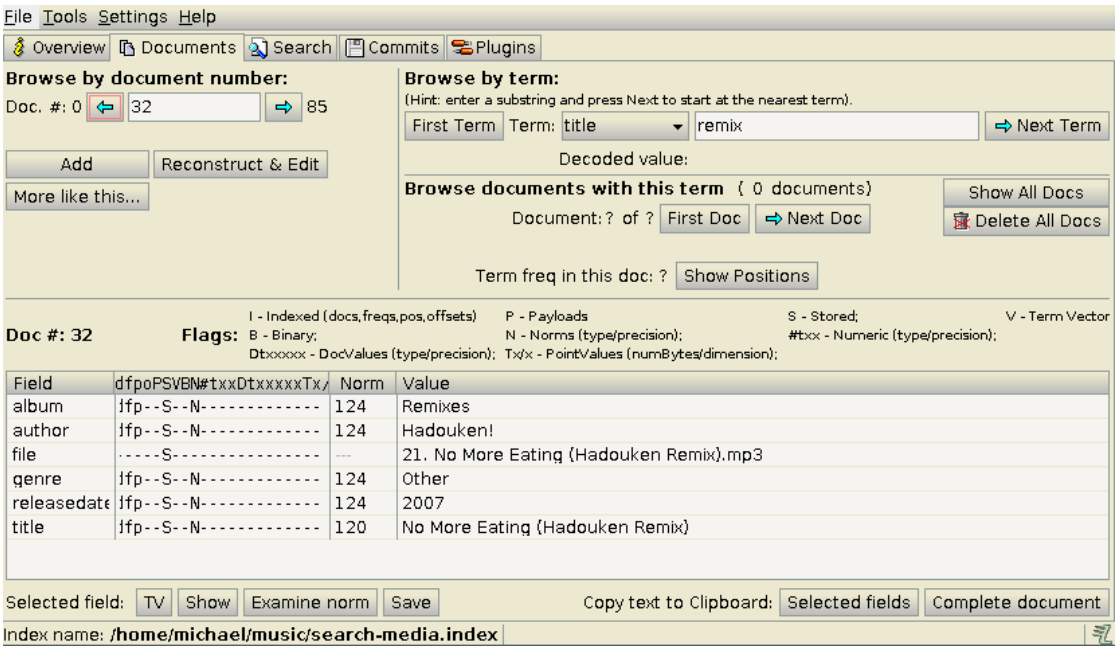

Abbildung 4.1: Ansicht Dokumente in Luke

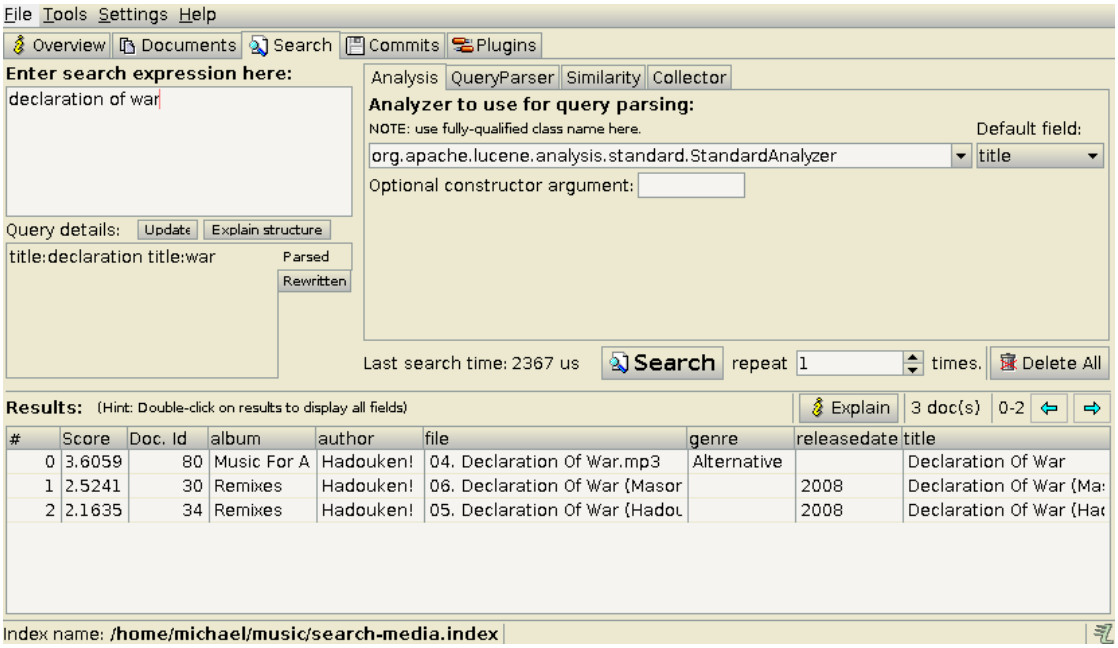

Abbildung 4.2: Ansicht Suche in Luke

4.1 Test Daten

Damit man eine verlässliche Aussage über die Precision und den Recall machen kann, muss man die indizierten Dokumente genau kennen. Das heisst wir müssen genau wissen welches die erwarteten Treffer einer Abfrage sind.[15, S. 152]

Aus der Media Library wählen wir dazu eine Menge von 85 Dateien aus und erstellen einen Index, worin nur diese Dateien indiziert werden.

Abfrage	**#Resultat**
:	85
author:hadouken	66
author:hadouken AND title:war	3
author:hadouken AND NOT title:war	63
title:girl*	4
title:liquid AND releasedate:2007	4
album:"museums of consciousness"	7

Tabelle 4.1: Abfragen und Erwartete Resultate

Mit den Abfragen in Tabelle 4.1, wollen wir ein Spektrum von Abfragen abdecken, welches nahe am tatsächlichen Gebrauch eines solchen Index ist. Die Anzahl der erwarteten

Results wird mit EasyTAG gezählt, es werden dazu nicht die Dokumente in Luke betrachtet. Damit würden allfällige Fehler, welche beim Indizieren oder beim Abfragen mit Luke entstehen könnten, kaschiert.

4.2 Precision & Recall

