Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

ESTRAZIONE AUTOMATICA DI INFORMAZIONI DA ARTICOLI SCIENTIFICI IN FORMATO PDF E PUBBLICAZIONE IN LINKED OPEN DATA

Tesi di Laurea in Informatica

Relatore:
Dottor
Angelo Di Iorio

Presentata da:
Fulvio Marcelli

♯ 1
2015-2016

Introduzione

L' obbiettivo di questa tesi è presentare uno strumento per l'analisi di articoli scientifici in formato PDF, chiamato *Investiga*. Questo strumento permette di estratte informazioni da articoli scientifici in formato PDF e di pubblicarle secondo i principi e formati di Linked Open Data. In particolare *Investiga* lavora sulla struttura dei contenuti e sulle didascalie delle figure e delle tabelle.

Introdurremo la serie di sfide SemPub, una challenge nata nel 2014 e riproposta nel 2015 e nel 2016, il cui principale obbiettivo è quello di migliorare l'estrazione automatica di dati da articoli scientifici in formato PDF per renderli leggibili dalle macchine e pubblicarli secondo i principi e i formati Linked Open Data.

Ne descriveremo il metodo di valutazione, comune per tutte le sfide SemPub e utilizzato per valutare l'esecuzione dell'applicazione creata per la tesi, entreremo più nel dettaglio del Task 2 della SemPub 2016, sul quale si è basato il progetto della tesi, e descriveremo brevemente le soluzioni proposte dai partecipanti alla SemPub 2015, delle quali abbiamo già gli articoli che ne descrivono il funzionamento.

Verranno descritti alcuni dei principali strumenti per l'estrazione automatica di dati da articoli PDF e verranno mostrati i principali limiti per ognuno di essi, sia ristretti ai soli articoli scientifici che non.

Spiegheremo il funzionamento di *Investiga*, l'applicazione creata per la tesi, che ha lo scopo di estrarre automaticamente informazioni da articoli scientifici in formato PDF con layout ACM e LNCS e creare un dataset RDF di tali informazioni estratte. Ne verrà descritta l'implementazione per ogni modulo che la costituisce e verranno indicati i principali problemi conosciuti. Descriveremo i risultati ottenuti da *Investiga* durante la valutazione attraverso lo script PHP messo a disposizione dei partecipanti dalla SemPub

e ne verranno discussi i limiti principali.

Vediamo più nel dettagli di cosa parleremo:

- Nel primo capitolo introdurremo il problema di rendere disponibile in un linguaggio utilizzabile dalle macchina le informazioni contenute negli articoli PDF, e quindi l'estrazione automatica di informazioni da articoli in formato PDF, introdurremo la serie di sfide SemPub e principalmente il Task 2 della SemPub 2016 che tratta questa problematica limitata agli articoli scientifici in formato PDF, ne descriveremo le regole e le modalità di valutazione e parleremo delle soluzioni proposte dai partecipanti alla SemPub 2015.

- Nel secondo capitolo introdurremo alcuni dei principali strumenti per l'estrazione di dati da documenti PDF esistenti, PDFMiner, Tabula e PDFBox, ne descriveremo il funzionamento, i metodi d'utilizzo e i principali limiti. Descriveremo poi le applicazioni proposte alla SemPub 2015 che trattano le stesse problematiche ma limitate agli articoli scientifici in formato PDF.

- L'obbiettivo del terzo capitolo è quello di introdurre *Investiga*, l'applicazione creata per la tesi, che ha come scopo quello di estrarre i capitoli di primo livello, le didascalie delle figure e le didascalie delle tabelle in modo automatico da articoli scientifici in formato PDF, principalmente con layout ACM e LNCS, e di creare un dataset RDF delle informazioni estratte. Ne spiegheremo il funzionamento, l'utilizzo di PDFMiner al suo interno, i possibili output e descriveremo le situazioni conosciute nelle quali *Investiga* non lavora correttamente.

- Nel quarto capitolo entreremo più nel dettaglio del progetto *Investiga*, spiegando l'implementazione e le principali funzioni di ciascun modulo che lo compongono.

- Nel quinto capitolo parleremo dei risultati ottenuti dall'applicazione *Investiga*, valutata tramite lo script PHP messo a disposizione dei partecipanti dalla SemPub, su vari insiemi di articoli PDF, il primo composto da soli documenti PDF con layout ACM e LNCS, il training dataset messo a disposizione dei partecipanti alla SemPub 2016 e infine il dataset finale, l'insieme sul quale tutti i partecipanti alla SemPub 2016 hanno dovuto valutare il proprio lavoro.

- Il sesto e ultimo capitolo tratterà la discussione delle problematiche descritte in precedenza e le conclusioni sul progetto *Investiga*, dei suoi risultati, dei suoi problemi e dei margini di miglioramento.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Estrazione automatica di dati da articoli PDF

Il formato principalmente usato per la pubblicazione di articoli scientifici è il formato PDF. Uno dei principali problemi di questo formato è che è pensato per essere leggibile dagli umani ma non dalle macchine, e ne complica l'utilizzo nel Semantic Web. Con il termine Semantic Web, ideato da Tim Berners-Lee, si intende la trasformazione del World Wide Web in un ambiente in cui i documenti pubblicati (pagine HTML. file, immagini, ecc.) sono associati ad informazioni e dati (metadati) che ne specificano il contesto semantico in un formato adatto all'interrogazione, all'interpretazione e all'elaborazione automatica, consentendo ricerche molto più evolute.

Per riuscire ad utilizzare tutte le informazioni contenute negli articoli scientifici in formato PDF nel Semantic Web è necessario estrarre queste informazioni traducendole in un linguaggio utilizzabile dalle macchine, questo è molto importante per riuscire ad utilizzare questi dati nella Semantic Publishing (pubblicazione di informazioni sul web come documenti accompagnati da markup semantico, fornendo alle macchine un modo per comprendere la struttura e il significato delle informazioni pubblicate), per migliorare l'editoria scientifica e per poterli utilizzarli nelle tecnologie Linked Open Data (progetto del W3C, ha l'obbiettivo di estendere il Web pubblicando diversi dataset open come RDF sul Web e collegare attraverso link RDF i dati provenienti da diverse risorse).

Questo, però, non è un compito affatto facile, soprattutto perché la struttura di un

documento PDF è molto diversa da altri tipi di file tipo world, html, ecc. e spesso non ha una struttura logica, come, ad esempio, frasi o paragrafi. Il formato PDF (Portable Document Format) è un formato di file basato su un linguaggio di descrizione di pagine, sviluppato da Adobe System nel 1993, che serve a rappresentare documenti indipendentemente dall'hardware e software utilizzati per generarli e visualizzarli.

Quindi un documento PDF è composto da un insieme di istruzioni dedicate alla visualizzazione degli elementi sullo schermo o all'interno di un foglio di carta, specificandone anche l'esatta posizione.

A causa della struttura così complessa del formato PDF estrarre informazioni da articoli PDF correttamente e senza perdita di informazioni non è affatto facile.

Esistono diverse sfide proposte in tutto il mondo che hanno come scopo principale quello di migliorare l'estrazione automatica di informazioni da articoli scientifici in formato PDF e renderle leggibili dalle macchine, tra le principali "*PDF Liberation Hackathon*" e la serie di challenge "*SemPub*", iniziata nel 2014 e proposta anche nel 2015 e nel 2016.

Introduciamo di seguito la challenge "*PDFLiberation Hackathon*" e la challenge "*SemPub*", che è divisa in tre Task, uno dei quali, il Task 2, è rivolto specificatamente all'estrazione automatica di informazioni da articoli scientifici in formato PDF e creazione di un dataset RDF sia nell'edizione del 2015 sia in quella del 2016. Spiegheremo il metodo di valutazione della "*SemPub 2015*", che è rimasto uguale anche per la "*SemPub 2016*", mostreremo le soluzioni delle candidature ricevute per il Task 2 della "*SemPub 2015*" e le loro valutazioni ed elencheremo le informazioni da estrarre richieste nel Task 2 della "*SemPub 2016*".

Il progetto proposto per la tesi è basato sull'estrazione automatica di alcune delle informazioni richieste nel Task 2 della "*SemPub 2016*" da articoli scientifici in formato PDF, cioè i capitoli di primo livello, le didascalie delle figure e le didascalie delle tabelle, utili per poter creare in automatico l'indice, la lista delle figure e la lista delle tabelle di un articolo PDF.

1.1 PDF Liberation Hackathon

"*PDF Liberation Hackathon*" è una sfida che si svolge in diverse città americane alcuni giorni all'anno. I partecipanti possono lavorare su una sfida di estrazione da PDF proposta dagli sponsor organizzatori o possono sviluppare miglioramenti per gli strumenti per l'estrazione di PDF open source esistenti [1]. Nell'edizione del 2014 ci sono state diverse sfide che trattano l'estrazione da documenti PDF. Vediamo alcune:

- estrarre un resoconto delle entrate e delle spese degli ultimi dieci anni dalle relazioni finanziarie annuali pubblicate dalle principali città degli Stati Uniti.
- estrarre le informazioni dalle relazioni in PDF delle finanze personali rilasciate annualmente da alcuni membri della Camera di rappresentanza e rappresentarle in un formato tabulare.
- estrarre le informazioni riguardanti le torture praticate nel corso del tempo pubblicate negli *Amnesty International Annual Report*. Questi dati sono stati utilizzati per una campagna mondiale contro la tortura.
- tradurre i rapporti qualitativi in formato PDF emessi dall'USAID (agenzia statunitense per lo sviluppo internazionale, è il dipartimento del governo responsabile per l'assistenza estera negli 80 paesi a basso reddito di tutto il mondo) in linguaggi comprensibili dalle macchine per poter identificare ciò che ha funzionato e ciò che non ha funzionato nello sviluppo internazionale.

Per la valutazione alcuni fornitori di soluzioni PDF commerciali hanno deciso di offrire delle versioni di valutazione del loro software ai partecipanti della sfida *Hackathon*.

1.2 SemPub

Descriviamo brevemente la serie di sfide SemPub: SemPub sta per *Semantic Publishing Challenge - Assessing the Quality of Scentific Output by Information of Extraction and Interlinking* , e viene proposta da Angelo Di Iorio dell'Università di Bologna, Cristoph Lange e Sahar Vahdati dell'Università di Bonn (Germania), Anastasia Dimou dell'Università di Ghent (Belgio).

La serie di challenge *Semantic Publishing* ha lo scopo di investigare i tipi di nuovi approcci proposti per poter migliorare l'editoria attraverso l'utilizzo della tecnologia Linked Open Data, e in particolare lo scopo di migliorare l'estrazione di informazioni da articoli scientifici in formato PDF [2].

Siccome l'ipotesi che la comunicazione digitale attraverso i mezzi semantici possa migliorare notevolmente l'editoria digitale ricca di media rispetto a stampe e documenti PDF sta diventando realtà, riuscire a convertire il formato PDF in un formato leggibile dalle macchine sta diventando sempre più importante.

1.2.1 Definizione SemPub 2015

La SemPub proposta nel 2015 è divisa in tre Task:

Task 1: rispondere alle domande relative alla qualità dei laboratori attraverso il calcolo di metriche sui dati, calcolate dalle procedure create dai partecipanti, considerando anche informazioni su persone ed eventi.

Task 2: focalizzato sull'estrazione di informazioni contestuali dal testo completo degli articoli: citazioni, affiliazioni degli autori, agenzie di finanziamento, capitoli, didascalie delle figure e delle tabelle, eccetera.

Task 3: interconnettere i linked data CEUR-WS.org con altri insiemi rilevanti di linked data.

1.2.2 Procedura di valutazione

La valutazione segue una procedura comune per tutte le attività, simile alle altre challenge di Semantic Web, ed è rimasta la solita anche per la valutazione della *SemPub 2016*, e quindi è la modalità con cui ho valutato i risultati dell'esecuzione dell'applicazione creata per la tesi:

1. Per ogni attività è stato inizialmente pubblicato un *trainig dataset*, utilizzato dai partecipanti per testare e formare i loro strumenti per l'estrazione ed interconnessione dei dati.
2. Per la Task 2, è stata specificata una struttura di base del dataset RDF che deve essere creato con i dati estratti dal traning dataset, senza prescrivere nessun vocabolario.
3. È stata fornita la query da eseguire sul dataset RDF in linguaggio naturale e i risultati attesi per ogni articolo PDF.
4. Pochi giorni prima del termine di presentazione è stato pubblicato l'*evaluation dataset*, cioè un sovrainsieme del training dataset, che doveva essere usato dai partecipanti alla SemPub come input per la valutazione finale.
5. È stato chiesto ai partecipanti di inviare i risultati del linked data derivante dalla estrazione ed interconnessione (sotto una licenza open per consentirne il riutilizzo).
6. Sono stati assegnati premi per la miglior performance e per l'approccio più innovativo.
7. Durante tutto il periodo della SemPub è stata mantenuta la trasparenza con i partecipanti invitandoli a porre domande, alle quali è stato risposto pubblicamente.

Dopo la valutazione sono stati messi a disposizione dei partecipanti i gold standard e i punteggi. La valutazione è stata automatizzata grazie ad una raccolta di script scritti in PHP la cui funzione è quella di confrontare i file CSV risultanti dalle query SPARQL

al dataset RDF con i file CSV dei gold standard dei risultati attesi e compilare un report con i risultati ottenuti. Il punteggio finale è dato dalla variabile **F-Score** (chiamata anche F1-Score, misura di precisione di un test, va da 0, peggior valore, a 1, miglior valore), che è data dalla proporzione di altre 2 misure (F-Score = 2 * ((Precision * Recall)/(Precision + Recall))):

- **Precision**: misura di esattezza, cioè la proporzione tra informazioni pertinenti e quelle recuperate:

  numero di informazioni attinenti recuperate diviso il numero totale delle informazioni recuperate durante la ricerca.

- **Recall**: misura di completezza, cioè la proporzione fra il numero di informazioni attinenti rilevate e il numero di informazioni rilevanti disponibili:

  numero di informazioni attinenti recuperate da una ricerca diviso il numero totale di informazioni attinenti esistenti.

Nella figura sottostante viene mostrato un report di valutazione creato tramite l'utilizzo degli script PHP.

[TOP] [>> Q4.41] SEMPUB CHALLENGE - LOOSE EVALUATION

Q4.40

Identify the first-level sections of the paper http://ceur-ws.org/Vol-1315/paper9.pdf

Precision	0.857
Recall	0.75
F-score	0.8
Matches (TP)	6
Missed (FN)	2
Incorrect (FP)	1

Gold Standard

section-iri	section-number	section-title	
<http://ceur-ws.org/section/vol-1315-paper9_sec1>	1^^xsd:integer	Introduction	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec2>	2^^xsd:integer	Image schema	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec3>	3^^xsd:integer	Conceptual blending	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec4>	4^^xsd:integer	Formalising conceptual blending	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec5>	5^^xsd:integer	Blending with image schemas	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec6>	6^^xsd:integer	Outlook	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec7>	7^^xsd:integer	Acknowledgments.	MISSED
<http://ceur-ws.org/section/vol-1315-paper9_sec8>	8^^xsd:integer	References	MISSED

Under Evaluation

section-iri	section-number	section-title	
<http://ceur-ws.org/section/vol-1315-paper9_sec1>	1^^xsd:integer	Introduction	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec2>	2^^xsd:integer	Image schema	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec3>	3^^xsd:integer	Conceptual blending	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec4>	4^^xsd:integer	Formalising conceptual blending	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec5>	5^^xsd:integer	Blending with image schemas	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec6>	6^^xsd:integer	Outlook	MATCH
<http://ceur-ws.org/section/vol-1315-paper9_sec7>	7^^xsd:integer	References	FP

Generated on: Wed, 01 Jun 2016 12:41:42

Figura 1.1: In figura viene mostrato un esempio di valutazione da parte degli script PHP.

1.2.3 Task 2: Estrazione di informazione contestuali dal testo completo di articoli scientifici in formato PDF

Il progetto della mia tesi è basato sulla Task 2 della *SemPub 2016*, e più precisamente sull'estrazione dei capitoli di primo livello, didascalie delle figure e delle tabelle, e creazione di dataset RDF delle informazioni così ottenute.

Introduciamo quindi più specificatamente la Task 2 della SemPub 2016.

Motivazioni e Obbiettivi

Il Task 2 è stato progettato per testare la capacità dei partecipanti di estrarre informazioni dall'intero testo degli articoli. Ai partecipanti è stato chiesto di estrarre dati sulle citazioni, per consentire una valutazione precisa del collegamento, condivisione e valutazione della ricerca attraverso citazioni e altri dati e di renderli disponibili come Linked Open Data. Il formato degli articoli scelto è il formato PDF, questo perché il formato PDF è ancora predominante per la pubblicazione di articoli scientifici. Una delle principali difficoltà è dovuta al fatto che la struttura interna di un PDF non corrisponde ad una struttura logica del contenuto, ma ad una sequenza di comandi per la formattazione e il layout. Per questo la sfida per i partecipanti è stata quella di recuperare la struttura logica per poter in seguito estrarre le informazioni contestuali e rappresentarle come affermazioni semantiche.

Articoli di input

La costruzione del dataset di input è stata fatta in modo da coprire un'ampia varietà di casi. Il dataset comprendeva una serie di articoli di ricerca codificati in XML, presi dagli archivi PubMedCentral e Pensoft Open Access, con una struttura interna, stile e numeri eterogenei. Sia il set di dati che le query richieste sono state completamente disgiunte da Task 1.

Domande del Task 2 della SemPub 2016

Il task 2 comprende 8 diverse domande:

- **Q2.1**: identificare le affiliazioni degli autori di un certo articolo X.
- **Q2.2**: identificare i paesi delle affiliazioni degli autori di un certo articolo X.
- **Q2.3**: identificare il materiale supplementare di un articolo X.
- **Q2.4**: identificare i titoli delle sezioni di primo livello di un articolo X.
- **Q2.5**: identificare le didascalie delle tabelle dell'articolo X.
- **Q2.6**: identificare le didascalie delle figure dell'articolo X.
- **Q2.7**: identificare le agenzie di finanziamento che hanno sostenuto la ricerca presentata dall'articolo X (o parte di essa).
- **Q2.8**: identificare i progetti europei che hanno sostenuto la ricerca presentata dal documento X.

