Strukturierte Extraktion von Text aus PDF

Präsentation der Masterarbeit von

Fabian Schillinger

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Übersicht

UNI FREIBURG

- Motivation
- Probleme bei der Textextraktion
- Ablauf des entwickelten Systems
- Ergebnisse
- Präsentation an einem Beispiel

Motivation

UNI FREIBURG

Motivation

- Verwendung von PDF
  - Prospekte
  - Zeitungen
  - Bücher
  - Wissenschaftliche Artikel
  - ...

Motivation

UNI FREIBURG

- Extraktion des Text für
  - Vorlesesysteme
  - „Reflowable“ Anwendungen
  - Bessere Darstellung
  - Semantische Suche

Probleme bei der Textextraktion

UNI FREIBURG

Probleme bei der Textextraktion

- Layouts mit mehreren Spalten

In this paper we explore the use of CSD for OIE. In fact, the contexts above already look close to the kind of triples expected from an OIE system. All that is missing is the distinction into the subject, predicate, and object part. Since CSD is computed from a full parse of the sentence, with explicit markup denoting the verb phrases, this is relatively straightforward. Also note that some of the contexts above are missing a verb. In that case the verb is implicit, but can (typically) be easily deduced from the context, e.g. *is* for contexts #3 and #4 (noun phrase with pre-modifying noun phrase). Our system, called CSD-IE is described in detail in Section III.

A. Quality Aspects

In this paper, we evaluate OIE systems with respect to the following three quality aspects.

1. The accuracy of the extracted facts. Two aspects are important here. First, whether the fact actually has the form of a meaningful relational triple. For example, the triple *(Ruth) (Gabriel was) (San Fernando)* would be considered inaccurate for two reasons: (1) the P part contains words which do not belong to the verb (but to the S part in this

another explicit goal of our system to extract minimal facts. The four facts listed under 2. above are all minimal.

The goal of minimality comes with a challenge that is not apparent in our example sentences, but occurs in sentences of a more complex type. For example, consider the sentence:

The Embassy said that 6,700 Americans were in Pakistan.

This sentence contains two facts: that the Embassy made some statement, and that 6,700 Americans were in Pakistan. However, by simply extracting these two facts, we would lose the information what statement the Embassy made. We solve this by allowing the S and O part of a fact to contain *references* to other extracted facts. In this case, we would extract:

#1: *(The Embassy) (said) (that #2)*
#2: *(6,700 Americans) (were) (in Pakistan.)*

where the numbers are simply unique ids for each extracted triple. That way, no information is lost, and the application may choose to either keep the facts separate (and avoid mixing of facts) or substitute the references with the referred fact (and thus obtain output as in other OIE systems). A

Probleme bei der Textextraktion

- Kopf- und Fußzeilen

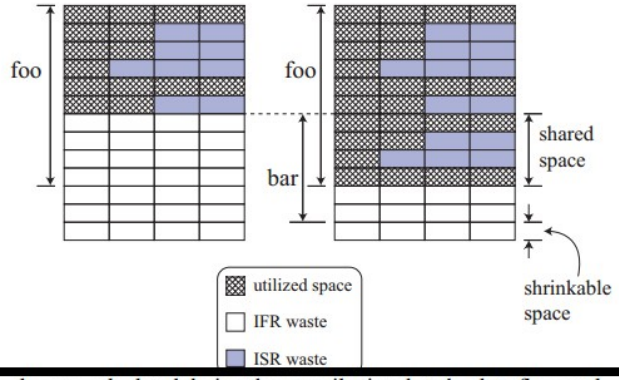
list of patterns of length $\leq k$ by traversing the bytecode exactly once. When a valid pattern i of any length is encountered the first time, it is added to the hash map with $\zeta_i = 1$. Then, ζ_i is incremented whenever the same pattern i

---

is found again while traversing the remaining bytecode. Consequently, after a single traversal of the Java bytecode, the hash map contains all possible patterns of length $\leq k$ with their corresponding frequencies. The algorithm returns a subset σ of patterns from within the hash map such that $|\sigma| \leq m$ and $\xi_{(\sigma)}$ is

Probleme bei der Textextraktion

- Tabellen, Grafiken und Algorithmen
- Bildunterschriften




```
1   public static void foobar(...) {
2       if (...) {
3           foobar(...);
4       }
5       ....
6   }
```

Figure 4. Offline data-flow analysis limitation in the presence of a recursive function.

also pre-calculated during the compile time but the data-flow analysis to compute this size is inter-procedural and field-sensitive.

CVCS allows a single chunk to carry multiple frames leading to less frequent runtime memory allocations, which results in a reduction in FAT waste and, therefore, an increase in the lifetime of power-constrained devices. As our results indicate (Section 4), the steady state memory utilization using CVCS remains lower than

In the above equation, F_f is the total size of a frame of the function f. $F_{f,i}$ is the maximum size of function f's frame at instruction i and $g(f, i)$ is the function g invoked by the instruction i of the function f.