Die vorbereiteten Abfragen führen wir nun mit Luke aus. Von den die erhaltenen Resultate vergleichen wir dann mit den gespeicherten Resultaten der in Abschnitt 4.1 vorbereiteten Abfragen.

Seien e die erwarteten Resultate und r das erreichte Resultat. Dann sind die Precision und dre Recall wie folgt definiert.

Recall Die Fähigkeit die erwarteten Resultate zu liefern. $r = \frac{|r \cap e|}{|e|}$

Precision Die Korrektheit der gelieferten Resultate. $p = \frac{|r \cap e|}{|r|}$

[12, S. 63]

Abfrage	$\|r\|$	$\|e\|$	$\|r \cap e\|$	p	r
:	85	85	85	1	1
author:hadouken	66	66	66	1	1
author:hadouken AND title:war	3	3	3	1	1
author:hadouken AND NOT title:war	63	63	63	1	1
title:girl*	4	4	4	1	1
title:liquid AND releasedate:2007	4	4	4	1	1
album:"museums of consciousness"[1]	0	7	0	–	0

Tabelle 4.2: Abfragen und Erreichte Resultate

Diese Resultate scheinen im ersten Moment etwas überraschend, lassen sich jedoch erklären. Bis auf die letzte Zeile sind jeweils genau die erwarteten Resultate erreicht worden. Dafür gibt es unterschiedliche Gründe. Die Menge ist relativ klein, die Wahrscheinlichkeit sinkt dadurch Ausreisser zu treffen. Die Abfragen sind einfach, es wurden keine N-Gramme geladen, es existieren keine Ähnlichkeitsabfragen. Dies alles sind Features welche eine Suche in verschiedenen Bereichen für einen Endanwender bequemer jedoch auch ungenauer machen.