Il progetto per la tesi è basato sulle query **Q2.4**, **Q2.5** e **Q2.6** e la creazione di un dataset RDF delle informazioni estratte permettendone l'utilizzo nella tecnologia Linked Open Data.

Applicazioni partecipanti alla SemPub 2015

Per il Task 2 proposto nella SemPub 2015 sono state ricevute 6 candidature:

1. Sateli/Witte [3]: proposto un approccio basato su regole. Hanno proposto due componenti logici in una procedura:

   - un processore sintattico per identificare il layout di base.
   - un processore semantico per identificare le entità nel testo tramite modelli di ricerca.

   Inoltre questo tool include un mappatore RDF che trasforma i dati estratti in triple RDF. Questo progetto ha vinto il premio per l'approccio più innovativo.

2. Traczky/Bolikowski [4]: proposta un'applicazione Java per l'estrazione di metadati da pubblicazioni scientifiche, appositamente modificata. É stato anche implementato un esportatore RDF. Questo tool ha vinto il premio miglior performance.

3. Klampf/Kern [5]: proposto un framework modulare che identifica e raggruppa i gruppi del layout del PDF. Dei classificatori scelti hanno contribuito a rilevare il ruolo di ciascun blocco (dati, autore, affiliazioni, ecc.). É stata costruita dagli autori un'ontologia dei concetti d'informatica utilizzata per la rilevazione automatica delle annotazioni di finanziamenti, sovvenzioni e dati del progetto.

4. Ronzano et al. [6]: hanno esteso la loro partecipazione al Task 1 per estrarre dati da articoli PDF. La loro procedura include l'elaborazione del testo, moduli di riconoscimento delle entità e utilizzo di servizi esterni per l'estrazione dagli articoli PDF.

5. Nuzzolese et al. [7]: proposto il sistema *MACJa*, principalmente scritto in Python e Java, che integra molte tecniche e servizi:

   - utilizza PDFMiner per estrarre il contenuto testuale da un articolo PDF e gestire diverse analisi del contenuto.
   - Named Entity Recognition (NER): tecniche utilizzate per identificare gli autori e affiliazioni.

- API CrossRef: interrogate per estrarre i dati dalle citazioni.
- tecniche PNL, pattern matching e allineamento delle risorse lessicali: utilizzate per la rilevazione di ontologie, borse di studio e agenzie di finanziamento.

6. Kovriguina et al. [8]: proposta un'architettura semplice ed efficace implementata in Python. Approccio basato su template, espressioni regolari e alcuni servizi esterni per migliorare la qualità dei risultati. Un modulo esterno estrae il testo dal file PDF, questo testo viene confrontato con un insieme di espressioni regolari per estrarne le informazioni cercate. É presente anche un serializzatore RDF basato su un'ontologia personalizzata.

Mostriamo i risultati delle diverse sottoscrizioni in tabella:

Autori	**Precision**	**Recall**	**F-Score**
Tkaczyk/Bolikowski	0.369	0.417	0.381
Klampf/Kern	0.388	0.285	0.292
Nuzzolese et al.	0.274	0.251	0.257
Sateli/Witte	0.3	0.252	0.247
Kovriguina et al.	0.289	0.3	0.265
Ronzano et al.	0.316	0.401	0.332

Tabella 1.1: La tabella riporta i risultati relativi alla valutazione del Task 2 della SemPub 2015.

Descriveremo più dettagliatamente le applicazioni partecipanti alla SemPub 2015 nel prossimo capitolo.

Capitolo 2

Applicazioni per l'estrazione automatica di informazioni da articoli PDF: stato dell'arte

Lo scopo di questo capitolo è quello di introdurre brevemente alcuni degli strumenti esistenti per l'estrazione automatica di informazioni da file in formato PDF.

In particolare PDFMiner sul quale mi sono appoggiato per la creazione del mio progetto, poi esamineremo Tabula, PDFBox, e le applicazioni partecipanti alla Task 2 della SemPub 2015 per l'estrazione automatica di informazioni da articoli scientifici in formato PDF.

2.1 PDFMiner

PDFMiner è una libreria per l'analisi di documenti PDF scritta in Python (linguaggio di programmazione ad alto livello, orientato agli oggetti) da Yusuke Shiniama [9]. È un progetto in via di sviluppo ed è protetto da licenza MIT/X: l'autorizzazione per l'utilizzo del software è concessa, a titolo gratuito, a chiunque ottenga una copia di questo software e del file di documentazione (il "Software"), inclusi, senza limitazione, i diritti di utilizzare, copiare, modificare, unire, pubblicare, distribuire, concedere in

licenza e/o vendere copie del software, e di consentire alle persone a cui viene fornito il software di farlo. Questa autorizzazione è soggetta alle seguenti condizioni: l'avviso di copyright e questo devono essere inclusi in tutte le copie o parti sostanziali del software.

A differenza di altri strumenti relativi a documenti PDF, PDFMiner si concentra interamente su come ottenere e analizzare i dati di testo ed è totalmente focalizzato sull'analisi dei dati testuali.

Questa libreria permette di ottenere la posizione esatta di un segmento di testo presente in una pagina, o anche di un carattere o una linea, e comprende anche un convertitore utile a trasformare file PDF in altri formati di testo (HTML, XML, TEXT, TAG).

Inoltre ha un parser PDF estendibile che può essere utilizzato per scopi diversi dall'analisi del testo.

2.1.1 Funzionamento di PDFMiner

I documenti in formato PDF sono molto diversi da file in formato world, html, eccetera.

Il contenuto del formato PDF, come già visto, è un insieme di istruzioni che specificano la posizione esatta degli elementi e come visualizzarli sul display o all'interno di un foglio di carta e, nella maggior parte dei casi, non hanno una struttura logica come, ad esempio, frasi e paragrafi [10].

A causa della struttura molto complicata l'analisi completa di un documento in formato PDF richiederebbe una grande quantità di memoria e tempo, tuttavia, per la maggior parte delle attività di analisi di un PDF non è necessaria l'analisi della struttura completa del documento.

PDFMiner si basa sulla ricostruzione di alcune delle informazioni contenute nel formato PDF, adottando una strategia di analisi pigra (parsing lazy), cioè quella di analizzare le istruzioni della struttura del documento PDF solo quando è necessario.

Per la sua analisi, PDFMiner ha bisogno di diverse classi. Le 2 classi principali sono PDFParser, che si occupa di recuperare i dati dal file, e PDFDocument che si occupa di

memorizzare i dati recuperati. Queste 2 classi sono collegate tra loro.

Inoltre ha bisogno delle seguenti classi:

- **PDFPageInterpreter**: si occupa di processare il contenuto della pagina.
- **PDFDevice**: ha lo scopo di tradurre tutte le informazioni di cui abbiamo bisogno.
- **PDFResourceManager**: viene utilizzata per memorizzare le risorse condivise, quali font e immagini.

In figura viene mostrata la relazione tra le diverse classi:

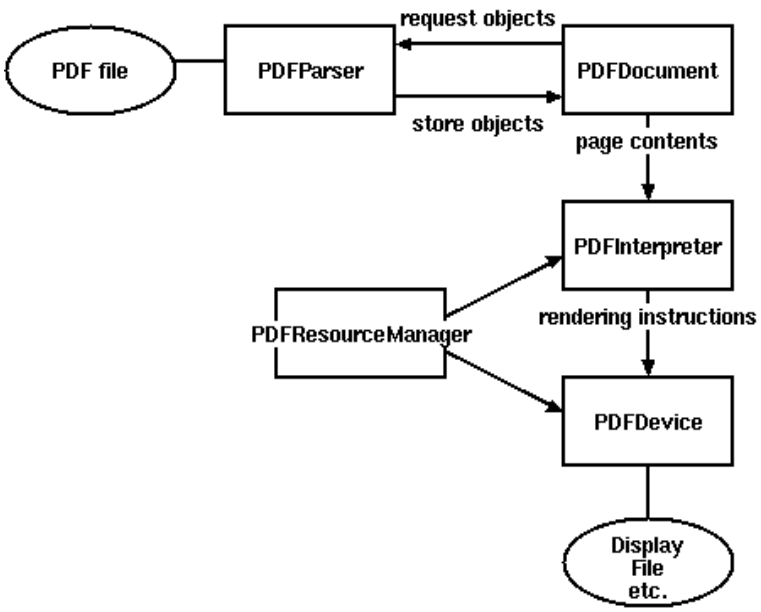


Figura 2.1: In figura viene descritta la relazione tra le diverse classi di PDFMiner.

2.1.2 Utilizzi di PDFMiner

La libreria PDFMiner è utile per diversi scopi:

Analisi di un documento. Il modo tipico di utilizzo di PDFMiner per l'analisi di un documento PDF è il seguente:

```
from pdfminer.pdfparser import PDFParser
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfpage import PDFTextExtractionNotAllowed
from pdfminer.pdfinterp import PDFResourceManager
from pdfminer.pdfinterp import PDFPageInterpreter
from pdfminer.pdfdevice import PDFDevice

# Apre un file PDF.
fp = open('mypdf.pdf', 'rb')

# Crea un oggetto parser PDF associato con il file.
parser = PDFParser(fp)

# Crea un oggetto documento PDF dove memorizzare la struttura del documento.
# Fornire la password per l'inizializzazione.
document = PDFDocument(parser, password)

# Controlla se il documento consente l'estrazione del testo, in caso contrario termina
l'esecuzione.
if not document.is_extractable:
    raise PDFTextExtractionNotAllowed

# Crea un oggetto manager delle risorse PDF per contenere le risorse condivise.
rsrcmgr = PDFResourceManager()

#Crea un oggetto device PDF.
device = PDFDevice(rsrcmgr)

# Crea un oggetto interprete PDF.
interpreter = PDFPageInterpreter(rsrcmgr, device)

# Processa ogni pagina contenuta nel documento.
for page in PDFPage.create_pages(document):
    interpreter.process_page(page)
```

Analisi del layout. Un tipico approccio per analizzare il layout di un documento tramite PDFMiner è il seguente:

```
from pdfminer.layout import LAParams
from pdfminer.converter import PDFPageAggregator

# Setta i parametri per l'analisi.
laparams = LAParams()

# Crea un oggetto aggregatore di pagine PDF.
device = PDFPageAggregator(rsrcmgr, laparams=laparams)
interpreter = PDFPageInterpreter(rsrcmgr, device)
for page in PDFPage.create_pages(document):
    interpreter.process_page(page)

    # Riceve l'oggetto LTPage per la pagina.
    layout = device.get_result()
```

Un oggetto LTPage è una struttura dati rappresentante un'intera pagina, contenente diversi oggetti figli, tra i quali, per esempio, LTTextBox, LTFigure, LTImage, LTRcect, LTCurve e LTLine, che a loro volta possono contenere diversi oggetti figli. La struttura di un oggetto LTPage è quindi una struttura ad albero, come mostrato nella seguente figura:

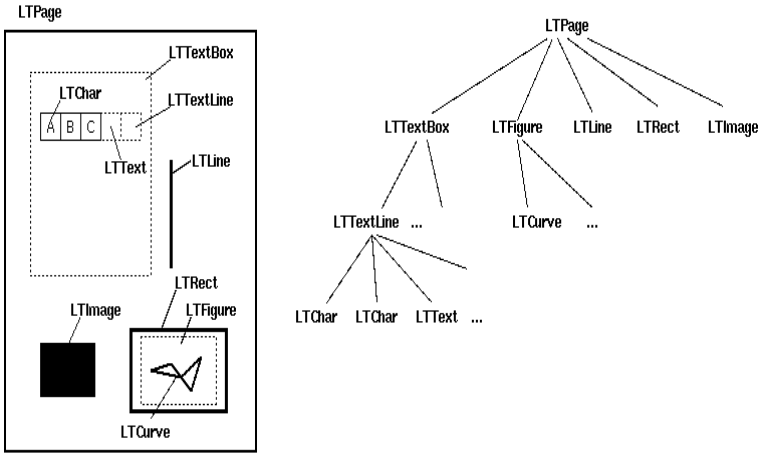


Figura 2.2: In figura viene descritta la struttura dati di un elemento LTPage.

Accedere alla tabella dei contenuti del documento. Per poter accedere alla tabella dei contenuti del documento, l'approccio da utilizzare è il seguente:

```
from pdfminer.pdfparser import PDFParser
from pdfminer.pdfdocument import PDFDocument

# Apre un documento PDF.
fp = open('mypdf.pdf', 'rb')
parser = PDFParser(fp)
document = PDFDocument(parser, password)

# Prende i contenuti del documento.
outlines = document.get_outlines()
for (level,title,dest,a,se) in outlines:
    print (level, title)
```

2.1.3 Interfaccia da riga di comando

PDFMiner fornisce anche un'interfaccia da riga di comando attraverso 2 comandi [11, 12]: *pdf2txt.py* e *dumppdf.py*.

pdf2txt.py

Il pacchetto PDFMiner include l'istruzione da riga di comando **pdf2txt.py**, che può essere usata per estrarre immagini o testo da documenti in formato PDF, o anche per convertire un documento in formato PDF in un file di un altro formato (html, xml, text o tag).

Inoltre, per ogni porzione di testo, estrae anche le posizioni corrispondenti, i nomi dei font, le dimensioni dei caratteri, la direzione della scrittura (orizzontale o verticale).

Per i file PDF protetti è necessario fornire una password e non è possibile estrarre porzioni di testo da documenti che non hanno il permesso di estrazione.

Questo comando è molto flessibile e supporta diverse opzioni, delle quali riportiamo una panoramica delle più comuni:

PDF2TXT.PY [OPZIONI] NOMEFILE.PDF

Opzioni:

- **-o**: nome file di output.
- **-p**: elenco separato da virgole del numero delle pagine estratte.
- **-t**: formato di output (text/html/xml/tag [per PDF Tagged]).
- **-O**: nome directory (innesca l'estrazione delle immagini dal documento PDF, che inserirà nella directory specificata).
- **-P**: password.

Si noti che PDFMiner non è in grado di ricostruire il testo redatto sotto forma di immagini perché ciò richiederebbe il riconoscimento ottico dei caratteri. Inoltre non tutti

i caratteri presenti in un documento PDF possono essere convertiti in modo sicuro in Unicode. Per questo PDFMiner non può estrarre la locazione corrispondente, il nome del font, la dimensione del font, eccetera per ogni bit del testo.

Spesso però è possibile estrarre il testo ed utilizzare dei modelli tipici di python per l'elaborazione del testo ed ottenere i dati in un formato utilizzabile.

dumppdf.py

Il pacchetto PDFMiner include inoltre un'altra istruzione da riga di comando, **dumppdf.py**, usata principalmente per scopi di debugging.

Attraverso questo comando si scarica il contenuto di un file PDF in uno pseudo formato XML, ed è anche possibile estrarre alcuni elementi significativi, come, ad esempio, le immagini.

Riportiamo di seguito una breve panoramica delle opzioni più comuni:

DUMPPDF.PY [OPZIONI] NOMEFILE.PDF

Opzioni:

- **-a**: scarica tutti gli oggetti.
- **-p**: elenco separato da virgole del numero delle pagine estratte.
- **-i**: id dell'oggetto.
- **-E**: nome directory (estrae gli oggetti presenti nel documento PDF nella directory specificata).
- **-T**: scarica la tabella dei contenuti.
- **-P**: password.

Questo comando è molto utile quando si ha a che fare con un documento PDF problematico e si desidera conoscere gli ID degli oggetti contenuti nel PDF per poterli riutilizzare in seguito.

2.1.4 Margini di miglioramento

PDFMiner è un progetto in via di sviluppo e i programmatori si sono prefissati diversi margini di miglioramento [13].

Elenchiamo alcuni dei principali obbiettivi per il futuro:

- l'euristica ha descritto il buon funzionamento dell'analisi sui documenti PDF processati finora, ma si può prevedere un forte calo delle prestazioni per l'analisi di documenti più complessi.

- essere in grado, in futuro, di salvare ogni documento in formato PPM (Portable Pixel Map, è un formato per il salvataggio delle immagini [14]) o in formato predefinito PNM (non è un vero e proprio formato ma è un nome comune per indicare i formati PGM, PPM e PBM che sono formati di tipo grafico [15]).

- risolvere alcuni problemi presenti per titolo, font titolo e spaziatura, dovuti al fatto che titolo e paragrafi non sono distinguibili dal resto del testo.

- risolvere alcune problematiche nella rimozione del numero di pagina, dovute al fatto che ogni documento numera le pagine in maniera leggermente diversa.

- eliminare le difficoltà nella manipolazione delle note, che si riscontrano specialmente quando si estendono su diverse pagine, anche se consecutive.

2.2 Tabula

Tabula è un'Applicazione web nata dalla fusione di due progetti nati separatamente [16]:

- un'applicazione web sulla quale stava lavorando Manuel Aristán, membro della Knight-Mozilla.

- Ferrago: un utility da riga di comando utile per individuare ed estrarre dati dalle tabelle nei documenti in formato PDF, sul quale Jeremy B. Merrill ha scritto un resoconto.

Tabula è un progetto open source in via di sviluppo progettato da giornalisti con lo scopo di essere utilizzato dai giornalisti e da chiunque altro per l'utilizzo di dati "bloccati" su file PDF [17]. È disponibile per Windows, Mac OS X e Linux, per funzionare ha bisogno di una Java Runtime Enviroment compatibile con Java 7 (cioè Java 7 o superiore).

Questa applicazione consente di caricare un file PDF su una semplice interfaccia web ed estrarne i dati tabulari in formato CSV ed è stata progettata pensando alla sicurezza: il documento PDF e i dati estratti non toccano mai la rete, finché sulla barra degli indirizzi del browser viene visualizzato "localhost" o "127.0.0.1" tutta l'elaborazione avviene sulla macchina locale.

Tabula è molto diffuso, soprattutto nell'ambito del giornalismo investigativo, negli organi d'informazione di tutte le dimensioni, tra i quali ProPublica, il Times di Londra, Foreign Policy, La Nación (Argentina), il New York Times e il St.Paul Pioneer Press (Minnesota). Inoltre è molto usato dai ricercatori per trasformare documenti PDF in fogli di calcolo Excel, CSV o file JSON per poterli utilizzare nelle applicazioni di analisi e database.

