Information Harvesting in MoK: information extraction using text mining

Sistemi Distribuiti
Omicini, Mariani

Filippo Alberto Brandolini

Indice

Abstract

Introdurre i concetti di information harvesting, information extraction e data/text-mining. Valutare l'importanza di queste aree informatiche nell'ambito del progetto MoK (Molecules of Knowledge), sperimentare alcuni tools di recupero delle informazioni e di elaborazione di testi con particolare approfondimento di quelli più adatti a supportare il progetto di cui sopra.

Keywords: information harvesting, information extraction, data mining, text mining, Molecules of Knowledge.

Capitolo 1

Data Mining

Il Data Mining (a.k.a. Knowledge Discovery from Data) è l'insieme delle pratiche che analizzano i sistemi e i database informativi allo scopo di estrarre modelli, schemi e tendenze, mediante l'utilizzo di algoritmi matematici che segmentano il set informativo e valutano le probabilità di eventi futuri [1].
Questo campo dell'informatica combina tecniche di statistica e intelligenza artificiale (come reti neurali e apprendimento automatico) con strumenti di gestione dei database, ed è ampiamente utilizzato in ambito di ricerche scientifiche, ambienti di business e sistemi di sicurezza [6].

1.1 Text Mining

Quando i modelli vengono estratti dal linguaggio naturale, anziché da database strutturati di eventi, si parla di Text Mining (a.k.a. Knowledge Discovery from Text): si ricercano nuove informazioni tramite l'analisi di risorse testuali. Per prima cosa, dunque, si estraggono fatti ed eventi dal testo, quindi si procede con la formulazione di ipotesi relative a schemi, modelli e tendenze del set informativo [2].
Gli strumenti di Text Mining sono utilizzati allo scopo di:

- estrarre informazioni rilevanti da un documento testuale, utilizzando algoritmi e funzioni analitiche derivanti dalle tecniche di *information*

retrieval e di *pattern matching* [2];

- scoprire ulteriori informazioni e relazioni tra entità confrontando le informazioni estratte dai documenti [2];
- classificare, organizzare e raggruppare i documenti a seconda del loro contenuto informativo estratto [3] [4].

Un sistema Text Mining è composto da 3 componenti principali [2]:

- **Information Feeders**: permettono al sistema di connettersi alle risorse web e alle collezioni di documenti destinate all'analisi.
- **Intelligent Tagging**: abilita la distillazione dei documenti consentendo il rilevamento delle strutture e delle informazioni di base che li hanno generati. Si estraggono dunque termini, categorie, informazioni semantiche e strutturali (es: ADIOS, Automatic Distillation of Structure [5]).
- **Business Intelligence Suite**: consolida le informazioni estratte dalle fonti e permette l'analisi simultanea dell'intero ambiente informativo.

Capitolo 2

Tools di Information Retrieval, Extraction and Mining

2.1 Terrier

Terrier è un motore di ricerca ad alte prestazioni che consente un rapido sviluppo di applicazioni di information retrieval su larga scala [7]. » Questo tool è adatto alla prima fase di harvesting, l'information retrieval con web crawlers.

2.2 Apache Marmotta

La piattaforma Apache Marmotta può essere utilizzata, estesa e implementata facilmente per la pubblicazione di Linked Data e la realizzazione di applicazioni basate su di essi [8]. » Questo tool è adatto all'estrazione di informazioni semantiche dalle collezioni di dati. Pertanto, presumo sia più adatto, rispetto a Terrier, al focus del mio subproject, ossia il text-mining.

2.3 OpenNLP

OpenNLP è uno strumento basato su tecniche di autoapprendimento per l'elaborazione di testi in linguaggio naturale; permette, ad esempio, il raggrupamento per token, ossia il processo di suddivisione di un flusso testuale in simboli, parole o frasi che lo costituiscono e la classificazione degli elementi estratti dalle informazioni [10]. » Questo tool è adatto all'elaborazione delle informazioni secondo tecniche di mining. Pertanto, presumo sia più adatto al focus del mio subproject, il text-mining, sia rispetto a Terrier, sia rispetto a Marmotta.

2.4 KEA

KEA (Keyphrase Extraction Algorithm) è un algoritmo per l'estrazione automatica di keyphrase[1] da un testo. L'algoritmo, basato su tecniche di autoapprendimento, per prima cosa identifica le stringhe candidate ad essere keyphrase, usando un modello ricavato da documenti le cui keyphrase sono conosciute, quindi, sfrutta tale modello per stabilire quali fra le stringhe candidate del nuovo documento possano essere valide keyphrase [9]. » Tra i tools studiati, probabilmente questo è il più adatto all'estrazione automatica di informazioni dalle collezioni di dati, e all'utilizzo di tecniche di mining. Pertanto, presumo sia più adatto al focus del mio subproject, il text-mining, sia rispetto ai tool Terrier e Marmotta, sia rispetto al più simile OpenNLP.

