Exstrakcija podatkov s spletnih strani

Poročilo seminarske naloge

Adam Prestor, Lojze Žust
ap2408@student.uni-lj.si, lojze.zust@student.uni-lj.si

Fakulteta za računalništvo in informatiko, Univerza v Ljubljani

1 Uvod

Avtomatizirana ekstrakcija podatkov s spleta omogoča izločanje strukturiranih podatkov iz strani, ki v HTML obliki prikazujejo podatke iz baze. V tej seminarski nalogi smo se tega procesa lotili s tremi različnimi pristopi. Pri pristopih z regularnimi izrazi in XPath potmmi določimo specifične iskalne nize za posamezen tip strani, razvili pa smo tudi algoritem, ki iz para strani istega tipa skuša avtomatsko detektirati podatkovna polja.

2 Dodatna spletna stran

Slika 1: Primer spletne strani themoviedb.org. Označena so polja, ki jih želimo ekstrahirati.

V nalogi sta vključeni spletni strani *rtvslo.si* in *overstock.com*, vključili pa smo še stran *themoviedb.org*. Gre za spletno bazo filmov in tv serij. S strani smo prenesli HTML vsebino strani za dva filma. Primer in označena podatkovna

polja so vidna na Sliki 1. Nekatera izmed podatkovnih polj (Genres in People) so seznami. Vsak element polja People je zgrajen iz dveh polj (ime osebe in njena vloga).

Za namen testiranja delovanja avtomatske ekstrakcije smo dodali še četrto stran, test, ki ima zelo preprosto HTML strukturo in vsebuje sezname in opcijske elemente.

3 Metodologija

3.1 Regularni izrazi

Z regularnimi izrazi definiramo vzorce, s katerim potem poiščemo ustrezna polja v HTML kodi strani. En regularni izraz omogoča tudi ujemanje večih podatkovnih polj naenkrat.

rtvslo.si Pri polju Content smo zajeli celotno HTML vsebino znotraj okvirja označenega na shemi v navodilih (vključena je tudi slika). HTML vsebino smo očistili, da ne vsebuje odvečnih skript in slogov. Uporabljeni regularni izrazi:

```
Title: '<h1>(.+)</h1>'
SubTitle: '<div class="subtitle">(.+)</div>'
Lead: '<p class="lead">(.+)</p>'
Content: '<div class="article-body">\s*([\s\S]+?\s*</article>)'
Author: '<div class="author-name">(.+)</div>'
PublishedTime: '<div class="publish-meta">\s*(.*)\s*<br>'
```

overstock.com V nasprotju z rtvslo.si, je ta stran zelo skromno opremljena s semantičnimi podatki v atributih HTML značk, zato je ekstrakcija z regularnimi izrazi precej težja. Opirali smo se zato bolj na nekatera fiksna besedila na strani in vsebino povezav. Polji Saving in SavingPercent je bilo najlažje zajeti z enim regularnim izrazom.

```
Title: '<a.+PAGE=PROFRAME.+?PROD_ID=\d+[^<]*><b>(.+)</b>'
Content: '<span class="normal">([\S\s]+?)\s*<br>'
ListPrice: 'List Price:.+<s>(.+)</s>'
Price: 'Price:.+<b>(.+)</b>'
Saving & SavingPercent & : 'You Save:.+(\$.+?)\s*\((.+)\)'
```

themoviedb.org Polji Genres in People sta seznama, pri tem regularni izraz za People ulovi ime in vlogo hkrati.

```
Title: '<div class="title[\S\s]+?>(.+)</a>'
Year: '<span class="tag release_date">\((\d+)\)</span>'
Certification: '<span class="certification">\s*(.+)\s*</span>'
Release date: '<span class="release">\s*(.+)\s*</span>'
```

```
Genres: '<a href="/genre/.*?/movie">\s*(.*?)\s*</a>'
Runtime: '<span class="runtime">\s*(.*?)\s*</span>'
Rating: 'data-percent="(.*?)"'
Tagline: '<h3 class="tagline" dir="auto">\s*(.*?)\s*</h3>'
Overview: '<div class="overview" dir="auto">\s*<p>\s*(.*)\s*</p>\s*</div>'
People: '<li class="profile">[\s\S]*?<a href="/person/.*>(.*)
         </a>[\s\S]*?<p class="character">(.*)</p>'
```