Essendo open source, Tabula permette a tutti di incorporare delle sue parti nei propri progetti.

2.2.1 Funzionamento di Tabula

L'utilizzo di Tabula è molto semplice [18]: dopo aver scaricato il pacchetto .zip relativo alla propria architettura dal sito ufficiale, è sufficiente andare nella directory appena estratta ed eseguire il programma Tabula presente al suo interno. Si aprirà una pagina del browser all'indirizzo "HTTP://LOCALHOST:8080" (o "127.0.0.1:8080") con l'interfaccia web di Tabula, mostrata in figura:

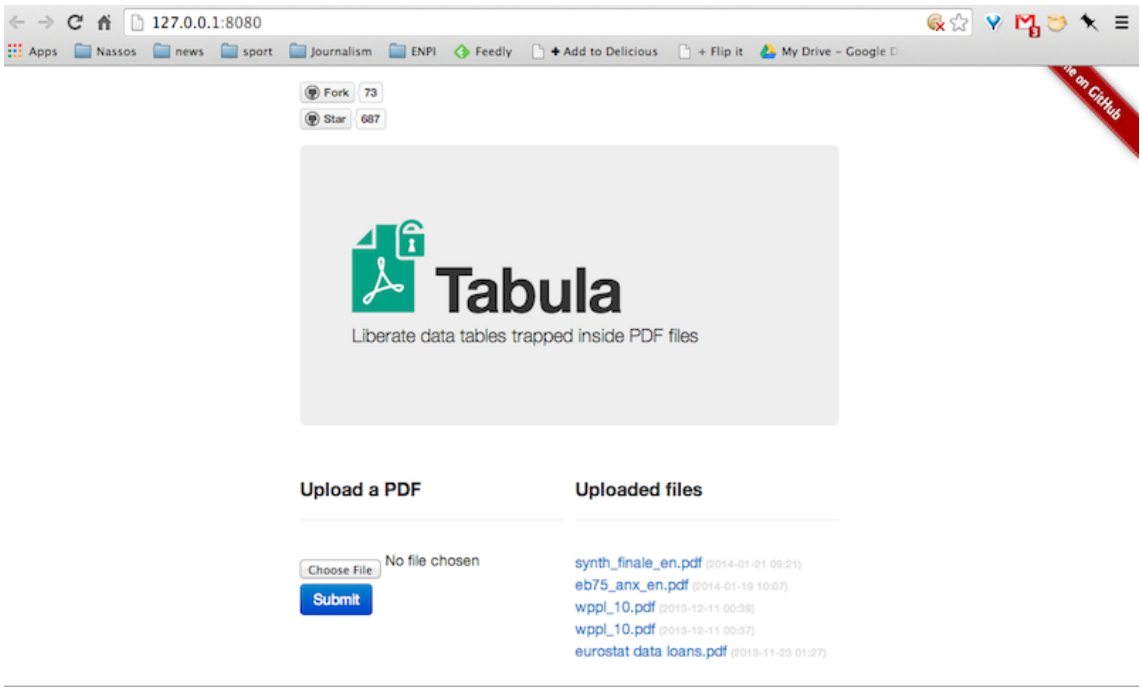


Figura 2.3: In figura viene mostrata l'interfaccia web di Tabula.

A questo punto si può iniziare ad utilizzare Tabula:

la prima cosa da fare è caricare un file PDF contenente una tabella di dati. Fatto questo si può aprire il file appena caricato cliccandoci sopra e passare alla pagina che si desidera per selezionare la tabella o la parte di essa che ci interessa cliccando e trascinando con il mouse per contornare la tabella come mostrato dalla seguente figura:

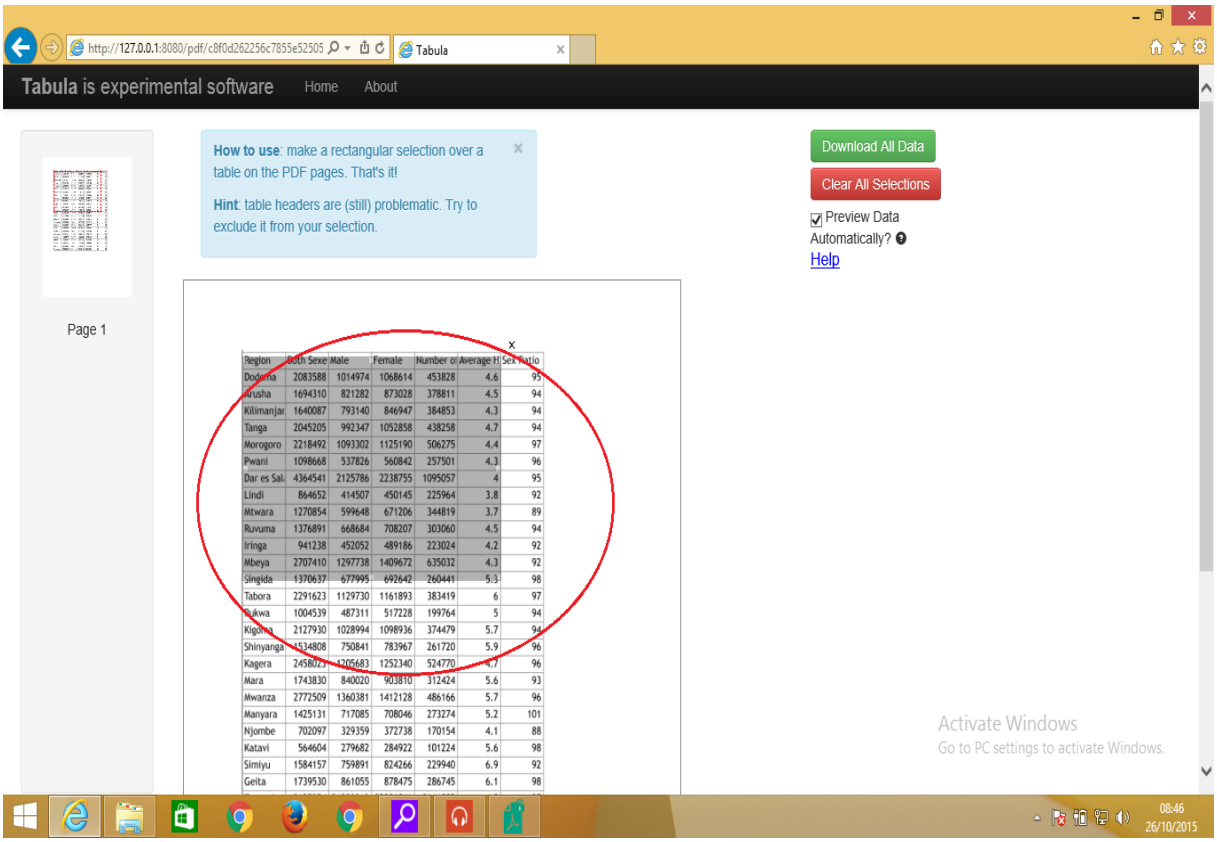


Figura 2.4: In figura viene mostrata la selezione di una tabella del file PDF caricato.

Cliccando sul bottone *"Preview & Export Extracted Data"* Tabula cercherà di estrarre i dati e visualizzarne l'anteprima.

Se i dati sembrano essere corretti, si può procedere cliccando sul bottone *"Esporta"*, che consentirà di lavorare sui dati così estratti come su un file di testo o un foglio di calcolo invece che su un file PDF, altrimenti si può tornare indietro e regolare la selezione. É possibile aprire il file così scaricato con Microsoft Excel o attraverso la libreria LibreOffice Calc. La figura seguente mostra l'anteprima dei dati selezionati dove si ha la possibilità di copiare i dati negli appunti o scaricarli in formato CSV:

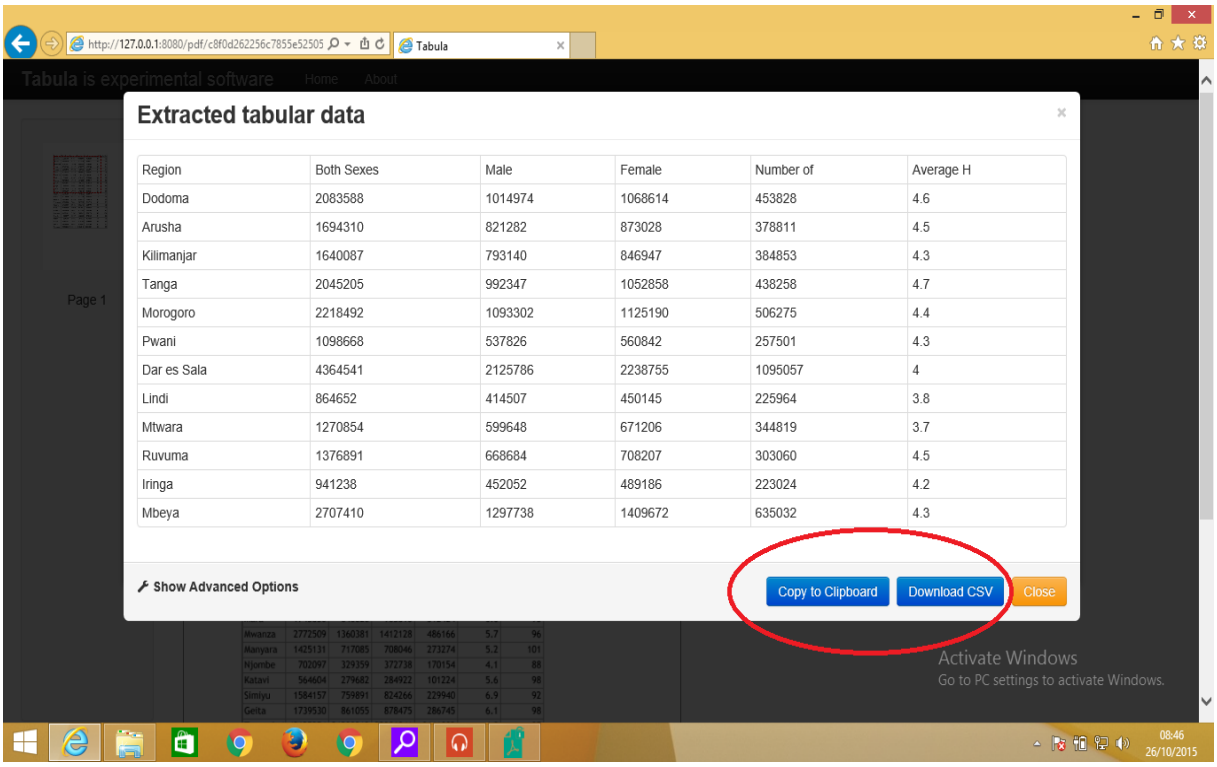


Figura 2.5: In figura viene mostrata l'anteprima dei dati selezionati.

Inoltre Tabula mette a disposizione un tool da linea di comando [19], **tabula.java** (in realtà è il processo principale, ma è possibile anche utilizzarlo autonomamente da linea di comando), utile quando si vogliono processare documenti PDF di grandi dimensioni o diversi file PDF impaginati nella stessa identica maniera.

Per funzionare questo comando ha bisogno delle coordinate che quindi bisogna aver ottenuto in precedenza.

Il comando da lanciare da terminale è il seguente:

```
JAVA -JAR ./TARGET/TABULA-0.9.9-JAR-WIDTH-DEPENDENCIES.JAR -P ALL -A $Y1,&X1,$Y2,$X2 -O $CSVFILE $FILENAME
```

dove:

- **$y1, $x1, $y2, $x2** sono le coordinate ottenute precedentemente.

- **$csvfile** è il nome del file dove verranno inserire le tabelle di output.

- **$filename** è il nome del file PDF di input.

2.2.2 Limitazioni di Tabula

Le principali limitazioni di Tabula sono:

- **Documenti PDF scannerizzati**: Tabula funziona solo su documenti PDF basati sul testo, quindi se il documento è scannerizzato il lavoro di estrazione di informazioni deve essere eseguito manualmente.

  Questo perché la tecnologia libera OCR (Optical Character Recognition, cioè i sistemi di riconoscimento ottico dei caratteri, che sono dei programmi dedicati alla conversione di un'immagine contenente testo, di solito acquisite tramite scanner, in testo digitale modificabile con un normale editor) non è a buon punto con l'automazione dell'esecuzione su molti documenti.

- **Rilevamento automatico delle tabelle**: per il momento si sta ancora lavorando sul rilevamento automatico delle tabelle, fino al raggiungimento di questo obbiettivo si dovrà fare manualmente una selezione rettangolare attorno alle tabelle che si intendono processare.

- **Tabelle complesse**: la tabella che si ottiene può risultare complessa. Ottenere una rappresentazione funzionale delle tabelle non è un compito facile; per ora Tabula funziona meglio lavorando su tabelle che non contengono righe o colonne che attraversano diverse celle.

Mostriamo un esempio di tabella difficoltosa da processare per Tabula:

Year	Term	Mark					
		Assignments			Examinations		Grade
		Ass1	Ass2	Ass3	Midterm	Final	
1991	Winter	85	80	75	60	75	75
	Spring	80	65	75	60	70	70
	Fall	80	85	75	55	80	75
1992	Winter	85	80	70	70	75	75
	Spring	80	80	70	70	75	75
	Fall	75	70	65	60	80	70

Figura 2.6: Esempio di tabella complicata da processare. Tabula riesce a lavorare correttamente con la parte della tabella delimitata dalla linea rossa.

2.3 Apache PDFBox

La libreria Apache PDFBox è uno strumento Java open source per lavorare con documenti in formato PDF [20].

Questo progetto permette la creazione di nuovi file PDF, la manipolazione di documenti già esistenti e l'estrazione di contenuti dai documenti PDF. Comprende anche diversi utility da riga di comando ed è pubblicato sotto la licenza Apache v2.0 (Non è possibile utilizzare questo file se non in conformità con la licenza, è possibile richiedere una copia della licenza al sito "*http://www.apache.org/licenses/LICENSE-2.0*". Se non richiesto dalla legge il software viene distribuito "così com'è", senza garanzie o condizioni di alcun tipo [21]).

La struttura di Apache PDF è composta dai seguenti componenti:

- **PDFBox**: la parte principale.

- **FontBox**: gestisce le informazioni sui font.
- **XmpBox**: gestisce i metadati XMP (Extensible Metadata Platform, è uno standard ISO per la creazione, l'elaborazione e lo scambio dei metadati per documenti digitali e dataset).
- **Verifica preliminare (opzionale)**: controlla i file PDF per verificare che siano conformi a PDF/A-1b (standard ISO per l'archiviazione di lungo periodo. É un sottoinsieme del formato PDF, ripulito dalle funzioni non pensate per l'archiviazione di lungo periodo).

2.3.1 Utility da riga di comando

PDFBox viene fornito come una serie di utility da riga di comando, che sono disponibili come applicazioni Java standard.

Questi insieme di comandi da terminale è stato creato per diversi scopi:

Estrazione delle immagini

Se si desidera estrarre tutte le immagini di un dato documento si può usare il comando:

java -jar pdfbox-app-2.y.z.jar ExtractImages [OPTIONS] <inputfile>

Questo comando ha i seguenti parametri:

- **-password**: la password del documento PDF.
- **-prefix**: prefisso per il nome dell'immagine, di default è il nome del PDF.
- **-directJPG**: forza l'estrazione diretta di immagini JPEG, falso di default.

Estrazione del testo

Se invece si desidera estrarre il testo di un certo documento PDF si può utilizzare il comando:

java -jar pdfbox-app-2.y.z.jar ExtractText [OPTIONS] <inputfile> [Text file]

Le principali opzioni di questo comando sono le seguenti:

- **-password**: la password del documento.
- **-encoding**: il tipo di codifica del file di testo (ISO-8859-1, UTF-8, UTF-16 BE, ecc.).
- **-console**: se s'intende visualizzare il testo sulla console anziché in un file di testo.
- **-html**: se si desidera ricevere il testo in formato html.
- **-sort**: ordina il testo prima di scriverlo.
- **-force**: consente a PDFBox di ignorare gli oggetti corrotti.
- **-debug**: Abilita il debug circa il consumo di tempo di ogni fase.
- **-startPage**: la prima pagina da cui estrarre il testo, 1 di default.
- **-endPage**: ultima pagina da cui estrarre il testo, l'ultima di default.

Sovrapponi PDF

Per sovrapporre un documento PDF con il contenuto di un altro documento PDF si può utilizzare il seguente comando da terminale:

java -jar pdfbox-app-2.y.z.jar OverlayPDF <input.pdf> [OPTIONS] <output.pdf>

Le principali opzioni di questo comando sono le seguenti:

- **-inputfile**: il documento da sovrapporre.
- **defaultOverlayPDF**: file di default da ricoprire.
- **-position**: dove inserire la sovrapposizione, in primo piano o di sfondo.
- **outputfile**: il nome del file PDF restituito dall'esecuzione.

Debug PDF

Questa applicazione vi permette di controllare ed analizzare la struttura interna di un documento PDF dato, eseguendo il seguente comando:

java -jar pdfbox-app-2.y.z.jar PDFDebugger [inputfile]

Le opzioni per questo comando sono:

- **-password**: password del documento PDF.
- **-viewstructure**: attiva la "vista struttura" all'avvio.
- **-inputfile**: nome del documento PDF del quale si vuole veder la struttura.

Inoltre esistono molte altre applicazioni che ti permettono di decifrare e cifrare un documento PDF, unire più documenti PDF, separare un documento PDF in più documenti PDF, convertire ogni pagina di un file PDF in un'immagine, creare un documento PDF da un file di testo e altro ancora.

2.3.2 Limitazioni conosciute

Come tutti gli strumenti per l'estrazione di dati da documenti PDF, anche la libreria Apache PDFBox ha diverse limitazioni. Vediamone le principali:

- Quando si estrae del testo, può capitare di ottenere testo come

  "G38G43G36G51G5" al posto del testo che ci si aspetta. Questo perché i caratteri sono codificati in una codifica interna senza senso che punta a dei glifi incorporati nel documento PDF.