2.5 MAUI

MAUI è un tool per keyword extraction e subject indexing, ed è probabilmente la migliore alternativa a KEA, ad oggi non più aggiornato e divenuto quindi obsoleto. » Questo tool è specificamente pensato per l'estrazione automatica di informazioni dalle collezioni di dati, e all'utilizzo di tecniche

[1] Keyphrase: collezione di keywords.

di mining. Pertanto si candida ad essere il miglior tool per il mio subproject, il text-mining, sia rispetto ai primi tool studiati, sia rispetto a KEA, programma ormai non più aggiornato.

2.6 Apache Tika

Apache Tika è una libreria utilizzata per il parsing di documenti e l'estrazione di contenuti e metadati a partire da testi in linguaggio naturale. » Questo tool è adatto all'estrazione di informazioni semantiche dalle collezioni di dati, è costantemente aggiornato e può effettuare l'analisi dei contenuti su innumerevoli tipi di documenti, pertanto sarà interessante effettuare un confronto con MAUI.

Capitolo 3

MAUI vs Apache Tika

3.1 Requisiti e funzionalità dei tool

1) Quali tipi di documenti possono elaborare? Che tipo di training è necessario?

MAUI: richiede un modello di training per effettuare le operazioni di analisi dei contenuti e keyword extraction. In questo lavoro sfrutto il modello e i documenti di training presi da una versione modificata della repository originale di *RAKE*[1], una libreria in Python che implementa l'omonimo algoritmo *Rapid Automatic Keyword Extraction*.

```
java -Xmx1024m -jar maui.jar train -l data/docs/fao_train/
   -m data/models/keyword_extraction_model -v none -o 2
20 giu 2016 14:21:14  INFO MauiModelBuilder - Building
   model with options:
20 giu 2016 14:21:14  INFO MauiModelBuilder - -l data/docs/
   fao_train/ -m data/models/keyword_extraction_model -v
   none -f null -e default -i en -z -x 5 -y 1 -o 2 -s com.
   entopix.maui.stopwords.StopwordsEnglish -t com.entopix.
   maui.stemmers.PorterStemmer
20 giu 2016 14:21:14  INFO MauiModelBuilder - -- Building
   the model...
```

[1]https://github.com/zelandiya/RAKE-tutorial/tree/master/data

```
20 giu 2016 14:21:14  INFO MauiModelBuilder - -- Adding
   documents as instances...
20 giu 2016 14:21:16  INFO MauiModelBuilder - -- Building
   the model...
20 giu 2016 14:21:23  INFO MauiModelBuilder - Model built.
   Saving the model...
20 giu 2016 14:21:23  INFO MauiModelBuilder - Done!
application/xhtml+xml
```

Apache Tika: può analizzare più di un migliaio di documenti (es: txt, ppt, xls, pdf, html, jpeg) e non richiede un training iniziale per essere operativo.

2) Che funzionalità offrono?

MAUI: Può estrarre keywords da un documento contenente un testo in linguaggio naturale[2], effettuare subject indexing e automatic tagging.

```
java -Xmx1024m -jar maui.jar run data/docs/fao_test/w2167e.
   txt -m data/models/keyword_extraction_model -v none -n
   8
Keyword: food security 0.7637371615312792
Keyword: food production 0.5992965367965367
Keyword: forest management 0.4277212321802565
Keyword: forest resources 0.3992965367965367
Keyword: household 0.31886138513376366
Keyword: women 0.225714611513248
Keyword: consumption 0.22042805024842593
Keyword: nutrition 0.2139865915472591
application/xhtml+xml
```

Apache Tika:

- può rilevare il tipo di documento analizzato

```
java -jar tika.jar --detect http://tika.apache.org/
application/xhtml+xml
```

- può estrarre il testo contenuto in un documento

[2] `w2167e.txt` contiene un testo di 496 linee e 60154 caratteri intitolato *Considering nutrition in National Forestry Programmes.*

```
java -jar tika.jar --text test1.txt
Apache Tika text document content sample...
```

```
java -jar tika.jar --text http://tika.apache.org/
Apache website homepage content...
```

```
java -jar tika.jar --text-main http://tika.apache.org/
Apache website homepage MAIN content
```

Il parametro `text-main` specifica l'intenzione di estrarre solo il contenuto principale della pagina (in questo caso, non i vari menu e submenu del sito).