3

Probleme bei der Textextraktion

- Fußnoten

made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
JIWES '12 August 12 2012, Portland, OR, USA
Copyright 2012 ACM 978-1-4503-1601-9/12/08 ...$15.00.

facts), where each element of each triple has an identifier that is consistent among triples. For example, *Broccoli is-*

[1]Let us ignore the untypical case that a precompiled document containing those words and the result list exist.
[2]`semsearch.yahoo.com`

a Vegetable or *Vegetable is-subclass-of Plant* or *Broccoli is-native-to Europe.*

ing different names meaning the same thing via user-created links (*owl:sameAs*). This works well for popular relations

Probleme bei der Textextraktion

- Fußnotenzeichen

event at a particular station. For example, a node might stand for the event of ICE 500 arriving at Mannheim Hauptbahnhof at 21:24.[2] Arcs between nodes then either correspond to waiting from one event to the next at a particular

- Quellenverweise

potentially apply on various classes of documents in various languages. This approach is similar to some previous works [2] [3] and shares with them a geometric approach issued from the XY-cut page segmentation method. The use of of-the-shelf PDF

Ablauf

Ablauf


Vorbereitung des Textes
Sequentielle Ausführung aller Analysewerkzeuge
Ausgabe des Textes

Ablauf

- Vorbereiten des Textes
  - Extraktion der Buchstaben
    - Mit Eigenschaften wie Schriftart, -Farbe, -Größe
    - Seite
    - Mögliche Zugehörigkeit zu einem Wort (PDFBox)
  - Zusammensetzen der Worte basierend auf
    - Mögliche Zugehörigkeit zu einem Wort (PDFBox)
    - Gleiche Schrifteigenschaften
    - Passende Koordinaten
  - Bereitstellung der Datenstrukturen

Ablauf

- makeGrid()
  - Ziel: Zusammenfassen von Textblöcken
  - Ablauf:
    - Dehne Wörter aus
    - Teile Seite rekursiv
      - makeHorizontalLines()
      - makeVerticalLines()

Ablauf

Ablauf

- makeHorizontalLines()
  - Ziel: Teile Block (bzw. Seite) horizontal
  - Ablauf:
    - Wähle horizontale Linie
    - Prüfe für jedes Wort, ob es oberhalb oder unterhalb der Linie liegt
    - Wenn kein Wort auf der Linie liegt speichere Worte oberhalb in Liste
    - Wähle weitere horizontale Linie
    - …
    - Gebe HashMap aus, die Listen beinhaltet

Ablauf

- makeVerticalLines()
  - Ziel: Teile Block (bzw. Seite) vertikal
  - Ablauf:
    - Wähle vertikale Linie in der Mitte
    - Prüfe für jedes Wort, ob es links oder rechts der Linie liegt
    - Wenn kein Wort auf der Linie liegt speichere Worte in Listen
    - Sonst wähle vertikale Linie rechts/links der Mitte
    - …
    - Gebe HashMap aus, die Listen beinhaltet

Ablauf

- calculateUppercaseAndNumbers()
  - Ziel: Finde Tabellen oder Grafiken anhand der Anzahl an Großbuchstaben und Zahlen
  - Ablauf:
    - Zähle Großbuchstaben und Zahlen aller Blöcke
    - Berechne Durchschnittliche Anzahl pro Block
    - Entferne Blöcke, die mehr als die 1,2-fache Anzahl beinhalten
      - Markiere dazu jedes Wort des Blocks als nicht interessant

Ablauf

- findCaptions()
  - Ziel: finde Bildunterschriften
  - Ablauf:
    - Durchsuche alle Blöcke nach Schlüsselworten „Table“, „Figure“, „Algorithm“
    - Suche Doppelpunkt rechts neben Schlüsselwort

Ablauf

- findSmallerBlocks()
  - Ziel: Finde Kopfzeilen, Fußzeilen, Grafiken, etc. die sich vom normalen Layout unterscheiden
  - Ablauf:
    - makeGrid() um kleinere Blöcke zu finden
    - Speichere Worte basierend auf y-Koordinate in HashMap
    - Berechne für jede y-Koordinate die Breite
    - Speichere Durchschnittliche Breite für jeden Block in Liste
    - Entferne Blöcke, deren durchschnittliche Breite
      - größer als 150% ist
      - kleiner als 90% ist

Ablauf

- Zwischenstand
  - Layouts mit mehreren Spalten
  - ~~Kopf- und Fußzeilen~~
  - ~~Tabellen, Grafiken und Algorithmen~~
  - ~~Bildunterschriften~~
  - ~~Fußnoten~~ und Fußnotenzeichen
  - Quellenverweise