  L'unico modo per risolvere questo problema è utilizzare la tecnologia OCR, che però, come già visto, non è ancora a buon punto con l'automazione dell'esecuzione su molti documenti PDF.

- Può capitare di ottenere il seguente messaggio d'errore: "java.IOException: Can't hadle font width". Potrebbe essere dovuto al fatto che non si ha la directory "org/apache/pdfbox/resources" all'interno del classpath (parametro della Java Virtual Machine o del compilatore Java che specifica la posizione dei pacchetti e delle classi definite dall'utente). Per risolvere questo problema basta includere "apache-pdfbox-x.x.x.jar nel classpath.

- Un altro problema che può verificarsi è quello di ottenere il testo corretto ma in ordine sbagliato. Può essere dovuto al fatto che non è stato attivato l'ordinamento, il testo nei file PDF viene memorizzato a blocchi e per restituirlo in ordine questi blocchi vanno memorizzati nell'ordine nel quale vengono visualizzati su una pagina. Per impostazione predefinita PDFBox non ordina il testo.

2.4 Applicazioni per l'estrazione di dati da articoli scientifici in formato PDF

Guardiamo ora più nel dettaglio l'implementazione delle applicazioni partecipanti al Task 2 della SemPub 2015, elencate nel capitolo precedente:

2.4.1 Costruzione automatica di un Semantic Knowledge Base

La soluzione proposta da Bahar Sateli e René Witte dell'università Concordia di Montréal (Canada), ha come obbiettivo finale quello di estrarre automaticamente le informazioni da una data letteratura scientifica e conservarle in un linguaggio utilizzabile sia dagli umani che dalle macchine [3].

Un requisito essenziale di questa applicazione è l'esistenza di vocabolari controllati con lo scopo di descrivere entità di informazioni tramite l'uso di vocaboli predefiniti.

Gli input di questa procedura di estrazione del testo sono documenti PDF, HTML o testo semplice da URL. Il primo compito di questa procedura è quello di normalizzare il testo del documento e l'output, e etichettare ogni elemento lessicale con un tag, ad esempio aggettivo o pronome. Dopo di questo il testo verrà passato ad un sottosistema di elaborazione semantica per il rilevamento delle entità.

La procedura è quindi composta da due fasi principali:

1. una procedura per l'estrazione di testo, che per prima cosa cerca di convertire in una struttura unificata la base di rappresentazione dei vari documenti (HTML, PDF, LATEX), e poi estrae le entità strutturali e semantiche, come le informazioni degli autori e i riferimenti, dal testo e crea della annotazioni semantiche scritte.

2. un modulo flessibile per esportare le annotazioni del documento in triple RDF secondo un file di mapping personalizzato, sfruttando una combinazione di molteplici tecniche di Natural Language Processing e Semantic Web per costruire una rappresentazione semantica delle informazioni contenute in un documento scientifico.

2.4.2 Estrazione di informazioni testuali da articoli scientifici in formato PDF tramite CERMINE

Descriviamo l'implementazione della soluzione proposta da Dominika Tkaczyk e Lukasz Bolikowski dell'Università di Varsavia, un'applicazione basata su CERMINE [4].

CERMINE è un sistema open source per l'estrazione di metadati strutturati e riferimenti da articoli scientifici digitali [22]. Questo sistema è in grado di estrarre anche informazioni relative al contesto nel quale l'articolo è stato scritto, ad esempio gli autori e le loro affiliazioni o riferimenti ad altri articoli. CERMINE è stato progettato come una soluzione universale, capace di gestire una grande varietà di layout ed è basato su un flusso di lavoro modulare che consente facilmente di apportare dei miglioramenti e di adattarsi a nuovi stili e layout di articoli.

Il sistema CERMINE accetta articoli scientifici in formato PDF in input, ne ispeziona l'intero contenuto e fornisce due tipi di output: i metadati dell'articolo e la bibliografia.

Per fare ciò, CERMINE, durante l'analisi utilizza un struttura gerarchica che contiene l'intero testo dell'articolo, che viene rappresentato come una lista di pagine contenenti una lista di zone, ciascuna zona contiene una lista di righe che a loro volta contengono un elenco di parole, ed infine le parole contengono un elenco di caratteri. Questa struttura contiene l'ordine di lettura naturale per gli elementi su ogni livello della struttura e delle etichette che descrivono il ruolo delle zone.

esecuzione di CERMINE

Il flusso di lavoro di CERMINE per estrarre informazione è composto dalle seguenti fasi:

- estrazione della struttura base: analisi del file PDF di input e costruzione della sua rappresentazione gerarchica nel seguente modo:

    1. estrazione dei singoli caratteri e delle loro coordinate e delle dimensioni dall'articolo di input.

2. costruire la struttura gerarchica dell'articolo contenente le pagine, zone, righe, parole e caratteri.
3. determinare l'ordine di lettura per tutti gli elementi della struttura.
4. classificare le zone dell'articolo in categorie: metadati, corpo, riferimenti, eccetera.

- estrazione di un ricco insieme di metadati dell'articolo provenienti dalle zone etichettate come metadati eseguendo le seguenti operazioni:

  1. classificare le zone dell'articolo in specifiche classi di metadati.
  2. estrarre informazioni dai metadati dalle zone etichettate.
  3. identificare organizzazione, indirizzo e paese nelle stringhe di affiliazione.

- estrazione di una lista di citazioni dalle zone etichettate come riferimenti, eseguendo le seguenti operazioni:

  1. dividere le zone etichettate come riferimenti in stringhe di riferimento individuali.
  2. estrarre i metadati dalle stringhe di riferimento utilizzando l'oggetto classificatore CRF.

2.4.3 Tecniche di apprendimento automatico per estrarre automaticamente informazioni da pubblicazioni scientifiche

La soluzione proposta da Stefan Klampfl e Roman Kern è un sistema basato sulla libreria open source PDFBox, flessibile e modulare. La sua procedura di elaborazione, che analizza la struttura di un articolo PDF, incorpora diverse tecniche di apprendimento della macchina [5].

Vediamone il procedimento:

1. Per la prima parte viene utilizzato l'apprendimento non supervisionato per estrarre i blocchi di testo contigui come unità logiche dell'articolo: prima di poter estrarre informazioni testuali da un articolo scientifico dev'essere elaborato il flusso di caratteri di basso livello del file PDF per ottenere le unità logiche, tipo parole o linee. Per questo viene utilizzato PDFBox, che restituisce l'elenco dei caratteri presenti nel testo, con la loro posizione (posizione x e y sulla pagina, larghezza e altezza) e le informazioni sul loro font.

   In questo sistema vengono presi in considerazione i blocchi di testo contigui come elementi di base di un articolo PDF. Ogni blocco è costituito da diverse linee, contenenti delle parole, a loro volta composte da più caratteri. Per combinare in modo iterativo singoli caratteri in parole, linee e blocchi di testo vengono utilizzate le tecniche di apprendimento automatico non supervisionato.

2. Successivamente viene utilizzato l'apprendimento supervisionato, impiegato per classificare i blocchi in differenti categorie di metadati, inclusi autori e affiliazioni: queste tecniche di apprendimento automatico supervisionato utilizzano degli esempi di formazione etichettati per imparare uno schema di classificazione per i singoli elementi di testo di un articolo. Questo punto dell'esecuzione è diviso in 2 fasi:

   - bisogna classificare i blocchi di testo estratti dall'articolo PDF in più categorie di metadati: le informazioni relative agli autori (nome, indirizzo e-mail, affiliazioni), quelle relative al titolo, al nome della rivista, delle conferenze, della sede e delle parole chiave.
   - riapplicare la classificazione ai blocchi etichettati come metadati relativi agli autori per ottenere i nomi, i cognomi e le affiliazioni.

3. Poi, tramite un insieme di algoritmi euristici, vengono rilevate le sezioni dei riferimenti alla fine dell'articolo, vengono divise in singole stringhe di riferimenti e vengono catalogati il tipo dei singoli riferimenti all'interno di ogni stringa di riferimento.

4. Infine vengono usate le tecniche per il riconoscimento degli oggetti per estrarre i riferimenti alle borse di ricerca, agenzie di finanziamento e progetti comunitari. Per

fare ciò è stata costruita dagli autori un'ontologia dei concetti d'informatica utilizzata per la rilevazione automatica dei riferimenti ai finanziamenti, alle sovvenzioni e ad altri dati del progetto.

2.4.4 Generazione automatica di publishing Linked Dataset: articoli da CEUR-WS

Francesco Ronzano, Beatriz Fisas, Gerard Casamayor del Bosque e Horacio Saggion dell'università Pompeu Fabra di Barcellona hanno proposto un sistema che genera automaticamente ricchi insiemi di dati RDF da degli articoli contenuti in CEUR-WS (database di articoli scientifici [23]) e li espone come Linked Data.

Vediamo quali sono le fasi principali della procedura:

1. I contenuti testuali degli articoli vengono analizzati attraverso apposite procedure di elaborazione del testo.

2. Vengono aggiunte le annotazioni semantiche da un insieme di classificatori SVM (Support Vector Machines, metodologie di apprendimento supervisionato per la classificazione di pattern) e perfezionati da euristiche e grammatiche basate su regole.

3. Vengono sfruttati dei servizi web per collegare annotazioni a insiemi di dati esterni, come, ad esempio DBPedia.

4. Infine i dati vengono modellati e pubblicati in un grafo RDF.

2.4.5 MACJa: Metadata And Citations Jailbreaker

Andrea Giovanni Nuzzolese, Silvio Peroni e Diego Reforgiato Recupero, hanno proposto MACJa (Metadata And Citations Jaibreaker) [7]. MACJa è un metodo per

l'elaborazione di articoli scientifici disponibili in CEUR-WS.org e memorizzati come file PDF, utile per estrarre rilevanti dati semantici e pubblicarli in un triplestore RDF in accordo con le ontologie SPAR (Semantic Publishing And Referencing).

MACJa utilizza le ontologie Semantic Publishing and Referencing (SPAR) come modello di riferimento per l'organizzazione della conoscenza scientifica da file PDF. Inoltre utilizza tecniche basate sul Natural Language Processing, tecnologie del Semantic Web e buone pratiche di progettazione di ontologie per trattare col linguaggio naturale ed estrarre metadati rilevanti.

La procedura di MACJa per estrarre i dati è un'esecuzione sequenziale di diversi script e tool scritti prevalentemente in Python, PHP, Java e Perl:

1. Il primo script eseguito, detto estrattore di testo, ha il compito di estrarre il testo completo dell'articolo dal file PDF.

2. Gli script successivi iniziano la loro esecuzione dall'output del primo script e restituiscono degli appropriati oggetti JSON contenenti le informazioni estratte.

3. Infine questi oggetti JSON vengono convertiti in RDF secondo le ontologie SPAR e pubblicate sul triplestore MACJa.

2.4.6 Estrazione di metadati da articoli di convegni usando un approccio basato su modelli

Liubov Kovriguina, Alexander Shipilo, Fedor Kozlov, Maxim Kolchin e Eugene Cherny hanno proposto uno strumento per l'estrazione di metadati basato su modelli di corrispondenza (pattern matching) e conversione di questi metadati in un dataset Linked Open Data (LOD) implementato nel linguaggio Python [8].

L'obiettivo è quello di scansionare e analizzare documenti PDF provenienti dal sito web CEUR-WS e creare un dataset Linked Open Data contenente informazioni dettagliate su documenti, citazioni, autori e le loro organizzazioni.

La procedura di estrazione dei metadati è basata sulle espressioni regolari, metodi per processare il linguaggio naturale ed euristica riguardante lo stile di documenti HTML (font, dimensione, ecc.) combinati tra loro.

Questo strumento è implementato utilizzando Grab Spider (un framework che permette di descrivere i crawler (software che analizza i contenuti di una rete o di un database in maniera metodica ed automatizzata) di siti web come un insieme di gestori [24]) che permette di costruire crawler per siti asincroni.

Vediamone la procedura:

1. il crawler scarica tutti gli articoli di convegni ed esegue le operazioni di analisi.
2. L'analizzatore di documenti utilizza la libreria per l'estrazione di metadati per raccogliere informazioni riguardanti l'articolo.
3. Il tool utilizza il modulo Ontology Mapper per costruire le proprietà e le entità di relazione.
4. Il modulo Ontology Mapper utilizza la libreria RDFLib per creare e salvare triple.

Capitolo 3

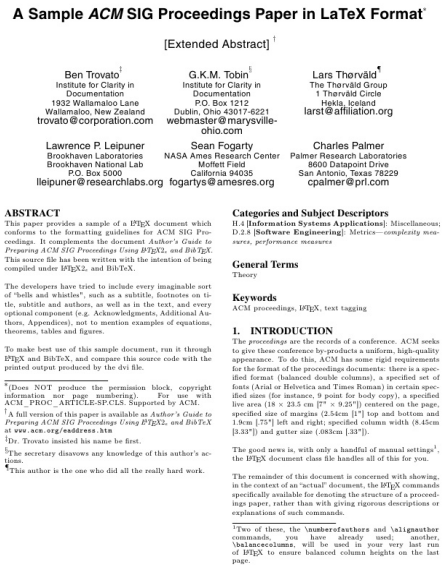
Investiga

Lo scopo del progetto per la tesi è quello di convertire alcune informazioni presenti negli articoli scientifici in formato PDF in un linguaggio interpretabile dalle macchine, e quindi di creare un'applicazione per l'estrazione automatica di informazioni da articoli scientifici in formato PDF e la pubblicazione di tali informazioni estratte secondo i principi e i formati Linked Open Data.

In particolare l'estrazione dei nomi dei capitoli di primo livello, delle didascalie delle figure e delle tabelle da articoli PDF con layout ACM, che è un formato in cui l'articolo PDF è diviso in 2 colonne, e con layout LNCS, formato in cui l'articolo PDF è composto da una sola colonna di testo, e la creazione di un grafo RDF contenenti le informazioni estratte opportunamente collegate tra loro, che, a seconda della scelta dell'utilizzatore, verrà restituito in un file di testo o ne verrà eseguita la post su un triplestore RDF indicato.

Queste informazioni sono utili per poter creare in automatico l'indice, la lista delle figure e la lista delle tabelle di un dato articolo scientifico.

In figura mostriamo un esempio dei layout ACM e LNCS:



A Sample *ACM* SIG Proceedings Paper in LaTeX Format*

[Extended Abstract] †

Ben Trovato‡
Institute for Clarity in Documentation
1932 Wallamaloo Lane
Wallamaloo, New Zealand
trovato@corporation.com

G.K.M. Tobin§
Institute for Clarity in Documentation
P.O. Box 1212
Dublin, Ohio 43017-6221
webmaster@marysville-ohio.com

Lars Thørväld¶
The Thørväld Group
1 Thørväld Circle
Hekla, Iceland
larst@affiliation.org

Lawrence P. Leipuner
Brookhaven Laboratories
Brookhaven National Lab
P.O. Box 5000
lleipuner@researchlabs.org

Sean Fogarty
NASA Ames Research Center
Moffett Field
California 94035
fogartys@amesres.org

Charles Palmer
Palmer Research Laboratories
8600 Datapoint Drive
San Antonio, Texas 78229
cpalmer@prl.com

ABSTRACT

This paper provides a sample of a LaTeX document which conforms to the formatting guidelines for ACM SIG Proceedings. It complements the document *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2ε and BibTeX*. This source file has been written with the intention of being compiled under LaTeX2ε and BibTeX.

The developers have tried to include every imaginable sort of "bells and whistles", such as a subtitle, footnotes on title, subtitle and authors, as well as in the text, and every optional component (e.g. Acknowledgments, Additional Authors, Appendices), not to mention examples of equations, theorems, tables and figures.

To make best use of this sample document, run it through LaTeX and BibTeX, and compare this source code with the printed output produced by the dvi file.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

The *proceedings* are the records of a conference. ACM seeks to give these conference by-products a uniform, high-quality appearance. To do this, ACM has some rigid requirements for the format of the proceedings documents: there is a specified format (balanced double columns), a specified set of fonts (Arial or Helvetica and Times Roman) in certain specified sizes (for instance, 9 point for body copy), a specified live area (18 × 23.5 cm [7" × 9.25"]) centered on the page, specified size of margins (2.54cm [1"] top and bottom and 1.9cm [.75"] left and right; specified column width (8.45cm [3.33"]) and gutter size (.083cm [.33"]).

The good news is, with only a handful of manual settings[1], the LaTeX document class file handles all of this for you.

The remainder of this document is concerned with showing, in the context of an "actual" document, the LaTeX commands specifically available for denoting the structure of a proceedings paper, rather than with giving rigorous descriptions or explanations of such commands.

†A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2ε and BibTeX* at `www.acm.org/eaddress.htm`

‡Dr. Trovato insisted his name be first.

§The secretary disavows any knowledge of this author's actions.

¶This author is the one who did all the really hard work.

[1]Two of these, the `\numberofauthors` and `\alignauthor` commands, you have already used; another, `\balancecolumns`, will be used in your very last run of LaTeX to ensure balanced column heights on the last page.

Jointly published by Akadémiai Kiadó, Budapest and Springer, Dordrecht

Scientometrics, Vol. 75, No. 3 (2008) 519–534
DOI: 10.1007/s11192-007-1781-1

Contribution of Chinese publications in computer science: A case study on LNCS

YING HE,[a] JIANCHENG GUAN[b]

[a] *School of Management, Beijing University of Aeronautics and Astronautics, Beijing (P. R. China)*
[b] *School of Management, Fudan University, Shanghai (P. R. China)*

Conference proceedings are one of the key communication channels in computer science. This paper aims to analyse the Chinese outputs in the context of conference papers in computer science through an exploration of the conference proceedings series book – *Lecture Notes in Computer Science* (*LNCS*) in the period of 1997–2005. Results indicate that: 1. The number of Chinese papers in *LNCS* keeps growing in the studied period; the share of Chinese papers in *LNCS* in recent years is much higher than that of Chinese SCI papers in the world; In sharp contrast with remarkable growth of the share of Chinese papers in *LNCS*, the share of SCI articles in top journals of computer science published by the scientists of mainland China is negligible during the same period. 2. Chinese researchers are more likely to collaborate with domestic fellows; 3. In spite of the increasing amounts of Chinese papers in *LNCS*, they receive only a few citations; 4. The articles are strikingly more cited by authors themselves and international authors' citations are more than Chinese authors' non-self-citations in the first three years after publication; 5. Based on the new indicator Impact Index (II) the authors proposed, the relative impact of Chinese articles in *LNCS* is increasing although the average impact of Chinese papers in *LNCS* is obviously less than that of the publications in *LNCS* in each year during the studied period.