- può dedurre la lingua in cui è scritto un testo

```
java -jar tika.jar --language test1.txt
it
```

- può estrarre i metadati di un documento

```
java -jar tika.jar --metadata test1.txt
Content-Encoding: UTF-8
Content-Length: 148
Content-Type: text/plain; charset=UTF-8
X-Parsed-By: org.apache.tika.parser.DefaultParser
X-Parsed-By: org.apache.tika.parser.txt.TXTParser
resourceName: test1.txt
```

```
java -jar tika.jar --metadata http://tika.apache.org/
Content-Encoding: UTF-8
Content-Length: 30741
Content-Type: application/xhtml+xml; charset=UTF-8
Content-Type-Hint: text/html; charset=UTF-8
X-Parsed-By: org.apache.tika.parser.DefaultParser
X-Parsed-By: org.apache.tika.parser.html.HtmlParser
dc:title: Apache Tika - Apache Tika
title: Apache Tika - Apache Tika
```

- l'elenco dei parser utilizzati dal tool sono verificabili specificando il parametro `list-parsers`

```
java -jar tika.jar --list-parsers
   org.apache.tika.parser.AutoDetectParser (Composite
      Parser):
      org.apache.tika.parser.DefaultParser (Composite
         Parser):
         org.apache.tika.parser.asm.ClassParser
         org.apache.tika.parser.audio.AudioParser
         org.apache.tika.parser.audio.MidiParser
         org.apache.tika.parser.chm.ChmParser
         org.apache.tika.parser.code.SourceCodeParser
         org.apache.tika.parser.crypto.Pkcs7Parser
         org.apache.tika.parser.dif.DIFParser
         org.apache.tika.parser.dwg.DWGParser
         org.apache.tika.parser.epub.EpubParser
         org.apache.tika.parser.executable.
            ExecutableParser
         org.apache.tika.parser.feed.FeedParser
         org.apache.tika.parser.font.
            AdobeFontMetricParser
         org.apache.tika.parser.font.TrueTypeParser
         org.apache.tika.parser.gdal.GDALParser
         org.apache.tika.parser.geo.topic.GeoParser
         org.apache.tika.parser.geoinfo.
            GeographicInformationParser
         org.apache.tika.parser.grib.GribParser
         org.apache.tika.parser.hdf.HDFParser
         org.apache.tika.parser.html.HtmlParser
         org.apache.tika.parser.image.BPGParser
         org.apache.tika.parser.image.ICNSParser
         org.apache.tika.parser.image.ImageParser
         org.apache.tika.parser.image.PSDParser
         org.apache.tika.parser.image.TiffParser
         org.apache.tika.parser.image.WebPParser
         org.apache.tika.parser.iptc.IptcAnpaParser
         org.apache.tika.parser.isatab.ISArchiveParser
         org.apache.tika.parser.iwork.IWorkPackageParser
         org.apache.tika.parser.jpeg.JpegParser
         org.apache.tika.parser.mail.RFC822Parser
```

```
org.apache.tika.parser.mat.MatParser
org.apache.tika.parser.mbox.MboxParser
org.apache.tika.parser.mbox.OutlookPSTParser
org.apache.tika.parser.microsoft.JackcessParser
org.apache.tika.parser.microsoft.OfficeParser
org.apache.tika.parser.microsoft.OldExcelParser
org.apache.tika.parser.microsoft.TNEFParser
org.apache.tika.parser.microsoft.ooxml.
   OOXMLParser
org.apache.tika.parser.mp3.Mp3Parser
org.apache.tika.parser.mp4.MP4Parser
org.apache.tika.parser.netcdf.NetCDFParser
org.apache.tika.parser.odf.OpenDocumentParser
org.apache.tika.parser.pdf.PDFParser
org.apache.tika.parser.pkg.CompressorParser
org.apache.tika.parser.pkg.PackageParser
org.apache.tika.parser.pkg.RarParser
org.apache.tika.parser.rtf.RTFParser
org.apache.tika.parser.txt.TXTParser
org.apache.tika.parser.video.FLVParser
org.apache.tika.parser.xml.DcXMLParser
org.apache.tika.parser.xml.FictionBookParser
org.gagravarr.tika.FlacParser
org.gagravarr.tika.OggParser
org.gagravarr.tika.OpusParser
org.gagravarr.tika.SpeexParser
org.gagravarr.tika.TheoraParser
org.gagravarr.tika.VorbisParser
```

3) Cosa prendono in input?

MAUI: prende in input documenti di testi semplice, mentre fallisce il parsing di documenti strutturati, come ad esempio .css o .docx:

```
java -Xmx1024m -jar maui.jar run w2167e.docx -m data/models
   /keyword_extraction_model -v none -n 8
Keyword: PK 0.19144690375664064
Keyword: Z 0.11858962299540507
Keyword: k 0.11618691995448405
```

```
Keyword: x 0.11618691995448405
Keyword: xF 0.10582452835300418
Keyword: eX 0.06716940651248719
Keyword: xg 0.02480120256924669
Keyword: S 0.02480120256924669
```

Apache Tika: prende in input sia documenti di testo semplice, sia documenti strutturati (es: .css, .html, .docx).

4) Forniscono API Java a supporto? Che tipo di file producono in uscita?