[1] Diese Abfrage muss mit dem Luke XML Query Parser ausgeführt werden, in regulären Abfragen entfernt Luke *of* aus dem String, so dass nur noch nach album:"museums consciousness"gesucht wird.

Dann ist in diesem noch das Resultate, bei welchem kein Hit gefunden wurde. Dies liegt an der String Suche, in Abschnitt 3.1 haben wir die meisten Attribute des Dokuments, darunter auch *album*, als *TextField* erstellt. Ein *TextField* wird in einzelne Worte aufgeteilt, davon werden nur die einzelnen Worte indiziert, nicht der gesamte String. Um auf diesem Feld auch eine String Suche zu ermöglichen, könnte als mögliche Lösung ein zusätzliches *StringField* mit dem gesamten String zum Dokument hinzugefügt werden. Dies müsste jedoch nicht auch gespeichert werden.

5 Fazit

In dieser Arbeit wurden zuerst die Formate der Mediadateien und deren Container für Tags analysiert. Es ist interessant die nicht geringen Unterschiede zwischen unterschiedlichen Formaten zu erkennen. Genauso ist es spannend zu sehen, dass ID3 Tags nicht nur für das MP3 und Vorbis Comments nicht nur für FLAC Format verwendet werden. Sobald man jedoch damit beginnt die Metadaten mit der Apache Tika Library zu parsen, stellt man fest, dass zumindest im Bereich von weit verbreiteten Mediaformaten, auch ohne jegliches Wissen über die verwendeten Tag Container oder deren Aufbau weiterkommt. Selbst mit Wissen über die Tag Container ist es einfacher den *AutoDetectParser* zu verwenden. Dies macht es unglaublich leicht Metadaten aus Dateien zu extrahieren.

Als nächster Schritt wurden die Metadaten indiziert. Auch dieser Schritt geht mit den Apache Libraries leicht von der Hand, zumindest solange man die Standardkonfiguration verwendet. Dies ist sehr angenehmn, so sieht man schnell ein Resultat, was motiviert weiterzumachen. Genauso sieht man schnell ein Resultat wenn man eine Abfrage erstellt.

Für die Analyse wurde das Tool Luke verwendet, der Funktionsumfang dieses Tools ist für die Entwicklung und die Analyse von Indizes unglaublich wertvoll. Um jedoch einmal erst soweit zu kommen hat es sich erstaunlich kompliziert herausgestellt eine Version von Luke zu finden, welche die aktuellste Version[1] des Lucene Index lesen kann. Dies mag daran liegen, dass normalerweise Elasticsearch oder Solr verwendet werden, welche gleich die eigenen Analyse Tools mitliefern. Bei der Analyse selbst wurden keine so hohe Precision und Recall Werte erwartet. Wenn man jedoch das Information Retrieval System betrachtet ist es logisch solche Werte zu erreichen.

[1] Zum Zeitpunkt dieser Arbeit war 6.0.0 die aktuellste stabile Version von Apache Lucene.

Abbildungsverzeichnis

2.1 Metadaten mit EasyTAG editieren 5

3.1 Architektur Information Retrieval Applikation 9

4.1 Ansicht Dokumente in Luke 13
4.2 Ansicht Suche in Luke 14

Tabellenverzeichnis

2.1 ID3v1 Layout 6
2.2 Metadaten einer MP3 Datei 8

4.1 Abfragen und Erwartete Resultate 14
4.2 Abfragen und Erreichte Resultate 15

Quellenverzeichnis

[1] Andrzej Bialecki. *Luke - Lucene Index Toolbox*. 14. Nov. 2008. URL: http://www.getopt.org/luke/ (besucht am 20. 03. 2016).

[2] Josh Coalson. *Format Overview — Xiph.Org Foundation*. URL: https://xiph.org/flac/documentation_format_overview.html (besucht am 01. 06. 2016).

[3] The Apache Software Foundation. *Apache Lucene*. 2016. URL: https://lucene.apache.org/ (besucht am 20. 03. 2016).