Introduction

By all accounts, computer science and technology sector is a rapidly developing area. It has been considered as a source of economic growth and a strategic tool to enhance the quality of life of the population [GU, 2002; ROJO & GÓMEZ, 2006]. In the

Received July 30, 2007

Address for correspondence:
JIANCHENG GUAN
School of Management, Fudan University, 670 Guoshun Road, Shanghai 200433, P. R. China
E-mail: guanjiancheng@hotmail.com; guanjiancheng@fudan.edu.cn; guanjiancheng@sina.com

0138–9130/$ $ 20.00

Figura 3.1: Esempio formato ACM

Figura 3.2: Esempio formato LNCS

Per iniziare il mio lavoro mi è stato assegnato un insieme di articoli PDF, composto da 23 documenti con layout o ACM o LNCS sui quali lavorare.

Questo progetto è stato chiamato *Investiga*, è stato scritto interamente in python ed è diviso in diversi moduli per migliorarne la leggibilità, e, per l'implementazione della prima parte, è stata utilizzata la libreria PDFMiner.

L'esecuzione dell'applicazione è descritta dalla seguente immagine:

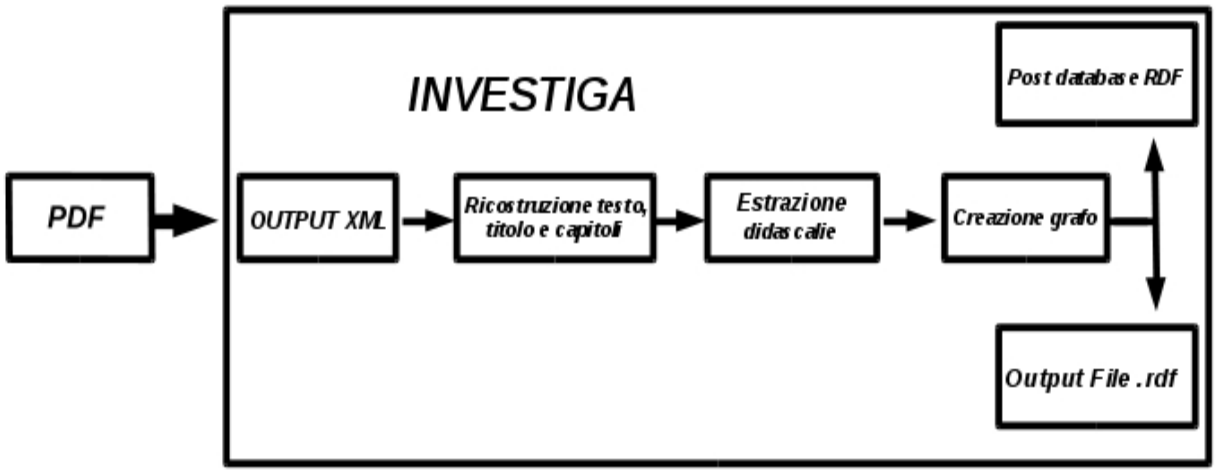


Figura 3.3: In figura vengono mostrati i vari passaggi dell'esecuzione di *Investiga*.

La figura mostra la procedura di *Investiga*:

1. Convertire l'articolo PDF in un documento XML, dal quale è più facile estrarre informazioni, attraverso l'utilizzo della libreria PDFMiner, appositamente modificata.

2. Ricostruire il testo dell'articolo PDF originale pagina per pagina, attraverso un ciclo per ogni riga del file XML, e spaziarlo con la funzione presente nel modulo *inserisci_spazi.py* per inserire gli spazi nelle didascalie delle figure e delle tabelle, e contemporaneamente ricostruire i capitoli di primo livello e i titoli (non necessari per la valutazione della SemPub), nei quali verranno inseriti gli spazi dalle altre funzioni create appositamente contenute nel modulo *inserisci_spazi.py*. Durante questo ciclo vengono anche individuate le posizioni delle figure grazie ai tag "figure".

3. Una volta ricostruito tutto il testo, ricercarne all'interno le didascalie delle figure e delle tabelle presenti in ogni pagina, che saranno già spaziate correttamente. Per la ricerca delle didascalie delle figure è molto importante l'utilizzo delle posizioni individuate precedentemente, soprattutto nelle pagine contenenti più di una figura.

4. Creare il grafo relativo all'articolo processato con tutte le informazioni raccolte, e, a seconda dell'opzione scelta, eseguire la post di questo grafo su un database RDF oppure restituire il grafo in un documento di testo.

La procedura verrà descritta più precisamente nel capitolo successivo.

Guardiamo ora il ruolo di PDFMiner all'interno di *Investiga*:

3.1 Utilizzo di PDFMiner

La prima cosa che è stata fatta per la creazione del progetto per la tesi è stata quella di utilizzare PDFMiner per convertire l'articolo PDF in un documento XML, dal quale estrarre tutte le informazioni di cui si ha bisogno.

Siccome l'utilizzo di PDFMiner nel mio progetto era riservato solo a questa trasformazione, ho modificato i moduli relativi eliminando tutta la parte che non mi interessava, lasciando solo le chiamate per la conversione di documenti PDF in documenti XML. La figura seguente riporta un esempio di output della chiamata a PDFMiner:

```
<?xml version="1.0" encoding="utf-8" ?>
<pages>
<page id="1" bbox="0.000,0.000,595.000,842.000" rotate="0">
<text font="BDQPNF+CMBX12" bbox="173.880,678.842,184.131,692.052" size="13.210">0</text>
<text font="BDQPNF+CMBX12" bbox="184.142,678.842,191.733,692.052" size="13.210">n</text>
<text font="BDQPNF+CMBX12" bbox="196.287,678.842,201.595,692.052" size="13.210">t</text>
<text font="BDQPNF+CMBX12" bbox="201.601,678.842,209.192,692.052" size="13.210">h</text>
<text font="BDQPNF+CMBX12" bbox="209.192,678.842,215.423,692.052" size="13.210">e</text>
<text font="BDQPNF+CMBX12" bbox="219.981,678.842,226.054,692.052" size="13.210">c</text>
<text font="BDQPNF+CMBX12" bbox="226.054,678.842,232.880,692.052" size="13.210">o</text>
<text font="BDQPNF+CMBX12" bbox="232.886,678.842,239.711,692.052" size="13.210">g</text>
<text font="BDQPNF+CMBX12" bbox="239.718,678.842,247.309,692.052" size="13.210">n</text>
<text font="BDQPNF+CMBX12" bbox="247.309,678.842,251.098,692.052" size="13.210">i</text>
<text font="BDQPNF+CMBX12" bbox="251.104,678.842,256.412,692.052" size="13.210">t</text>
<text font="BDQPNF+CMBX12" bbox="256.418,678.842,260.207,692.052" size="13.210">i</text>
<text font="BDQPNF+CMBX12" bbox="260.213,678.842,267.416,692.052" size="13.210">v</text>
<text font="BDQPNF+CMBX12" bbox="267.048,678.842,273.278,692.052" size="13.210">e</text>
<text font="BDQPNF+CMBX12" bbox="277.837,678.842,284.468,692.052" size="13.210">a</text>
```

Figura 3.4: In figura è mostrato un esempio di file XML creato dalla conversione di PDFMiner da documento PDF a file XML

Il file XML così generato è composto da una riga per ogni carattere presente all'interno dell'articolo PDF originale, con indicato, per ogni carattere, il font, la posizione all'interno dell'articolo e la dimensione. Tutte queste informazioni vengono raccolte attraverso l'utilizzo di espressioni regolari per l'esecuzione dell'applicazione.

3.2 Utilizzo di Investiga

Per funzionare *Investiga*, che è il nome della mia applicazione per l'estrazione automatica di informazioni da articoli scientifici in formato PDF e creazione di dataset RDF, ha bisogno della libreria PDFMiner e del pacchetto SPARQLWrapper (modulo per python per i servizi SPARQL, aiuta a creare gli URI della query e a convertire il risultato in un formato più gestibile [25]) presenti nella directory contenente l'applicazione.

Per eseguire bisogna lanciare da terminale il comando:

main.py DirectoryInput DirectoryOutfileXML Opzioni[-rdf |-graph] [DirectoryOutput |UrlEndpoint e UriGrafo]

dove:

- **DirectoryInput**: è l'indirizzo della directory contenente gli articoli PDF che si intendono processare.
- **DirectoryOutfileXML**: è l'indirizzo della directory nella quale si intende inserire i file XML generati dalla conversione da PDF a XML.
- **-rdf**: indicando l'opzione *-rdf* si dice all'applicazione di produrre in output dei file di testo in formato turtle contenenti i grafi prodotti dall'esecuzione, uno per ogni articolo più uno complessivo di tutti gli articoli presenti nella directory di input. Scegliendo questa opzione dopo andrà indicato **DirectoryOutput**, cioè l'indirizzo della directory nella quale si intendono inserire questi file.rdf.

- **-graph**: scegliendo l'opzione *-graph* si dice all'applicazione di fare la post del grafo ottenuto durante l'esecuzione. Bisogna quindi indicare **UrlEndpoint** e **UriGrafo** sui quali si intende eseguire la post (due parametri separati).

Se inoltre si vuole utilizzare il modulo *creazione_risultati.py*, che produce in output dei file di testo contenenti le informazioni estratte durante l'esecuzione, bisogna creare una directory chiamata "risultati"all'interno della directory contenente *Investiga*, e decommentare la riga *CAPITOLI = creazione_risultati.creaFileRisultati(doc, Titolo, CAPITOLI, FIGURE, TABELLE, colonne)* all'interno del modulo *main.py*.

Spiegheremo nel prossimo capitolo l'implementazione e l'utilizzo dei vari moduli.

3.2.1 my_get.py

Se s'intende estrarre i dati dal database RDF utilizzato per le varie post di *Investiga*, si può utilizzare il tool **my_get.py**. Questo strumento restituisce, per ogni articolo indicato, un file in formato CSV contenente le informazioni indicate relative al articolo PDF presente sul database RDF.

Per utilizzarlo bisogna utilizzare il seguente comando da terminale:

my_get.py DirectoryInput Opzioni[capitoli |tabelle |figure |all] FileConfigurazione UrlEndpoint UriGrafo DirectoryOutput

dove:

- **DirectoryInput**: è la directory contenente gli articoli PDF dei quali si intende estrarre le informazioni presenti nel database RDF.
- **Opzioni**: con questo parametro si indica quale tipo di informazione si vuole estrarre: se si indica *capitoli* si chiede di estrarre i nomi dei capitoli di primo livello, con *figure* di estrarre le didascalie delle figure, con *tabelle* di estrarre le didascalie delle tabelle. Inoltre si può anche utilizzare l'opzione *all* per indicare all'applicazione

di estrarre tutte le informazioni (sia capitoli, sia didascalie figure, sia didascalie tabelle).

- **FileConfigurazione**: il file di configurazione è un file di testo contenente il titolo che si vuole assegnare, per ogni articolo PDF cercato nel database RDF, al file CSV restituito da *my_get.py*.
- **UrlEndpoint**: è l'url del database sul quale si vuole eseguire la get.
- **UriGrafo**: è l'uri del grafo sul quale si vuole eseguire la get.
- **DirectoryOutput**: indirizzo della directory nella quale si vuole inserire i file restituiti dall'esecuzione di *my_get.py*.

Ho creato questo tool per agevolarmi con la valutazione: lo script PHP proposto nella SemPub per la valutazione, per funzionare ha bisogno di una directory contenente un file CSV per i capitoli di primo livello, uno per le didascalie delle figure e uno per le didascalie delle tabelle per ogni articolo PDF, chiamato con il nome della query alla quale si riferisce (ad esempio Q4.1.csv, che si riferisce ai capitoli di primo livello dell'articolo 1518_paper1.pdf).

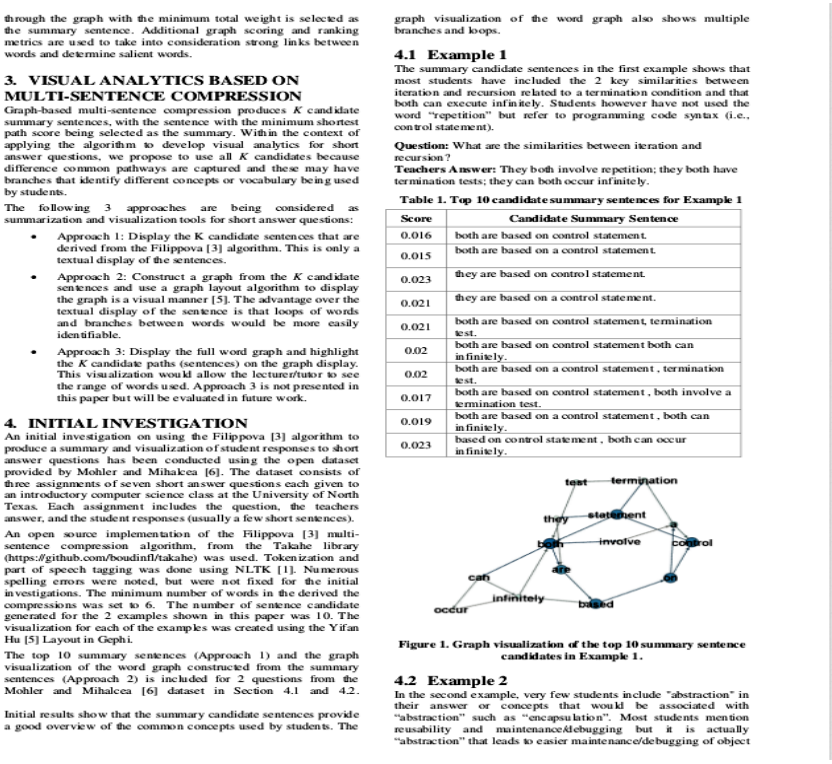
Esempio di file prodotto dalla chiamata a *my_get.py* (nella figura è mostrato il file "Q4.1.csv"in un normale editor di testo):

```
section-iri,    section-number,         section-title
<http://ceur-ws.org/section/vol-1518-paper1_sec1>,      "1"^^xsd:integer,       "INTRODUCTION"
<http://ceur-ws.org/section/vol-1518-paper1_sec2>,      "2"^^xsd:integer,       "PREDICTING ACADEMIC
RISK"
<http://ceur-ws.org/section/vol-1518-paper1_sec3>,      "3"^^xsd:integer,       "VISUALIZING
UNCERTAINTY"
<http://ceur-ws.org/section/vol-1518-paper1_sec4>,      "4"^^xsd:integer,       "CASE-STUDY: RISK TO
FAIL"
<http://ceur-ws.org/section/vol-1518-paper1_sec5>,      "5"^^xsd:integer,       "CONCLUSIONS AND
FURTHER WORK"
<http://ceur-ws.org/section/vol-1518-paper1_sec6>,      "6"^^xsd:integer,       "ACKNOWLEDGMENTS"
<http://ceur-ws.org/section/vol-1518-paper1_sec7>,      "7"^^xsd:integer,       "REFERENCES"
```

Figura 3.5: In figura viene mostrato un esempio di file di output prodotto da *my_get.py*.

3.3 Possibili Output

Osserviamo ora quali sono i possibili output di *Investiga*. Come abbiamo visto sono possibili più output in base alle opzioni indicate da riga di comando: un file.rdf contenente il grafo delle informazioni raccolte, la post di questo grafo su un database e, anche, un file di testo contenente le informazioni estratte (decommentando la chiamata al modulo *creazione_risultati.py*). Mostriamo degli esempi dei vari possibili output per un articolo scientifico in formato PDF:



through the graph with the minimum total weight is selected as the summary sentence. Additional graph scoring and ranking metrics are used to take into consideration strong links between words and determine salient words.

3. VISUAL ANALYTICS BASED ON MULTI-SENTENCE COMPRESSION

Graph-based multi-sentence compression produces *K* candidate summary sentences, with the sentence with the minimum shortest path score being selected as the summary. Within the context of applying the algorithm to develop visual analytics for short answer questions, we propose to use all *K* candidates because difference common pathways are captured and these may have branches that identify different concepts or vocabulary being used by students.

The following 3 approaches are being considered as summarization and visualization tools for short answer questions:

- Approach 1: Display the K candidate sentences that are derived from the Filippova [3] algorithm. This is only a textual display of the sentences.
- Approach 2: Construct a graph from the *K* candidate sentences and use a graph layout algorithm to display the graph is a visual manner [5]. The advantage over the textual display of the sentence is that loops of words and branches between words would be more easily identifiable.
- Approach 3: Display the full word graph and highlight the *K* candidate paths (sentences) on the graph display. This visualization would allow the lecturer/tutor to see the range of words used. Approach 3 is not presented in this paper but will be evaluated in future work.

4. INITIAL INVESTIGATION

An initial investigation on using the Filippova [3] algorithm to produce a summary and visualization of student responses to short answer questions has been conducted using the open dataset provided by Mohler and Mihalcea [6]. The dataset consists of three assignments of seven short answer questions each given to an introductory computer science class at the University of North Texas. Each assignment includes the question, the teachers answer, and the student responses (usually a few short sentences).

An open source implementation of the Filippova [3] multi-sentence compression algorithm, from the Takahe library (https://github.com/boudinfl/takahe) was used. Tokenization and part of speech tagging was done using NLTK [1]. Numerous spelling errors were noted, but were not fixed for the initial investigations. The minimum number of words in the derived the compressions was set to 6. The number of sentence candidate generated for the 2 examples shown in this paper was 10. The visualization for each of the examples was created using the Yifan Hu [5] Layout in Gephi.

The top 10 summary sentences (Approach 1) and the graph visualization of the word graph constructed from the summary sentences (Approach 2) is included for 2 questions from the Mohler and Mihalcea [6] dataset in Section 4.1 and 4.2.