Apache Tika: la classe "facciata" Tika *(org.apache.tika)* permette di accedere alle funzionalità del tool da Java. Si può quindi far effettuare a Tika il parsing di uno specifico documento, e avere il risultato sotto forma di testo semplice [11]:

```
1 public String parseToStringExample() throws IOException,
    SAXException, TikaException {
2     Tika tika = new Tika();
3     try (InputStream stream = ParsingExample.class.
          getResourceAsStream("test.doc")) {
4         return tika.parseToString(stream);
5     }
6 }
```

Oppure è possibile richiedere al parser di produrre il risultato in forma XHTML, restituito come stringa dalla funzione:

```
1 public String parseToHTML() throws IOException,
    SAXException, TikaException {
2     ContentHandler handler = new ToXMLContentHandler();
3
4     AutoDetectParser parser = new AutoDetectParser();
5     Metadata metadata = new Metadata();
6     try (InputStream stream = ContentHandlerExample.class.
          getResourceAsStream("test.doc")) {
7         parser.parse(stream, handler, metadata);
8         return handler.toString();
```

```
9       }
10  }
```

La stessa funzionalità di rilevamento lingua di cui a pag. 9 è disponibile anche tramite API Java grazie alla classe LanguageIdentifier:

```
1  public String identifyLanguage(String text) {
2      LanguageIdentifier identifier = new LanguageIdentifier(
           text);
3      return identifier.getLanguage();
4  }
```

Segue quindi un esempio completo di programma Java per effettuare l'estrazione di metadati da un'immagine:

```
1  import java.io.File;
2  import java.io.FileInputStream;
3  import java.io.IOException;
4
5  import org.apache.tika.exception.TikaException;
6  import org.apache.tika.metadata.Metadata;
7  import org.apache.tika.parser.AutoDetectParser;
8  import org.apache.tika.parser.ParseContext;
9  import org.apache.tika.parser.Parser;
10 import org.apache.tika.sax.BodyContentHandler;
11
12 import org.xml.sax.SAXException;
13
14 public class GetMetadata {
15
16    public static void main(final String[] args) throws
         IOException, TikaException {
17
18       //Assume that boy.jpg is in your current directory
19       File file = new File("test-image-sd2016.jpg");
20
21       //Parser method parameters
22       Parser parser = new AutoDetectParser();
23       BodyContentHandler handler = new BodyContentHandler()
            ;
```

```
24        Metadata metadata = new Metadata();
25        FileInputStream inputstream = new FileInputStream(
              file);
26        ParseContext context = new ParseContext();
27
28        parser.parse(inputstream, handler, metadata, context)
              ;
29        System.out.println(handler.toString());
30
31        //getting the list of all meta data elements
32        String[] metadataNames = metadata.names();
33
34        for(String name : metadataNames) {
35           System.out.println(name + ": " + metadata.get(name
                 ));
36        }
37     }
38  }
```

Il programma qui descritto estrae i metadati dal documento specificato e fornisce un output come di seguito:

```
Transparency Alpha: nonpremultipled
tiff:ImageLength: 1094
Compression CompressionTypeName: deflate
Data BitsPerSample: 8 8 8 8
Data PlanarConfiguration: PixelInterleaved
Dimension VerticalPixelSize: 0.17639795
IHDR: width=1760, height=1094, bitDepth=8, colorType=
   RGBAlpha, compressionMethod=deflate, filterMethod=
   adaptive, interlaceMethod=none
iCCP: profileName=ICC Profile, compressionMethod=deflate
Chroma ColorSpaceType: RGB
tiff:BitsPerSample: 8 8 8 8
Content-Type: image/png
height: 1094
X-Parsed-By: org.apache.tika.parser.DefaultParser
pHYs: pixelsPerUnitXAxis=5669, pixelsPerUnitYAxis=5669,
   unitSpecifier=meter
```

```
Text TextEntry: keyword=XML:com.adobe.xmp, value=<x:xmpmeta
    xmlns:x="adobe:ns:meta/" x:xmptk="XMP Core 5.4.0">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
      syntax-ns#">
      <rdf:Description rdf:about=""
            xmlns:exif="http://ns.adobe.com/exif/1.0/">
         <exif:PixelXDimension>1760</exif:PixelXDimension>
         <exif:PixelYDimension>1094</exif:PixelYDimension>
      </rdf:Description>
   </rdf:RDF>
</x:xmpmeta>, language=, compression=none
Dimension PixelAspectRatio: 1.0
iTXt iTXtEntry: keyword=XML:com.adobe.xmp, compressionFlag=
   false, compressionMethod=0, languageTag=,
   translatedKeyword=, text=<x:xmpmeta xmlns:x="adobe:ns:
   meta/" x:xmptk="XMP Core 5.4.0">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
      syntax-ns#">
      <rdf:Description rdf:about=""
            xmlns:exif="http://ns.adobe.com/exif/1.0/">
         <exif:PixelXDimension>1760</exif:PixelXDimension>
         <exif:PixelYDimension>1094</exif:PixelYDimension>
      </rdf:Description>
   </rdf:RDF>
</x:xmpmeta>
Compression NumProgressiveScans: 1
Dimension HorizontalPixelSize: 0.17639795
Chroma BlackIsZero: true
Compression Lossless: true
width: 1760
Dimension ImageOrientation: Normal
tiff:ImageWidth: 1760
Chroma NumChannels: 4
Data SampleFormat: UnsignedIntegral
```