[4] The Apache Software Foundation. *Apache Tika*. 2016. URL: http://tika.apache.org/ (besucht am 20. 03. 2016).

[5] The Apache Software Foundation. *Apache Tika API Usage Examples*. 2016. URL: http://tika.apache.org/1.12/examples.html (besucht am 01. 04. 2016).

[6] The Apache Software Foundation. *Lucene 6.0.0 core API*. 2016. URL: https://lucene.apache.org/core/6_0_0/core/index.html (besucht am 01. 04. 2016).

[7] The Apache Software Foundation. *org.apache.commons.io.FileUtils*. 1. Jan. 2016. URL: https://commons.apache.org/proper/commons-io/javadocs/api-release/org/apache/commons/io/FileUtils.html#listFiles(java.io.File,%20java.lang.String[],%20boolean) (besucht am 01. 06. 2016).

[8] The Apache Software Foundation. *Query Parser Syntax*. 2016. URL: https://lucene.apache.org/core/6_0_1/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package.description (besucht am 01. 04. 2016).

[9] The Apache Software Foundation. *Supported Document Formats*. 2016. URL: https://tika.apache.org/1.13/formats.html (besucht am 20. 03. 2016).

[10] Xiph.Org Foundation. *Ogg Vorbis I format specification: comment field and header specification*. 2005. URL: http://www.xiph.org/vorbis/doc/v-comment.html (besucht am 01. 06. 2016).

[11] Vivek Gite. *What Is /dev/shm And Its Practical Usage*. 29. Juni 2012. URL: http://www.cyberciti.biz/tips/what-is-devshm-and-its-practical-usage.html.

[12] Andreas Henrich. *Grundlagen, Modelle und Anwendungen*. Otto-Friedrich-Universität Bamberg, 7. Jan. 2008. URL: https://www.uni-bamberg.de/fileadmin/uni/fakultaeten/wiai_lehrstuehle/medieninformatik/Dateien/Publikationen/2008/henrich-ir1-1.2.pdf (besucht am 01. 04. 2016).

[13] Mitzimorris. *Using Luke the Lucene Index Browser to develop Search Queries*. 24. Juli 2012. URL: https://lingpipe-blog.com/2012/07/24/using-luke-the-lucene-index-browser-to-develop-search-queries/ (besucht am 20. 03. 2016).

[14] Dmitry Kan. *Luke: Lucene Toolbox Project — Github*. 19. Apr. 2016. URL: https://github.com/DmitryKey/luke (besucht am 23. 04. 2016).

[15] Christopher D. Manning, Prabhakar Raghavan und Hinrich Schütze. Online. Cambridge, England: Cambridge University Press, 1. Apr. 2009. URL: http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf (besucht am 01. 04. 2016).

[16] Ana-Maria Mihalceanu. *Understanding Information Retrieval by Using Apache Lucene and Tika - Part 1*. 22. Okt. 2014. URL: https://dzone.com/articles/understanding-information (besucht am 01. 04. 2016).

[17] The GNOME Project. *EasyTAG*. 2016. URL: https://wiki.gnome.org/Apps/EasyTAG (besucht am 20. 03. 2016).

[18] John Ferguson Smart. *A Short Introduction to Lucene*. Scalable Analytics Institute (ScAi). 14. Apr. 2006. URL: http://oak.cs.ucla.edu/cs144/projects/lucene/ (besucht am 01. 04. 2016).

[19] Wikipedia. *Extensible Metadata Platform — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=Extensible_Metadata_Platform&oldid=723375225 (besucht am 20. 03. 2016).

[20] Wikipedia. *ID3 — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=ID3&oldid=720443199 (besucht am 01. 06. 2016).

[21] Wikipedia. *MP3 — Wikipedia, The Free Encyclopedia*. 2016. URL: https://en.wikipedia.org/w/index.php?title=MP3&oldid=725267687 (besucht am 01. 06. 2016).