Initial results show that the summary candidate sentences provide a good overview of the common concepts used by students. The graph visualization of the word graph also shows multiple branches and loops.

4.1 Example 1

The summary candidate sentences in the first example shows that most students have included the 2 key similarities between iteration and recursion related to a termination condition and that both can execute infinitely. Students however have not used the word "repetition" but refer to programming code syntax (i.e., control statement).

Question: What are the similarities between iteration and recursion ?
Teachers Answer: They both involve repetition; they both have termination tests; they can both occur infinitely.

Table 1. Top 10 candidate summary sentences for Example 1

Score	Candidate Summary Sentence
0.016	both are based on control statement.
0.015	both are based on a control statement.
0.023	they are based on control statement.
0.021	they are based on a control statement.
0.021	both are based on control statement, termination test.
0.02	both are based on control statement both can infinitely.
0.02	both are based on a control statement , termination test.
0.017	both are based on control statement , both involve a termination test.
0.019	both are based on a control statement , both can infinitely.
0.023	based on control statement , both can occur infinitely.

Figure 1. Graph visualization of the top 10 summary sentence candidates in Example 1.

4.2 Example 2

In the second example, very few students include "abstraction" in their answer or concepts that would be associated with "abstraction" such as "encapsulation". Most students mention reusability and maintenance/debugging but it is actually "abstraction" that leads to easier maintenance/debugging of object

Figura 3.6: La figura mostra un esempio di articolo PDF con layout ACM (precisamente la pagina 2 dell'articolo 1518_paper2.pdf).

Se è stata scelta l'opzione *-rdf*, l'applicazione creerà un file.rdf contenente il grafo delle informazione estratte, come in figura:

```
@prefix ns1: <http://www.essepuntato.it/2008/12/pattern#> .
@prefix ns2: <http://purl.org/vocab/frbr/core#> .
@prefix ns3: <http://purl.org/spar/> .
@prefix ns4: <http://purl.org/spar/doco/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://ceur-ws.org/section/vol-1518-paper2> a <http://purl.org/spar/fabio/JournalArticle> ;
    ns2:part <http://ceur-ws.org/section/vol-1518-paper2-title>,
        <http://ceur-ws.org/section/vol-1518-paper2_fig1>,
        <http://ceur-ws.org/section/vol-1518-paper2_fig2>,
        <http://ceur-ws.org/section/vol-1518-paper2_sec1>,
        <http://ceur-ws.org/section/vol-1518-paper2_sec2>,
        <http://ceur-ws.org/section/vol-1518-paper2_sec3>,
        <http://ceur-ws.org/section/vol-1518-paper2_sec4>,
        <http://ceur-ws.org/section/vol-1518-paper2_sec5>,
        <http://ceur-ws.org/section/vol-1518-paper2_sec6>,
        <http://ceur-ws.org/section/vol-1518-paper2_sec7>,
        <http://ceur-ws.org/section/vol-1518-paper2_sec8>,
        <http://ceur-ws.org/section/vol-1518-paper2_tab1>,
        <http://ceur-ws.org/section/vol-1518-paper2_tab2> .

<http://ceur-ws.org/section/vol-1518-paper2-TitleFigure1> a ns4:FigureLabel ;
    ns3:c4ohasContent "Graph visualization of the top 10 summary sentence candidates in Example 1." .

<http://ceur-ws.org/section/vol-1518-paper2-TitleFigure2> a ns4:FigureLabel ;
    ns3:c4ohasContent "Graph visualization of the top 10 summary sentence candidates in Example 2." .

<http://ceur-ws.org/section/vol-1518-paper2-TitleTable1> a ns4:TableLabel ;
    ns3:c4ohasContent "Top 10 candidate summary sentences for Example 1" .

<http://ceur-ws.org/section/vol-1518-paper2-TitleTable2> a ns4:TableLabel ;
    ns3:c4ohasContent "Top 10 candidate summary sentences for Example 2" .

<http://ceur-ws.org/section/vol-1518-paper2-sectionTitle1> a ns4:SectionTitle ;
    ns3:c4ohasContent "INTRODUCTION" .

<http://ceur-ws.org/section/vol-1518-paper2-sectionTitle2> a ns4:SectionTitle ;
    ns3:c4ohasContent "MULTI-SENTENCE COMPRESSION" .

<http://ceur-ws.org/section/vol-1518-paper2-sectionTitle3> a ns4:SectionTitle ;
```

Figura 3.7: La figura mostra un esempio di file.rdf creato da *Investiga*, (nell'esempio 1518_paper2.rdf).

Se invece è stata scelta l'opzione *-graph*, *Investiga* eseguirà la post dei dati estratti sul database RDF indicato:

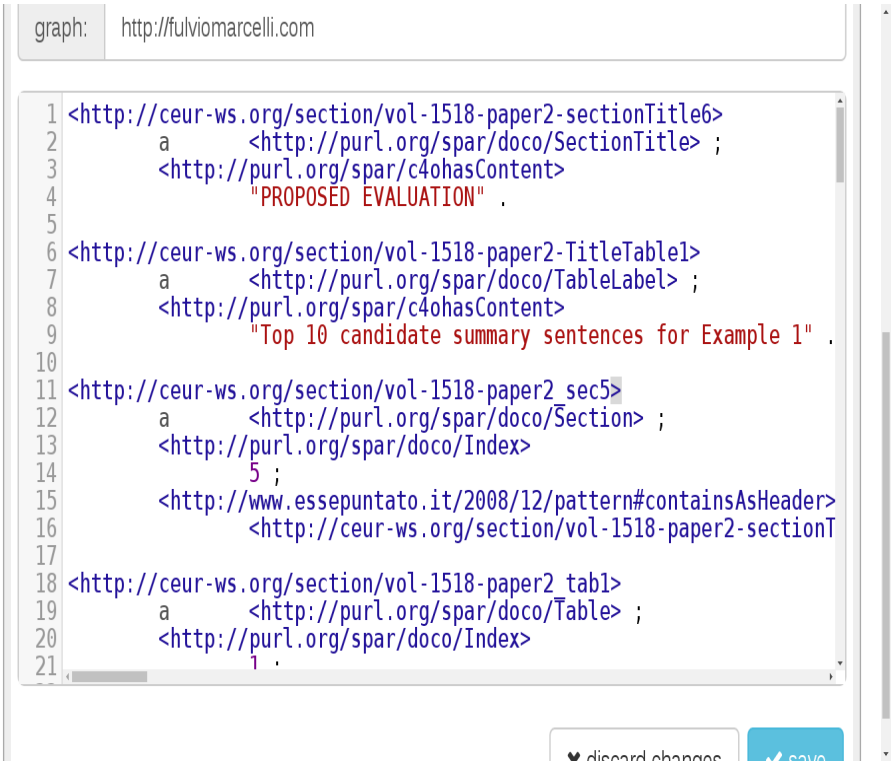


Figura 3.8: La figura mostra un esempio di post da parte di *Investiga*, (nell'esempio eseguita con indicato "*http//:fulviomarcelli.com*" come URI del grafo).

Se è stata decommentata la chiamata a *creazione_risultati*, il file di testo che verrà creato sarà simile al seguente:

```
DOCUMENTO: 1518_paper2


TITOLO:

Using Sentence Compression to Develop Visual Analytics for Student Responses to Short Answer Questions


CAPITOLI DI PRIMO LIVELLO:

1. INTRODUCTION
2. MULTI-SENTENCE COMPRESSION
3. VISUAL ANALYTICS BASED ON MULTI-SENTENCE COMPRESSION
4. INITIAL INVESTIGATION
5. TOOL DESIGN AND FUNCTIONALITY
6. PROPOSED EVALUATION
7. CONCLUSION
8. REFERENCES


FIGURE:

Graph visualization of the top 10 summary sentence candidates in Example 1.

Graph visualization of the top 10 summary sentence candidates in Example 2.



TABELLE:

Top 10 candidate summary sentences for Example 1

Top 10 candidate summary sentences for Example 2
```

Figura 3.9: In figura viene mostrato l'esempio di un file di testo creato dalla chiamata a *creazione_risultati*.

3.4 Problemi conosciuti

Sono stati riconosciuti diversi problemi, cioè situazioni nelle quali *Investiga* non riesce a svolgere il suo lavoro correttamente.

- Nel caso in cui il testo sia redatto sotto forma di immagini, *Investiga* non riesce a riconoscere tale testo come immagine e quindi non rileva la didascalia. Questo succede perché neanche la libreria PDFMiner è in grado di riconoscere queste immagini, e quindi durante la conversione da PDF a XML dell'articolo non inserisce il tag figure relativo al testo redatto sotto forma di immagini, e questa informazione è fondamentale per la corretta esecuzione dell'applicazione.

  Mostriamo un esempio di testo redatto sotto forma di immagine:

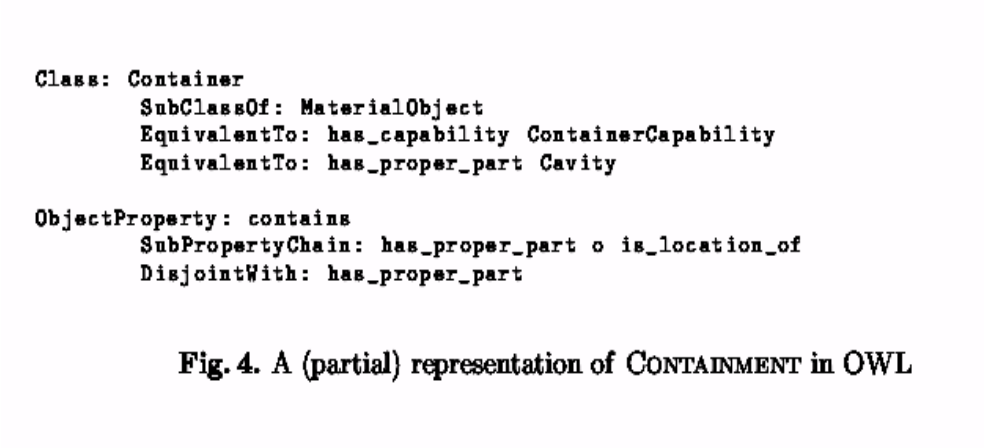


Figura 3.10: In figura viene mostrato un esempio di testo redatto sotto forma di immagine.

- Nel caso in cui la didascalia della figura si trovi nella pagina seguente rispetto a quella nella quale si trova la figura, *Investiga* non riesce a individuarla, perché per cercare le didascalie delle figure utilizza la posizione individuata grazie ai tag "figure" e la pagina nella quale si trova, e in questi casi la pagina non corrisponde perché il tag si trova nella pagina nella quale è presente la figura.

Mostriamo un esempio nella figura seguente:

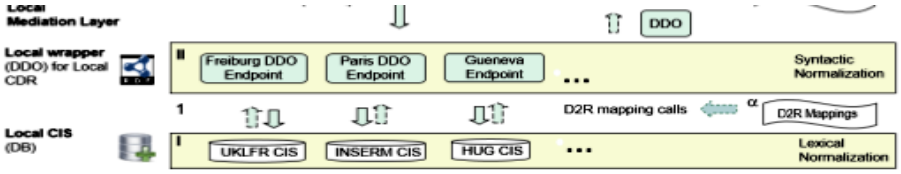


Fig. 1. Layered DebugIT mediator architecture. A schematic overview of the data formalization layers serving distributed data integration in the DebugIT interoperability platform. Clinical data in local relational databases (I) can be accessed in real-time via SPARQL querying the local endpoints via a DSSQ using DDOs on an Extract Transform Load wrapper (II).

Figura 3.11: In figura viene mostrato un esempio di didascalia della figura nella pagina seguente a quella contenente la figura.

- Nel caso in cui la didascalia di una figura o di una tabella sia inserita in mezzo al testo, *Investiga* non riesce a individuarne il termine, e quindi prenderà più testo del necessario riportando una didascalia errata.

  Mostriamo un esempio di didascalia inserita nel testo nella seguente figura:

Fig. 2. Question Authoring Tool. An already formalized query template can be aligned according to a specific research question, by binding it to concrete actual variables by selecting appropriate DO classes.

Figura 3.12: In figura viene mostrato un esempio di didascalia di una figura inserita nel testo.

- Ci sono alcuni problemi per quanto riguarda il riconoscimento dei capitoli di primo livello "*Acknowledgments*" e "*Bibliography*", dovuti al fatto che spesso si trovano scritti in font e dimensioni diverse rispetto agli altri capitoli di primo livello.
- Ci sono alcuni problemi legati a caratteri speciali (ad esempio caratteri dell'alfabeto greco) non codificabili in "*UTF-8*".

Capitolo 4

Implementazione

4.1 Moduli principali

Come già detto, questo strumento per l'estrazione automatica di informazioni da articoli scientifici in formato PDF e creazione dataset RDF è suddiviso in moduli. Diamo quindi una panoramica dei principali compiti di questi moduli e della loro implementazione.

4.1.1 main.py

Il modulo *main.py* è il modulo principale dell'applicazione, riceve in input l'indirizzo della directory contenente i file PDF da processare, inizializza tutte le variabili comuni tra i vari moduli e gestisce l'esecuzione.

Per ogni file presente nella directory di input, utilizzando il modulo *creazione_outfileXML.py* (che è PDFMiner modificato) lo si converte in un file XML e si esegue un ciclo per ogni riga del documento XML nel quale vengono raccolte, attraverso le espressioni regolari, tutte le informazioni di cui si ha bisogno (font, dimensione, posizione, carattere, eccetera), e si riesce anche a distinguere il caso in cui l'articolo PDF sia in formato ACM o LNCS.

Durante questo ciclo, inoltre, viene ricostruito il testo del PDF originale riga per riga e pagina per pagina.

Il documento XML creato dalla chiamata a *creazione_outfileXML.py* è composto da una riga per ogni carattere presente nell'articolo PDF, come mostrato dalla seguente figura:

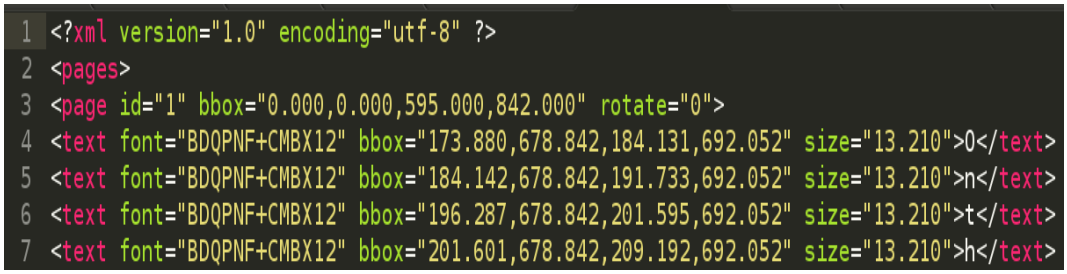


Figura 4.1: In figura viene mostrata una parte del documento XML generato dall'esecuzione di *Investiga*.

Ogni riga del documento XML contiene il font, la posizione all'interno dell'articolo (*bbox*), la dimensione del carattere e il carattere. Grazie alle espressioni regolari possiamo estrarre queste informazioni.

L'espressione regolare *altezza1* viene utilizzata per conoscere l'altezza all'interno dell'articolo originale del carattere corrente, mentre *bbox* verrà usata in seguito per calcolare la distanza tra il carattere corrente e il precedente (utilizzata per l'inserimento degli spazi dalle funzioni del modulo *inserisci_spazi.py*). Mostriamo in figura le relative espressioni regolari:

```
font = re.compile(r"""(                                                    #FONT
                [font="]\w+\s*[+]*\w*\s*\w*\s*[,-]*\w+ ["]
            )""", re.VERBOSE)

dim = re.compile (r"""(                                                  #DIMENSIONE
                [size="][0-9]{1,2}[.]\d+[">]
            )""", re.VERBOSE)

bbox = re.compile (r"""(                                                 #BBOX
                [bbox="][0-9]{2,3}[.][0-9]{3}
            )""",re.VERBOSE)


tabellaBbox = re.compile(r"""(
                [,][0-9]{2,3}[.][0-9]{2,3}["]
            )""", re.VERBOSE)

altezza1 = re.compile(r"""(
                [,][0-9]{2,3}[.][0-9]{3}[,]
            )""", re.VERBOSE)
```

Figura 4.2: In figura vengono mostrate le espressioni regolari utilizzate dal *main.py*.

Le variabili relative al font del testo e la variabile *colonne* (posizione dell'inizio del titolo, utilizzata per distinguere i layout ACM e LNCS, se *colonne* è minore di 131, allora il layout è ACM, altrimenti è LNCS) vengono inizializzate alla quarta riga del documento XML perché è lì che inizia il testo in tutti gli articoli assegnatomi, nel seguente modo:

```
if stringaNumero == 4:
    colonne = bbox.search(string)

    if colonne:
        colonne = int(round(float(colonne.group().strip('"'))))     #
        se il font e' gia' spaziato Times = True
    if fontStringa.find('Times')>-1 or fontStringa.find("Helvetica")>-1
         :
            Times = True

    if fontStringa.find("NimbusSanL")>-1:                # font tutto
    maiuscolo per i capitoli
        Nimbus = True
```

Figura 4.3: In figura viene mostrata l'inizializzazione delle variabili relative al font in *main.py*.

Inoltre vengono distinti i vari tipi di font, se *Times* è True allora il testo è già spaziato, se *Nimbus* è True allora i capitoli di primo livello sono tutti maiuscoli. Queste informazioni serviranno per inserire correttamente gli spazi tra le diverse parole.

Il testo dell'articolo PDF viene ricostruito riga per riga, controllando, tramite le altezze recuperate grazie alle espressioni regolari, se il carattere corrente è sulla stessa altezza della precedente. Se non lo è viene controllato che ci sia una certa distanza tra la riga corrente e la precedente, se non c'è si rimane sulla stessa riga. Questo permette alle didascalie delle figure e delle tabelle che occupano più righe di essere ricostruite su un'unica riga consentendo all'applicazione di individuarle correttamente. Se invece la distanza è maggiore di una certa quantità viene terminata la riga corrente, che viene memorizzata nella struttura dati *PDFrigaPerRigaArray* insieme alla sua altezza, che verrà utilizzata in seguito nel modulo *ricercaFigureTabelle.py*, e iniziata la successiva.