Figura 3.1: Immagine di prova "img.png" usata nel programma ExtractMetadataTest

3.2 Esito del confronto

Questa analisi mi ha permesso di concludere che Apache Tika sia un tool più ricco di Maui, con più funzionalità, più supporto alle estensioni dei file e soprattutto con ottime API Java. L'unico aspetto negativo di Tika, rispetto a Maui, è che non esegue l'estrazione di keywords a partire da un testo, bensì estrae i metadati. Nell'ambito del text mining, l'estrazione di keywords può risultare più interessante rispetto all'estrazione di metadati del documento, tuttavia non sono riuscito a inglobare le funzionalità di Maui in un progetto Java dato che non vengono messe a disposizioni API (o meglio, l'unica libreria-maui che ho provato pare non funzionare).

Obiettivo di questo lavoro era capire quale tool è più efficace nell'ambito del text mining e in relazione a MoK, e integrare tale tool in un modulo software con apposite API in modo da rendere il lavoro modulare e facilmente integrabile con quello di altri gruppi o di lavori futuri. Il modulo software è

ExtractMetadataTest.java

```java
package test;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import org.apache.tika.exception.TikaException;
import org.apache.tika.metadata.Metadata;
import org.apache.tika.parser.AutoDetectParser;
import org.apache.tika.parser.ParseContext;
import org.apache.tika.parser.Parser;
import org.apache.tika.sax.BodyContentHandler;
import org.xml.sax.SAXException;

public class ExtractMetadataTest {

   public static void main(final String[] args) throws IOException, TikaException {

      File file = new File("img.png");
      Parser parser = new AutoDetectParser();
      BodyContentHandler handler = new BodyContentHandler();
      Metadata metadata = new Metadata();
      FileInputStream inputstream = new FileInputStream(file);
      ParseContext context = new ParseContext();
```

Problems Javadoc Declaration Search Console

<terminated> ExtractMetadataTest [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home

```
Transparency Alpha: nonpremultipled
tiff:ImageLength: 1094
Compression CompressionTypeName: deflate
Data BitsPerSample: 8 8 8 8
Data PlanarConfiguration: PixelInterleaved
Dimension VerticalPixelSize: 0.17639795
IHDR: width=1760, height=1094, bitDepth=8, colorType=RGBAlpha, compressionMethod=deflate, f
iCCP: profileName=ICC Profile, compressionMethod=deflate
Chroma ColorSpaceType: RGB
tiff:BitsPerSample: 8 8 8 8
Content-Type: image/png
height: 1094
X-Parsed-By: org.apache.tika.parser.DefaultParser
pHYs: pixelsPerUnitXAxis=5669, pixelsPerUnitYAxis=5669, unitSpecifier=meter
Text TextEntry: keyword=XML:com.adobe.xmp, value=<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmpt
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <rdf:Description rdf:about=""
            xmlns:exif="http://ns.adobe.com/exif/1.0/">
         <exif:PixelXDimension>1760</exif:PixelXDimension>
         <exif:PixelYDimension>1094</exif:PixelYDimension>
      </rdf:Description>
   </rdf:RDF>
</x:xmpmeta>, language=, compression=none
Dimension PixelAspectRatio: 1.0
iTXt iTXtEntry: keyword=XML:com.adobe.xmp, compressionFlag=false, compressionMethod=0, lang
```

Figura 3.2: I metadati estratti dall'immagine

stato implementato avendo in mente la seguente strategia:
L'obiettivo di aggiungere un API "wrapper" è quello di avvicinare il livello di astrazione delle API verso l'utilizzo ipotizzato in MoK. In particolare:

- ho un metodo di libreria che mi permette dato un URL di scaricare il documento PDF
- ho un metodo di libreria che mi permette dato un documento PDF di estrarne testo e metadati in forma String (o StringBuffer, o altro)
- ho un metodo di libreria che mi permette dato uno StringBuffer che rappresenta un testo di estrarne X parole chiave
- ho un metodo di libreria che mi permette date due liste di parole chiave mi dice quante "matchano"