Ablauf

- findReferences()
  - Ziel: Finde Quellenangaben
  - Ablauf:
    - Durchsuche alle Blöcke, beginnend auf der letzten Seite nach Schlüsselwort „References“
    - Speichere Schrifteigenschaften des Schlüsselwortes
    - Entferne alle Blöcke nach dem Schlüsselwort

Ablauf

- Finde Titel und Autorenangaben
  - Teile erste Seite in 3 horizontale Blöcke
- Finde Abstract
  - Durchsuche erste Seite nach Schlüsselwort „Abstract“
  - Speichere Schrifteigenschaften des Schlüsselwortes ab

Ablauf

- makeOnePage()
  - Ziel: Füge alle Blöcke in Lesereihenfolge hintereinander
  - Ablauf:
    - Teile jede Seite vertikal auf (wenn möglich)
      - makeVerticalLines()
    - Hänge linken Block an gemeinsame Liste an
    - Hänge rechten Block an gemeinsame Liste an

Ablauf

- removeFootnoteSign()
  - Ziel: entferne Anmerkungsziffer
  - Ablauf:
    - Speichere jedes Wort basierend auf y-Koordinate in HashMap
    - Prüfe für jede y-Koordinate, ob höchstens 5 Worte zugeordnet sind
    - Wenn ja, prüfe, ob jedes dieser Worte aus höchstens 3 Ziffern besteht
    - Entferne diese Worte aus der Liste

Ablauf

- Zwischenstand
  - ~~Layouts mit mehreren Spalten~~
  - ~~Kopf- und Fußzeilen~~
  - ~~Tabellen, Grafiken und Algorithmen~~
  - ~~Bildunterschriften~~
  - ~~Fußnoten und Fußnotenzeichen~~
  - Quellenverweise

    → Regex

Ablauf

- Ausgabe des Textes als XML-Datei
  - Titel des Artikels
  - Autorenangaben
  - Abstract
  - Alle Kapitel mit Teilüberschriften
    - Vergleiche Schrifteigenschaften aller Worte in der gemeinsamen Liste von makeOnePage() mit gespeicherten Werten von „Abstract“ und „References“
      - Wort ist Teil einer Überschrift oder Teil des Inhalts

Ablauf

UNI FREIBURG

```
<?xml version="1.0"?>
<paper>
  <title>An Index for Efficient Semantic Full-Text Search</title>
  <authors>Hannah Bast, Björn Buchhold Department of Computer Science University of Freiburg 79110 Freiburg, Germany {bast, buchhold}@informatik.uni-freiburg.de</authors>
  <abstract>ABSTRACT In this paper we present a novel index data structure tailored...</abstract>
  <chapter>
    <header>1. INTRODUCTION </header>
    <content>Classic full-text search...</content>
  </chapter>
</paper>
```

Ergebnisse

Ergebnisse

- Bewertung der Klassifizierung
  - 2 Bilddateien mit eingefärbten Worten
  - Färbe Bilddateien von Hand
  - Lese Dateien ein und vergleiche Ergebnisse

- Analyse der Laufzeit
  - Speichere Zeitpunkte zwischen den Werkzeugen
  - Schreibe Differenzen in Datei

Ergebnisse

Ergebnisse

- Wissenschaftliche Artikel vom Lehrstuhl für Algorithmen und Datenstrukturen
  - 29 Artikel
  - Mehr als 20 verschiedene Zeitschriften, Workshops, Konferenzen, etc.
  - 364 Seiten / 478.659 Worte

Ergebnisse

- 478.659 Worte

  424.826 korrekt interessant eingeordnet

  4.286 falsch interessant eingeordnet

  39.811 korrekt uninteressant eingeordnet

  9.740 falsch uninteressant eingeordnet

- Precision = 0,990
- Recall = 0,978
- F_1 = 0,983

Ergebnisse

- PloS Biology Journal
  - 8 Artikel
  - Aus 4 Auflagen mit ähnlichen Layouts
  - 107 Seiten / 174.936 Worte

Ergebnisse

- 174.936 Worte
- 129.460 korrekt interessant eingeordnet
- 5.014 falsch interessant eingeordnet
- 32.425 korrekt uninteressant eingeordnet
- 8.036 falsch uninteressant eingeordnet

- Precision = 0,963
- Recall = 0,942
- F_1 = 0,952

Ergebnisse

UNI FREIBURG

- Laufzeit
  - 37 Dokumente
  - 471 Seiten / 653.595 Worte

- 107,7 Sekunden – Insgesamt (~230 ms pro Seite)
- 21,3 Sekunden – Blöcke finden
- 4,9 Sekunden – Analyse
- 11,2 Sekunden – Ausgabe der XML-Datei
- 68,4 Sekunden – Extraktion der Buchstaben

Präsentation des entwickelten Systems

UNI FREIBURG