La ricostruzione del testo dell'articolo PDF riga per riga viene mostrato in figura:

```
if str(altezza) ==  str(precAltezza):       #se sono sulla stessa altezza nel PDF
    if primaH :
        PDFrigaPerRiga = PDFrigaPerRiga + precLettera +lettera
        primaH = False

    else:
        PDFrigaPerRiga = PDFrigaPerRiga +lettera

else :
    differenzaBbox = float(precBbox)-float(Bbox2) # spazio tra riga corrente e
    successiva, se e' piu' piccolo di un tot riamngo nella stessa riga

    if  float(differenzaBbox) > 11 or float(differenzaBbox) < -3 or bordo == True :
        bordo = False

        stringa = [PDFrigaPerRiga ,precAltezza]
        PDFrigaPerRigaArray += [stringa]

        PDFrigaPerRiga = ""
        primaH = True
    else :
        if precLettera!="-":
            PDFrigaPerRiga = PDFrigaPerRiga +" "+lettera
        else:
            PDFrigaPerRiga = PDFrigaPerRiga +lettera

precAltezza = altezza
```

Figura 4.4: In figura viene mostrato il codice relativo alla ricostruzione del testo dell'articolo riga per riga.

Per la successiva ricerca delle didascalie delle figure, quando viene trovato il tag *figure* all'interno della riga viene salvata la posizione estratta dalla riga e il numero della pagina corrente, in una struttura dati che verrà passata come parametro alla funzione *ricercaFigure* contenuta nel modulo *ricercaFigureTabelle.py*. Inoltre il testo dell'articolo, per permettere all'applicazione di cercare le didascalie delle figure correttamente nella giusta pagina, viene diviso pagina per pagina. Se si trova una stringa contenente la parola "*page*", tranne la prima volta perché indica l'inizio della prima pagina, vengono inserite le righe ricostruite fino al quel momento nella struttura *paginePDF*, che è la struttura che alla fine del ciclo conterrà tutto il testo dell'articolo originale e verrà passato alle funzioni del modulo *ricercaFigureTabelle.py* per estrarre le didascalie di figure e tabelle. Vengono anche riazzerate tutte le variabile utilizzate per la costruzione del testo riga per riga.

La divisione pagina per pagina del testo viene mostrata nella seguente figura:

```
# ricreo il PDF riga per riga e pagina per pagina
if string.find("page ")>-1:
    pagina = pagina +1

    if primaPagina == False:
        stringa = [PDFrigaPerRiga ,precAltezza]
        PDFrigaPerRigaArray +=[stringa]
        paginePDF +=[PDFrigaPerRigaArray]
        PDFrigaPerRigaArray =[]
        PDFrigaPerRiga = ""

    PDFrigaPerRiga =PDFrigaPerRiga +'\n\n'+" pagina "+str(
        pagina)+'\n\n'
    primaPagina =False
```

Figura 4.5: In figura viene mostrato il codice relativo alla divisione pagina per pagina del testo ricostruito.

4.1.2 creazione_outfileXML.py

Il modulo *creazione_outfileXML.py* è il primo modulo che viene richiamato dal *main.py*. É il modulo contenente la parte di PDFMiner modificata, e viene richiamato per convertire l'articolo PDF corrente in un file XML. Alla funzione al suo interno viene passato l'indirizzo dell'articolo PDF, questa funzione lo apre, lo trasforma in un file XML che andrà a inserire nella directory specificata da riga di comando, chiude l'articolo originale e ritorna l'esecuzione al *main.py* che aprirà il file XML per proseguire con l'esecuzione.

4.1.3 ricercaPDF.py

Il modulo *ricercaPDF.py* viene chiamato per ogni carattere presente nel file XML. Contiene le funzioni per ricercare il titolo (non fa parte delle informazioni necessarie per la valutazione, ma è un informazione che viene estratta ugualmente da questa applicazione) e i capitoli sia nel caso di articolo PDF in formato ACM sia nel caso di articolo PDF in formato LNCS.

Nel *main.py* viene riconosciuto in quale caso ci troviamo (ACM o LNCS) e viene richiamata la funzione relativa nel modulo *ricercaPDF.py* per ogni riga presente nel documento XML. Le funzioni così chiamate, in base al font e la dimensione, riconoscono se la lettera corrente fa parte di un titolo di primo livello oppure no. Se il carattere corrente fa parte di un capitolo di primo livello viene aggiunto alla struttura dati "*Capitolo*". Inoltre, nel caso di layout LNCS la funzione riconosce se il carattere corrente è relativo al capitolo di primo livello successivo a quello rilevato fino a quel momento e separa i capitoli inserendo l'array "*Capitolo*" nella struttura dati "*CAPITOLI*" e lo azzera, mentre nel caso di layout ACM vengono estratti tutti i capitoli e, successivamente, nel *main.py* vengono selezionati solo quelli di primo livello. In entrambi i casi si riconosce la fine di un capitolo grazie alla variabile *nstringa* che contiene il numero della stringa contenente l'ultimo carattere avente font e dimensione dei capitoli, e se è diverso dal numero della stringa corrente meno 1 allora siamo in un'altra parte del testo, e si controlla se il carattere corrente è uguale ad un numero, se questo è verificato e il carattere precedente è diverso da uno spazio e dal carattere "." si suppone appartenere

al capitolo successivo. Mostriamo in figura il codice relativo alla separazione dei capitoli, dove, se presenti, vengono anche individuati ed eliminati i doppi spazi:

```
if (nStringa+1 != stringaNumero) or ('0' <= lettera <='9' and (
    precLettera != "." and precLettera!= " ")):

    if Capitolo:        # se il capitolo e' finito l'aggiungo alla
    struttura CAPITOLI
        if len(Capitolo)>3:
            pos = Capitolo.find("  ")
            if pos > -1:
                temp = Capitolo[pos +1 :]
                Capitolo = Capitolo[:pos] + temp

            Capitolo = inserisci_spazi.elimina_caratteri_speciali(
                Capitolo)
            CAPITOLI += [Capitolo]

        Capitolo = ""
```

Figura 4.6: In figura viene mostrato il codice relativo alla divisione dei capitoli.

4.1.4 ricercaFigureTabelle.py

Questo modulo viene richiamato dal *main.py* alla fine del ciclo eseguito per ogni riga del file XML e contiene le funzioni per la ricerca delle didascalie delle tabelle e delle figure. Alle funzioni del modulo *ricercaFigureTabelle.py* viene passata la struttura dati contenente il testo del PDF originale ricostruito pagina per pagina e anche la posizione delle figure rilevate dal *main.py* grazie ai tag "*figure*" presenti nel file XML. Mostriamo un esempio del tag "*figure*" presenti nel file XML:

```
<page id="3" bbox="0.000,0.000,595.000,842.000" rotate="0">
<figure name="Im1" bbox="178.672,573.136,419.193,690.593">
<image width="240" height="117" />
</figure>
```

Figura 4.7: In figura viene mostrato uno dei tag *figure* presenti nel file XML.

Queste funzioni riconoscono le didascalie delle figure e delle tabelle ricercando le parole chiave *Fig.*, *Figure*, *Table* all'inizio di ogni stringa di ogni pagina presente nella struttura dati contenente il testo dell'articolo ricostruito e le inseriscono nelle strutture dati relative (nel caso delle figure sono molto importanti le posizioni ricevute come parametro assieme alla pagina di riferimento e il fatto che la struttura dati contenente il testo dell'articolo PDF sia diviso pagina per pagina).

4.1.5 inserisci_spazi.py

Il modulo *inserisci_spazi.py* contiene le funzioni per inserire gli spazi tra le varie parole del testo, utili perché, per la maggior parte dei font, trasformando gli articoli PDF di input nei file XML si perdono tutti gli spazi. Questo modulo viene chiamato dal *main.py* durante la ricreazione del testo dell'articolo originale e anche dal modulo *ricercaPDF.py* durante la ricostruzione del titolo e dei capitoli di primo livello.

Le funzioni al suo interno inseriscono gli spazi in base alla distanza tra la lettera corrente e quella precedente, la dimensione del carattere corrente e di quello precedente e al tipo di carattere (lettera, numero, ecc. o anche al tipo di lettera, perché alcune, ad esempio la "m", occupano uno spazio maggiore rispetto ad altre).

4.1.6 creazione_grafo.py

Una volta che *main.py* ha finito di cercare tutte le informazioni all'interno dell'articolo viene chiamato il modulo *creazione_grafo.py* . Questo modulo contiene la funzione

che si occupa di creare un oggetto di tipo grafo con sintassi turtle collegando tutte le informazioni all'articolo in modo adeguato per poter essere postato su un endpoint RDF, oppure anche essere scritto su un file di testo (dipende dalle opzioni inserite da linea di comando), utilizzando il pacchetto SPARQLWrapper.

4.1.7 my_post.py

Se tra le opzioni da riga di comando è stato specificato di voler creare un dataset RDF su un database, dopo aver chiamato la funzione all'interno di *creazione_grafo.py*, il modulo principale *main.py* chiama la funzione contenuta nel modulo *my_post.py*, passandogli come parametro il grafo appena ottenuto. Questa funzione si occuperà di connettersi all'endopoint indicato e di postare il grafo ricevuto come parametro, come mostrato in figura:

```
from rdflib import ConjunctiveGraph
import json


def post(g, URLendpoint, graphURI):     # eseguo la post del grafo g
nel database specificato

    output = {  'risposta': [] }

    endpoint = ConjunctiveGraph ('SPARQLUpdateStore')
    endpoint.open(URLendpoint)
    enddata = """INSERT DATA {
                    GRAPH <"""+graphURI+"""> {
                        %s
                    }
            }""" % g.serialize(format="nt")

    endpoint.update(enddata)
    output['risposta'].append(g.serialize(format="turtle"))
```

Figura 4.8: In figura viene mostrato il codice relativo alla post del grafo sul database RDF specificato.

4.1.8 creazione_risultati.py

Questo modulo contiene una funzione il cui compito è quello di creare un file di testo contenente le informazioni estratte da *Investiga* durante la sua esecuzione. Se la si vuole utilizzare basta decommentare la riga *CAPITOLI = creazione_risultati.creaFileRisultati(doc, Titolo, CAPITOLI, FIGURE, TABELLE, colonne)* all'interno di *main.py*. Inoltre, è necessario creare nella directory contenente l'applicazione una directory chiamata "risultati" dove verranno inseriti i file di testo prodotti.

É molto utile in caso di debugging.

Capitolo 5

Valutazione Investiga

Vediamo ora i risultati della valutazione di *Investiga* sui vari insiemi di articoli proposti.

Come già visto, per la valutazione è stato messo a disposizione uno script PHP che prende in input un file in formato CSV per i capitoli di primo livello, uno per le didascalie delle figure e uno per le didascalie delle tabelle per ogni articolo per il quale si vuole eseguire la valutazione e un file di testo contenente tutte le query relative agli articoli che si intendono valutare. Questo script PHP mette a confronto questi file CSV con i relativi gold standard e fornisce un report contenente i risultati ottenuti. La valutazione è basata sulla variabile F-Score, che come già detto nella descrizione della valutazione della SemPub, è la proporzione tra le misure Precision e Recall, rispettivamente la proporzione tra le informazioni pertinenti e quelle recuperate e la proporzione tra le informazioni attinenti rilevate e quelle disponibili.

Più precisamente:
F-Score = $2 * \frac{Precision * Recall}{Precision + Recall}$

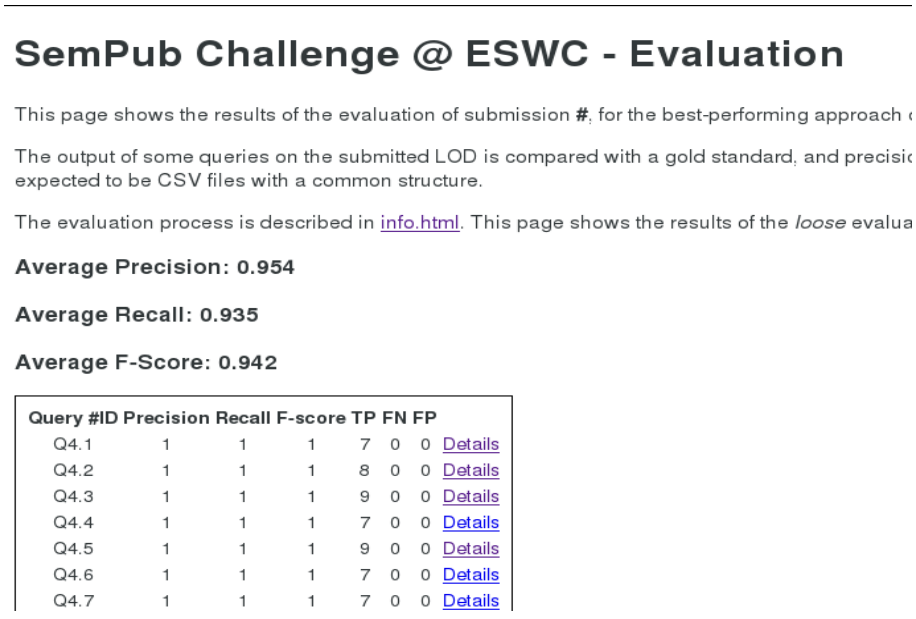
5.1 Valutazione articoli PDF nei formati ACM e LNCS

La valutazione sull'insieme di articoli PDF nei formati ACM e LNCS, cioè l'insieme che mi è stato assegnato per la creazione del progetto, contenente 23 file PDF, riporta questi risultati:

Average Precision	0.954
Average Recall	0.935
Average F-Score	0.942

Tabella 5.1: La tabella riporta i risultati della valutazione sugli articoli PDF con layout ACM e LNCS.

Mostriamo in figura il report creato dallo script PHP relativo alla valutazione sul dataset di articoli con layout ACM o LNCS:



SemPub Challenge @ ESWC - Evaluation

This page shows the results of the evaluation of submission #, for the best-performing approach

The output of some queries on the submitted LOD is compared with a gold standard, and precisi
expected to be CSV files with a common structure.

The evaluation process is described in info.html. This page shows the results of the *loose* evalua

Average Precision: 0.954

Average Recall: 0.935

Average F-Score: 0.942

Query #ID	Precision	Recall	F-score	TP	FN	FP	
Q4.1	1	1	1	7	0	0	Details
Q4.2	1	1	1	8	0	0	Details
Q4.3	1	1	1	9	0	0	Details
Q4.4	1	1	1	7	0	0	Details
Q4.5	1	1	1	9	0	0	Details
Q4.6	1	1	1	7	0	0	Details
Q4.7	1	1	1	7	0	0	Details

Figura 5.1: In figura viene mostrato il report della valutazione sul dataset contenente i 23 articoli PDF con layout ACM e LNCS.

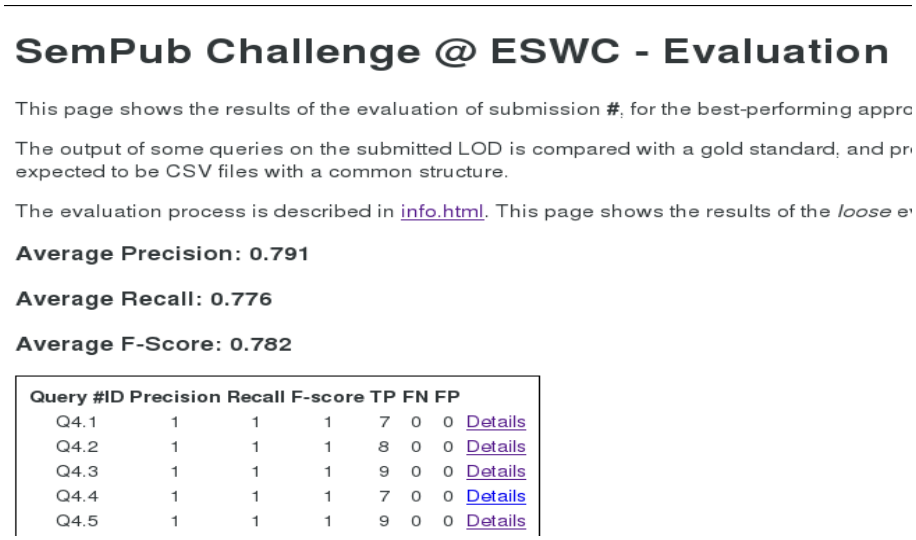
5.2 Valutazione articoli PDF nell'evaluetion dataset

Eseguendo la valutazione su tutti gli articoli del training Dataset, cioè l'insieme di 45 articoli PDF assegnati per realizzare i vari progetti ai partecipanti alla SemPub 2016, i risultati ottenuti sono i seguenti:

Average Precision	0.791
Average Recall	0.776
Average F-Score	0.782

Tabella 5.2: La tabella riporta i risultati della valutazione sugli articoli PDF del Training Dataset.

Mostriamo in figura il report relativo alla valutazione effettuata tramite lo script PHP:



SemPub Challenge @ ESWC - Evaluation

This page shows the results of the evaluation of submission #, for the best-performing appro

The output of some queries on the submitted LOD is compared with a gold standard, and pr expected to be CSV files with a common structure.

The evaluation process is described in info.html. This page shows the results of the *loose* e

Average Precision: 0.791

Average Recall: 0.776

Average F-Score: 0.782

Query #ID	Precision	Recall	F-score	TP	FN	FP	
Q4.1	1	1	1	7	0	0	Details
Q4.2	1	1	1	8	0	0	Details
Q4.3	1	1	1	9	0	0	Details
Q4.4	1	1	1	7	0	0	Details
Q4.5	1	1	1	9	0	0	Details

Figura 5.2: In figura viene mostrato il report della valutazione sugli articoli PDF del Training Dataset.