—> costruisco un API di alto livello che mi dice la similarità tra due documenti, dati gli URL a cui si trovano.
Come anticipato a pag. 16, il problema è che Tika, pur essendo effettivamente il migliore per quanto riguarda il numero di funzionalità e la disponibilità di ottime API Java, purtroppo non estrae le keywords di un testo in base a frequenza come fa Maui. Nell'ottica dell'API Wrapper inizialmente ipotizzata si può sfruttare Tika per arrivare fino al punto 3, ma a quel punto servirebbe Maui per estrarre le keywords. Da qui, la necessità di implementare un confronto tra il testo integrale dei documenti, anziché tra le keywords dei testi. In pratica, uso l'estrazione testi di Tika (e non estrazione metadati), quindi eseguo un confronto tra i due testi: una similarità alta si avrà quando i documenti contengono molte parole identiche oppure simili. In questo caso, la similarità tra i due testi può essere calcolata con uno degli algoritmi per la similarità quali la distanza di Levenshtein, la similarità di Cosine e l'algoritmo di Jaccard. A questo scopo, sfrutto una libreria extra (SimMetrics) e la lego al progetto in modo che sia possibile specificare una modalità con la quale si vuole calcolare la similarità tra i due testi.
L'API Wrapper così ottenuta scarica i pdf dagli URL specificati, utilizza Tika

per estrarre i testi dai pdf, quindi calcola la distanza tra i due testi in base all'algoritmo selezionato (specificato da un intero passato come argomento alla chiamata), per fornire un double che indica la similarità tra i due testi. Il codice sorgente allegato a questo lavoro contiene il progetto Java da cui ho generato l'API Wrapper e un altro semplice progetto usato come test della libreria. Nel dettaglio, il primo progetto contiene:

- una classe che può scaricare un pdf dal web, dato in input un URL. I metodi utilizzati sono:

```
1 ...
2 URL url = new URL(fileURL);
3 HttpURLConnection httpConn = (HttpURLConnection) url.
    openConnection();
4 ...
5 InputStream inputStream = httpConn.getInputStream();
6 String saveFilePath = saveDir + File.separator + fileName
    ;
7 ...
8     }
9 }
```

- una classe che sfrutta le API di Tika per estrarre il testo daa un pdf e lo restituisce come String:

```
1 public static String extractTextFromPdf(InputStream file)
    {
2         BodyContentHandler handler = new
              BodyContentHandler(Integer.MAX_VALUE);
3     Metadata metadata = new Metadata();
4     try {
5                 new PDFParser().parse(file, handler,
                      metadata, new ParseContext());
6         } catch (IOException e) {
7                 ...
8 }
```

- una classe che calcola la similarità tra due String e restituisce un double tra 0 e 1 (algoritmo Distanza di Levenshtein).
- una classe che, dati due URL, chiama i metodi delle altre tre classi e li esegue in successione per arrivare alla similarità tra i due testi dei pdf scaricati (figura 3.3).

Nel primo test che ho effettuato, ho specificato due pdf simili (versione compatta e versione estesa di una mia presentazione per il corso di Applicazioni e Servizi Web[3]). La libreria restituisce un valore di similarità pari a 0.415 nel caso di distanza di Levenshtein, 0.75 in caso di Cosine Similarity e 0.36 in caso di algoritmo di Jaccard. Nel secondo test, ho confrontano due pdf relativi ad argomenti diversi, ma basati sul medesimo template (Beamer di Latex per le presentazioni[4]): benché i contenuti siano differenti, molte parole sono in comune, quali il mio nome completo ripetuto ad ogni diapositiva, il mio indirizzo email, il formato del numero di pagine, ecc. Il set di similarità in questo caso consiste in 0.3 (Levenshtein), 0.46 (Cosine), 0.1 (Jaccard), chiaramente un set di valori inferiori a quelli calcolati nel primo test, ma non tanto quanto vorremmo, dato che i contenuti dei pdf sono effettivamente diversi in questo caso. L'utilizzo di keyword extraction che offrirebbe Maui sarebbe risultato molto utile in questo scenario. Infine, nel terzo test ho confrontato uno dei due pdf primari con un pdf selezionato casualmente dal web[5], ottenendo un set di valori di similarità pari a 0.2 (Levenshtein), 0.0 (Cosine), 0.0 (Jaccard), media 0.06, del tutto rassicurante.

```
1 package apiWrapper;
2
3 import java.io.File;
4 import java.io.FileInputStream;
5 import java.io.IOException;
6 import org.apache.commons.io.FilenameUtils;
```

[3]http://www.bran.altervista.org/asw1.pdf, http://www.bran.altervista.org/asw2.pdf
[4]http://www.bran.altervista.org/asw1.pdf, http://www.bran.altervista.org/WS_cesena.pdf
[5]http://www.bran.altervista.org/asw1.pdf, http://www.governo.it/sites/governo.it/files/slide-2annidigoverno.pdf

```
7  import org.apache.tika.exception.TikaException;
8  import org.simmetrics.StringMetric;
9  import org.simmetrics.metrics.StringMetrics;
10
11 import calculateSimilarity.SimilarityChecker;
12 import downloadPdfFromUrl.HttpDownloadUtility;
13 import extractTextFromPdf.PdfTextExtractor;
14 import static org.simmetrics.builders.StringMetricBuilder.
       with;
15
16 public class ApiWrapper {
17
18         private static String text1, text2;
19         private String fileURL;
20         private String fileURL2;
21         private String saveDir;
22         private int mode;
23
24         public ApiWrapper(String fileURL, String fileURL2,
               String saveDir, int mode) {
25                 this.fileURL = fileURL;
26                 this.fileURL2 = fileURL2;
27                 this.saveDir = saveDir;
28                 this.mode = mode;
29         }
30
31         public static double
               calculateSimilarityBetweenURLPdfs(String
               fileURL, String fileURL2, String saveDir, int
               mode){
32                 /* step 1: download pdfs from urls */
33                 try {
34                  HttpDownloadUtility.downloadFile(fileURL,
                        saveDir);
35                  HttpDownloadUtility.downloadFile(fileURL2,
                        saveDir);
36         } catch (IOException ex) {
37                  ex.printStackTrace();
```

```
38          }
39
40                  /* step 2: extract texts from pdfs */
41                  try {
42                          String filename1 = FilenameUtils.
                               getName(fileURL);
43                          String filename2 = FilenameUtils.
                               getName(fileURL2);
44              text1 = PdfTextExtractor.extractTextFromPdf(new
                    FileInputStream(saveDir+filename1));
45              text2 = PdfTextExtractor.extractTextFromPdf(new
                    FileInputStream(saveDir+filename2));
46          } catch (IOException ex) {
47              ex.printStackTrace();
48          }
49
50                  /* step 3: calculate similarity between
                       extracted texts, based on the algorithm
                        specified by the int mode */
51                  double doublesimilarityresult = 0;
52                  double doublejaccardresult = 0;
53                  double averageresult = 0;
54                  StringMetric similaritymetric =
                       StringMetrics.cosineSimilarity();
55                  StringMetric jaccardmetric = StringMetrics.
                       jaccard();
56                  float similarityresult;
57                  float jaccardresult;
58                  /* mode 1 use Levenshtein distance */
59                  if (mode == 1){
60                          SimilarityChecker.printSimilarity(
                               text1, text2);
61                          return SimilarityChecker.similarity
                               (text1, text2);
62                  } else if (mode == 2){
63                          /* mode 2 use Cosine Similarity */
64                          similarityresult = similaritymetric
                               .compare(text1, text2);
```

```
65                          doublesimilarityresult =
                                similarityresult;
66                          System.out.println("Distance
                                between texts using Cosine
                                Similarity is "+
                                doublesimilarityresult);
67                          return doublesimilarityresult;
68                  } else if (mode == 3) {
69                          /* mode 3 use Jaccard algorithm */
70                          jaccardresult = jaccardmetric.
                                compare(text1, text2);
71                          doublejaccardresult = jaccardresult
                                ;
72                          System.out.println("Distance
                                between texts using Jaccard
                                Algorithm is "+
                                doublejaccardresult);
73                          return doublejaccardresult;
74                  } else {
75                          /* every other int counts as
                                average of the main 3 methods
                                */
76                          similarityresult = similaritymetric
                                .compare(text1, text2);
77                          doublesimilarityresult =
                                similarityresult;
78                          jaccardresult = jaccardmetric.
                                compare(text1, text2);
79                          doublejaccardresult = jaccardresult
                                ;
80                          averageresult = (SimilarityChecker.
                                similarity(text1, text2) +
                                doublesimilarityresult +
                                doublejaccardresult) / 3;
81                          System.out.println("Distance
                                between texts based on average
                                of Cosine, Jaccard and
                                Levenshtein algorithms is "+
```

```java
package test;

import apiWrapper.ApiWrapper;

public class main {

    public static void main(String[] args) {
        String fileURL = "http://www.bran.altervista.org/asw1.pdf";
        String fileURL2 = "http://www.bran.altervista.org/asw2.pdf";
        String saveDir = ""+javax.swing.filechooser.FileSystemView.getFileSystemView().getHomeDirectory()+"/";
        ApiWrapper.calculateSimilarityBetweenURLPdfs(fileURL, fileURL2, saveDir);
    }

}
```

Figura 3.3: Classe di prova per l'api wrapper finale

```
                                averageresult);
82                            return averageresult;
83                    }
84            }
85    }
```

3.3 Performance del sistema

Per concludere, ho calcolato il tempo necessario all'esecuzione delle principali funzioni del sistema, sfruttando `System.currentTimeMillis()` prima e dopo l'esecuzione dei metodi da testare. I risultati sono illustrati in figura 3.4. L'estrazione del testo dal pdf è l'operazione più costosa in termini di tempo, mentre il calcolo di similarità impiega una quantità di tempo inferiore di un ordine di grandezza da 10 a 100, a seconda del tipo di algoritmo selezionato. Tra gli algoritmi, Jaccard è il più rapido; segue il Cosine, e infine chiude Levenshtein.