Le principali cause di questo calo di punteggio, che era preventivabile, oltre alle cause elencate nel capitolo precedente, sono dovute al fatto che, a differenza dell'estrazione delle didascalie delle tabelle che funziona abbastanza correttamente, per alcuni tipi di formato l'estrazione dei capitoli di primo livello e delle didascalie delle figure non funzionano correttamente. Nel caso dei capitoli di primo livello alcuni articoli usano i numeri romani al posto di quelli arabi per indicare l'inizio di un capitolo, in questo caso *Investiga* non riesce a riconoscere questi capitoli. Nel caso delle didascalie delle figure i problemi maggiori derivano dall'inserimento degli spazi in modo errato, probabilmente dovuto al fatto che si presentano dei casi non gestiti, alla presenza di caratteri speciali (per esempio lettere dell'alfabeto greco), e al fatto che in alcuni casi *Investiga* rileva più didascalie di quelle effettivamente presenti. Mostriamo un esempio delle cause dei principali errori nella seguenti figure:

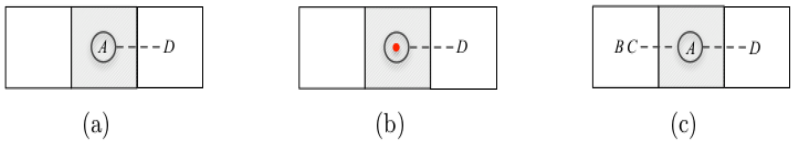


Fig. 4. Graphical representation of spatial itemsets (a) $A \cdot D$ (b) $\theta \cdot D$ (c) $A \cdot [BC; D]$

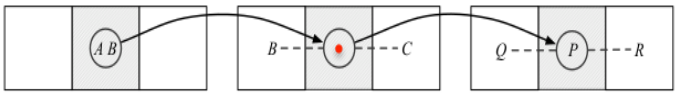
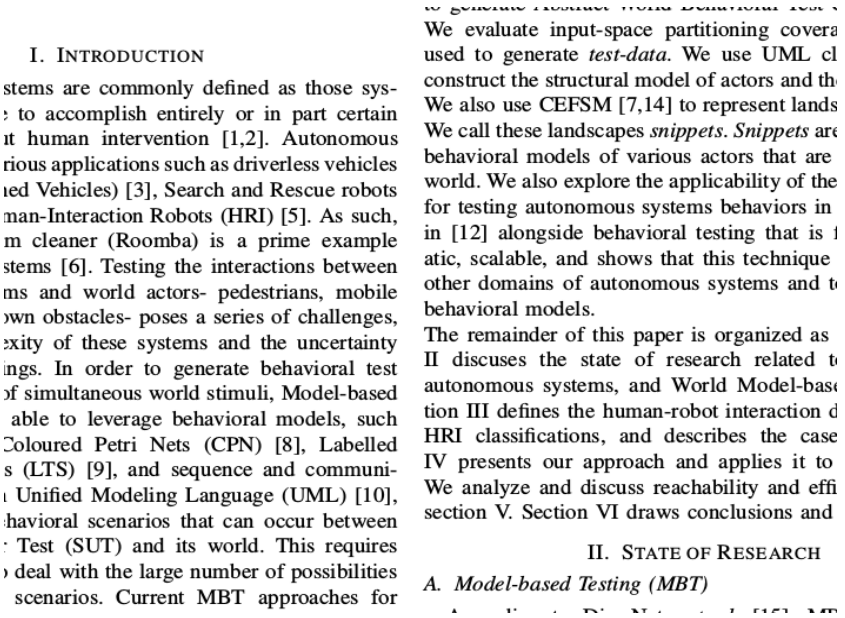


Fig. 5. Graphical representation of sequence $\langle (AB)(\theta \cdot [B; C])(P \cdot [Q; R]) \rangle$

Figura 5.3: In figura vengono mostrate didascalie di figure contenenti caratteri speciali.



I. Introduction

stems are commonly defined as those sys-
: to accomplish entirely or in part certain
ıt human intervention [1,2]. Autonomous
rious applications such as driverless vehicles
ıed Vehicles) [3], Search and Rescue robots
man-Interaction Robots (HRI) [5]. As such,
m cleaner (Roomba) is a prime example
stems [6]. Testing the interactions between
ms and world actors- pedestrians, mobile
own obstacles- poses a series of challenges,
exity of these systems and the uncertainty
ings. In order to generate behavioral test
of simultaneous world stimuli, Model-based
able to leverage behavioral models, such
Coloured Petri Nets (CPN) [8], Labelled
s (LTS) [9], and sequence and communi-
ı Unified Modeling Language (UML) [10],
havioral scenarios that can occur between
· Test (SUT) and its world. This requires
deal with the large number of possibilities
scenarios. Current MBT approaches for

We evaluate input-space partitioning covera
used to generate *test-data*. We use UML cl
construct the structural model of actors and th
We also use CEFSM [7,14] to represent lands
We call these landscapes *snippets*. *Snippets* are
behavioral models of various actors that are
world. We also explore the applicability of the
for testing autonomous systems behaviors in
in [12] alongside behavioral testing that is
atic, scalable, and shows that this technique
other domains of autonomous systems and t
behavioral models.
The remainder of this paper is organized as
II discuses the state of research related t
autonomous systems, and World Model-bas
tion III defines the human-robot interaction d
HRI classifications, and describes the case
IV presents our approach and applies it to
We analyze and discuss reachability and effi
section V. Section VI draws conclusions and

II. State of Research

A. Model-based Testing (MBT)

Figura 5.4: In figura viene mostrato un articolo PDF che utilizza i numeri romani per indicare i capitoli.

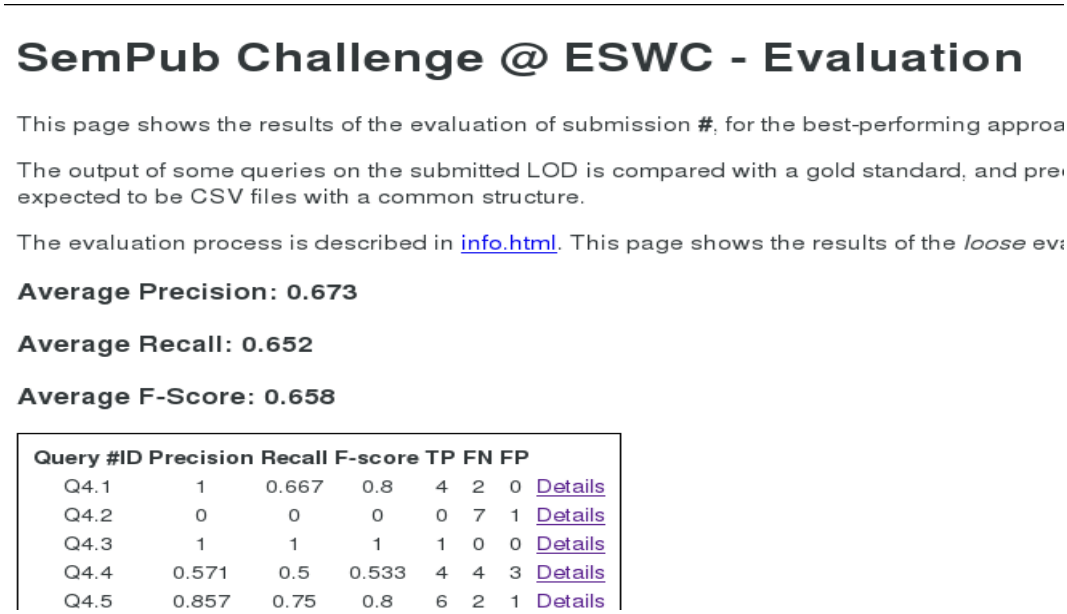
5.3 Valutazione PDF dataset finale

Infine vediamo i risultati ottenuti valutando l'esecuzione di *Investiga* sul dataset finale, cioè l'insieme di 40 articoli PDF assegnato dalla SemPub 2016 per calcolare i risultati finali.

Average Precision	0.673
Average Recall	0.652
Average F-Score	0.658

Tabella 5.3: La tabella riporta i risultati della valutazione sugli articoli PDF del Dataset Finale.

Mostriamo in figura il report relativo alla valutazione di *Investiga* eseguito sugli articoli scientifici in formato PDF del dataset finale:



SemPub Challenge @ ESWC - Evaluation

This page shows the results of the evaluation of submission #, for the best-performing approa

The output of some queries on the submitted LOD is compared with a gold standard, and pre
expected to be CSV files with a common structure.

The evaluation process is described in info.html. This page shows the results of the *loose* ev

Average Precision: 0.673

Average Recall: 0.652

Average F-Score: 0.658

Query #ID	Precision	Recall	F-score	TP	FN	FP	
Q4.1	1	0.667	0.8	4	2	0	Details
Q4.2	0	0	0	0	7	1	Details
Q4.3	1	1	1	1	0	0	Details
Q4.4	0.571	0.5	0.533	4	4	3	Details
Q4.5	0.857	0.75	0.8	6	2	1	Details

Figura 5.5: In figura viene mostrato il report creato dagli script PHP della valutazione di *Investiga* sul dataset finale.

L'ulteriore calo dei risultati è dovuto al fatto che il dataset finale comprende una più ampia varietà di articoli scientifici in formato PDF. I principali errori sono quelli già citati: gli errori descritti nel capitolo precedente, errori nel corretto inserimento degli spazi, presenza di caratteri speciali, la non capacità di *Investiga* di estrarre i capitoli di primo livello nel caso in cui questi inizino con dei numeri romani, il fatto che estragga più dati per le didascalie delle figure rispetto a quelle effettivamente presenti. In più, durante l'esecuzione sugli articoli del dataset finale, in alcuni casi *Investiga* estrae anche delle parti di testo sbagliate durante l'estrazione dei capitoli di primo livello.

Capitolo 6

Discussione e conclusioni

In conclusione si può vedere che riuscire a creare uno strumento per l'estrazione automatica di informazioni da articoli PDF, sia per tutti i tipi sia solo per quelli scientifici, che funzioni correttamente in tutte le casistiche e in tutti i tipi di layout è veramente difficile.

Questo è dovuto soprattutto alla struttura dei documenti in formato PDF, che è molto complessa e molto varia, e nella maggior parte dei casi non ha una struttura logica come frasi o paragrafi. Purtroppo, questo problema rende difficile l'utilizzo di queste informazioni nell'editoria e nelle tecnologie Linked Data rallentandone lo sviluppo.

Per l'implementazione di *Investiga* è stata innanzitutto modificata la libreria PDFMiner, lasciando solo la parte utile per convertire l'articolo PDF in un file XML. Da questo file XML, contenente una riga per ogni carattere dell'articolo originale con indicato font, dimensione e posizione all'interno del testo dell'articolo, sono state estratte le informazioni necessarie all'esecuzione dell'applicazione grazie alle espressioni regolari. Il testo dell'articolo originale viene ricostruito diviso riga per riga e anche pagina per pagina durante un ciclo per ogni riga del file XML, e nel solito ciclo vengono anche riconosciuti ed estratti il titolo e i capitoli di primo livello. Alla fine del ciclo vengono riconosciute le didascalie delle figure e delle tabelle mediante le posizioni delle figure, rilevate durante il ciclo precedente grazie ai tag *figure* presenti all'interno del documento XML, e alle parole chiave "*Fig.*", "*Figure*" e "*Table*" presenti all'inizio delle righe del testo dell'articolo originale ricostruito in precedenza. Infine viene costruito il grafo con-

tenente tutte le informazioni estratte collegate in modo opportuno e, in base alla scelta fatta dall'utilizzatore, o viene eseguita la post di questo grafo su un database RDF ed un URI grafo indicati o viene creato un file.rdf contenente questo grafo nella directory indicata da riga di comando.

Per quanto riguarda i risultati dell'esecuzione d'*Investiga* possiamo concludere che riesce a lavorare con dei buoni punteggi sugli articoli scientifici in formato PDF con layout ACM e LNCS.

Ampliando l'insieme degli articoli scientifici PDF ai non soli layout ACM e LNCS, le prestazioni di questa applicazione calano. Questo perché, principalmente, aumentano i casi che non vengono gestiti correttamente, aumentano i casi in cui gli spazi non vengono inseriti nelle giuste posizioni, per lo più dovuto al fatto che si presentano dei casi che non sono stati gestiti, e vengono estratte delle informazioni aggiuntive errate.

Questo progetto ha diversi margini di sviluppo:

- si potrebbe pensare di creare un processo per ogni articolo presente nella directory di input gestendone l'accesso alle risorse per aumentare le prestazioni dell'esecuzione.

- migliorare l'estrazione dei capitoli di primo livello cercando di coprire una maggior quantità di casi.

- ampliare l'applicazione permettendo l'estrazione di tutti i capitoli e non solo quelli di primo livello.

- migliorare l'estrazione delle didascalie di figure e tabelle cercando di riconoscere il termine di queste ultime anche se sono inserite all'interno del testo dell'articolo.

- riuscire a distinguere anche i casi in cui è presente del testo redatto sotto forma di immagine, per riuscire ad individuarne la didascalia.

- riuscire a mantenere in modo corretto anche i caratteri speciali, anche nel caso in cui questi non siano codificabili in "*UTF-8*".

- gestire la maggior parte di casi possibili per il corretto inserimento degli spazi tra le varie parole.

- riuscire ad ampliare l'insieme dei layout sui quali l'esecuzione di questa applicazione produce dei buoni risultati.

Bibliografia

[1] Pdf liberation — techniques for extracting data from adobe pdfs. `https://pdfliberation.wordpress.com/`. (Accessed on 06/29/2016).

[2] Angelo Di Iorio, Christoph Lange, Anastasia Dimou, and Sahar Vahdati. Semantic publishing challenge - assessing the quality of scientific output by information extraction and interlinking. *CoRR*, abs/1508.06206, 2015.

[3] Bahar Sateli and René Witte. Automatic construction of a semantic knowledge base from ceur workshop proceedings. In *The 12th Extended Semantic Web Conference (The Semantic Publishing Challenge 2015)*, volume 548 of *Semantic Web Evaluation Challenges: SemWebEval 2015 at ESWC 2015, Portorož, Slovenia, May 31 – June 4, 2015, Revised Selected Papers*, page 129–141, Portoroz, Slovenia, 06/2015 2015. Springer, Springer.

[4] Dominika Tkaczyk and Łukasz Bolikowski. *Extracting Contextual Information from Scientific Literature Using CERMINE System*, pages 93–104. Springer International Publishing, Cham, 2015.

[5] Stefan Klampfl and Roman Kern. *Machine Learning Techniques for Automatically Extracting Contextual Information from Scientific Publications*, pages 105–116. Springer International Publishing, Cham, 2015.

[6] Francesco Ronzano, Beatriz Fisas, Gerard Casamayor del Bosque, and Horacio Saggion. *On the Automated Generation of Scholarly Publishing Linked Datasets: The Case of CEUR-WS Proceedings*, pages 177–188. Springer International Publishing, Cham, 2015.

[7] Andrea Giovanni Nuzzolese, Silvio Peroni, and Diego Reforgiato Recupero. *MACJa: Metadata and Citations Jailbreaker*, pages 117–128. Springer International Publishing, Cham, 2015.

[8] Liubov Kovriguina, Alexander Shipilo, Fedor Kozlov, Maxim Kolchin, and Eugene Cherny. *Metadata Extraction from Conference Proceedings Using Template-Based Approach*, pages 153–164. Springer International Publishing, Cham, 2015.

[9] Github - euske/pdfminer: Python pdf parser. `https://github.com/euske/pdfminer`. (Accessed on 06/16/2016).

[10] Programming with pdfminer. `http://www.unixuser.org/~euske/python/pdfminer/programming.html`. (Accessed on 06/16/2016).

[11] Pdfminer. `http://www.unixuser.org/~euske/python/pdfminer/`. (Accessed on 06/17/2016).

[12] Manipulating pdfs with python - tutorial - binpress. `https://www.binpress.com/tutorial/manipulating-pdfs-with-python/167`. (Accessed on 06/17/2016).

[13] Github - dpapathanasiou/pdfminer-layout-scanner: A more complete example of programming with pdfminer, which continues where the default documentation stops. `https://github.com/dpapathanasiou/pdfminer-layout-scanner`. (Accessed on 06/17/2016).

[14] File con estensione pnm — come aprire un file .pnm. `https://www.filesuffix.com/it/extension/pnm`. (Accessed on 06/17/2016).

[15] The pnm format. `http://netpbm.sourceforge.net/doc/pnm.html#index`. (Accessed on 06/17/2016).

[16] Introducing tabula - features - source: An opennews project. `https://source.opennews.org/en-US/articles/introducing-tabula/`. (Accessed on 06/18/2016).

[17] Tabula: Extract tables from pdfs. `http://tabula.technology/`. (Accessed on 06/18/2016).

[18] Extracting data from pdfs using tabula — school of data - evidence is power. http://schoolofdata.org/extracting-data-from-pdfs/. (Accessed on 06/18/2016).

[19] Using the command line tabula extractor tool · tabulapdf/tabula-java wiki · github. https://github.com/tabulapdf/tabula-java/wiki/Using-the-command-line-tabula-extractor-tool. (Accessed on 06/18/2016).

[20] Apache pdfbox — a java pdf library. https://pdfbox.apache.org/index.html. (Accessed on 06/23/2016).

[21] Github - apache/pdfbox: Mirror of apache pdfbox. https://github.com/apache/pdfbox. (Accessed on 06/23/2016).

[22] Dominika Tkaczyk, Paweł Szostek, Mateusz Fedoryszak, Piotr Jan Dendek, and Łukasz Bolikowski. Cermine: automatic extraction of structured metadata from scientific literature. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(4):317–335, 2015.

[23] Ceur-ws.org - ceur workshop proceedings. http://ceur-ws.org/. (Accessed on 06/29/2016).

[24] What is grab::spider? — grab 0.6 documentation. http://docs.grablib.org/en/latest/spider/intro.html. (Accessed on 06/29/2016).

[25] Sparqlwrapper 1.7.6. https://rdflib.github.io/sparqlwrapper/doc/latest/. (Accessed on 06/21/2016).

Ringraziamenti

Vorrei ringraziare la mia famiglia che mi ha sempre supportato e ha sempre creduto in me, la mia ragazza e i miei amici che mi sono sempre stati vicini e il mio relatore, il dottor Angelo Di Iorio per la sua gentilezza e disponibilità.