3.4 Nota su API Java di Maui

Sarebbe stato interessante sfruttare la keyword extraction di MAUI tramite API Java, in modo da ottenere un confronto tra i contenuti semantici veri e propri dei pdf, e non tra i testi completi. Purtroppo, a differenza dell'eccellente supporto Java fornito da Apache Tika, Maui risulta

```
fileName = asw1.pdf
fileName = slide-2annidigoverno.pdf
Time needed for text extraction from pdf is 634 millisec
Time needed for text extraction from pdf is 220 millisec
Time needed for similarity check between text using Levenshtein distance is 65 millisec
0,201 similarity between extracted texts
Time needed for similarity check between text using Levenshtein distance is 43 millisec
fileName = asw1.pdf
fileName = slide-2annidigoverno.pdf
Time needed for text extraction from pdf is 40 millisec
Time needed for text extraction from pdf is 141 millisec
Time needed for similarity check between texts using Cosine Similarity is 25 millisec
Distance between texts using Cosine Similarity is 0.0
fileName = asw1.pdf
fileName = slide-2annidigoverno.pdf
Time needed for text extraction from pdf is 36 millisec
Time needed for text extraction from pdf is 85 millisec
Time needed for similarity check between texts using Jaccard Algorithm is 1 millisec
Distance between texts using Jaccard Algorithm is 0.0
fileName = asw1.pdf
fileName = slide-2annidigoverno.pdf
Time needed for text extraction from pdf is 28 millisec
Time needed for text extraction from pdf is 75 millisec
Time needed for similarity check between text using Levenshtein distance is 31 millisec
Time needed for similarity check between texts using average of Levenshtein, Cosine and Jaccard is 33 millisec
Distance between texts based on average of Cosine, Jaccard and Levenshtein algorithms is 0.06715308742061883
```

Figura 3.4: Classe di prova per l'api wrapper finale

assai difficile da integrare nel progetto. Esistono le apposite API, ma l'utilizzo di tali API per la keyword extraction da software Java presuppone un'ottima conoscenza della costruzione di modelli di training e testing in ambito di semantic extraction, e i file di supporto della repository github di Maui (`https://github.com/HIIT/maui-2`), che dovrebbero guidare l'utente nell'utilizzo delle API di Maui, non sono purtroppo sufficientemente esaurienti.

3.5 Analisi Android

Nessuno dei tool analizzati è pronto all'uso nell'ambiente Android. Apache Tika è forse il più adatto ad essere "portato" su Android, ma, a differenza del core del programma, che può essere reso compatibile con Android senza troppi problemi, i parser necessitano di essere parzialmente o interamente re-implementati.

Quello che si può fare per ottenere su Android le stesse funzionalità di estrazione metadati dei tool Tika e Maui è:

1. sfruttare la libreria libexif scritta in C (http://libexif.sourceforge.net),

che però funziona limitatamente alle immagini. Il codice C può essere utilizzato all'interno di un progetto Android utilizzando Android NDK per includere codice nativo all'interno del programma Java-Android. Non una soluzione immediata, dunque.

2. usare il programma Metadata Extractor (drewnoakes), disponibile in Java, che funziona limitatamente ai file multimediali (quindi anche video). In questo caso si può importare direttamente il jar nel progetto Android quindi decisamente meglio rispetto al primo scenario. Sfruttando la repository originale c'è da fare qualche tweak per renderlo compatibile con Android (credo il Manifest e altro), ma c'è già chi ha fatto le modifiche necessarie a renderlo pronto all'uso:
https://github.com/strangecargo/metadata-extractor

3. Per l'estrazione di metadati dai pdf c'è la libreria iTextG pronta per Android: http://www.worldbestlearningcenter.com/tips/Android-pdf-metadata.htm

Riferimenti

[1] Oracle, Data Mining Concepts 11g Release 1 (11.1).
http://docs.oracle.com/cd/B2835901/datamine.111/b28129/

[2] Ben-Dov, M., Feldman, R.: Text Mining and Information Extraction, chapter 38.

[3] Tkach, D., 1998: Text Mining Technology, Turning Information Into Knowledge, IBM.

[4] Wong, W., Fu, A.W., 2000: Incremental Document Clustering for Web Page Classification.

[5] ADIOS, Automatic Distillation of Structure.
http://adios.tau.ac.il

[6] Encyclopædia Britannica, Data mining.
http://www.britannica.com/EBchecked/topic/1056150/data-mining

[7] Ounis, I., Amati, G., Plachouras, V., He, B., Macdonald, C., Lioma, C.: Terrier: A High Performance and Scalable Information Retrieval Platform.

[8] Apache Marmotta.
http://marmotta.apache.org

[9] Witten, I.H., Paynter, G.W., Frank, E., Gutwin, C., Nevill-Manning, C.G.: KEA: Practical Automatic Keyphrase Extraction.

[10] Apache OpenNLP.
https://opennlp.apache.org

[11] Apache Tika, API Usage.
https://tika.apache.org/1.8/examples.html