3.2 XPath izrazi

Definicija ustreznih XPath izrazov je bila v večini primerov precej lažja kot pri regularnih izrazih, saj omogočajo enostavno iskanje po HTML drevesni strukturi. V primerih kjer so bili podatki znotraj ene same HTML značke pa smo uporabili še dodatne regularne izraze za posamezna polja. Na vseh tekstovnih poljih uporabimo funkcijo strip(), ki odstrani prazne znake na začetku in koncu besedila.

rtvslo.si Pri polju Content smo izpisali očiščeno vsebino najdenega vozlišča (inner HTML).

```
Title: '//h1/text()'
SubTitle: '//div[@class="subtitle"]/text()'
Lead: '//p[@class="lead"]/text()'
Content: '//div[@class="article-body"]'
Author: '//div[@class="author-name"]/text()'
PublishedTime: '//div[@class="publish-meta"]/text()'
```

overstock.com Polji Saving in SavingPercent smo dobili z uporabo regularnega izraza nad dobljenim tekstom.

```
Title: '//td[@valign="top"]/a/b/text()'
Content: '//span[@class="normal"]/text()'
ListPrice: '//td/s/text()'
Price: '//span[@class="bigred"]/b/text()'

Saving & SavingPercent: '//td[@align="left"]/span[@class="littleorange"]/text()'
Saving & SavingPercent regex: '(\$.+?)\s*\((.+)\)'
```

themoviedb.org Tokrat smo polja za ime in vlogo posamezne osebe dobili z ločenimi poizvedbami. Vsebina polja Rating se ne nahaja v besedilu strani (uporablja se canvas), ampak v atributu `data-percent`, zato smo uporabili naslavljanje atributov v XPath.

```
Title: '//h2/a/text()'
Year: '//span[@class="tag release_date"]/text()'
```

```
Certification: '//span[@class="certification"]/text()'
Release date: '//span[@class="release"]/text()'
Genres: '//span[@class="genres"]/a/text()'
Runtime: '//span[@class="runtime"]/text()'
Rating: '//div[@class="user_score_chart"]/@data-percent'
Tagline: '//h3[@class="tagline"]/text()'
Overview: '//div[@class="overview"]/p/text()'

People (name): '//li[@class="profile"]/p/a/text()'
People (role): '//li[@class="profile"]/p[@class="character"]/text()'
```

3.3 Avtomatska ekstrakcija

Inspiracija Začetna inspiracija našega algoritma je bila metoda RoadRunner [1]. Slednjega smo se poskusili lotiti prek drevesne strukture html-ja strani. Tako smo se veliko naslanjali na metode poravnave in ujemanja dreves [2] prek katerih smo iskali elemente z različno vsebino in pogojnimi elementi. Predvsem smo se naslonili na metodi preprostega ujemanja dreves *(angl. Simple Tree Matching)* z dodanim sledenjem *(angl. Backtracking)* in iskanjem podatkovnih regij *(angl. Mining Data Regions)*.

Opis Algoritma Naš algoritem v začetku zgradi html drevo iz vsebine prve in druge strani. Te dve drevesi potem poskušamo čim boljše poravnati med seboj. Tu nastopi algoritem preprostega ujemanja dreves, ki, če ga implementiramo z dodanim sledenjem, vrne najboljšo poravnavo obeh dreves. Slednji del je opisan s psevdo kodo 1. Dobljeno poravnano drevo ima v vsakem listu poravnana html elementa, razen pri tistih elementih, ki jih nismo uspeli poravnati. To drevo potem poskusimo posplošiti.

Posplošitev naredimo tako, da za v vsakem vozlišču združimo pripadajoča html elementa in si zapomnemo, ali je šlo za opcijski element in tekstovno razliko. Ko to naredimo, se sprehodimo čez sinove in za njih naredimo enako, ter jih dodamo v sinove tega drevesa, kot je prikazano s pseudo kodo 2.

Ko dobimo vse sinove pa poskušamo iz njih pridobiti ponavljajoča vozlišča. Za ta del smo vzeli motivacijo iz algoritma iskanje podatkovnih regij. Algoritem deluje tako, da primerja zaporednja sinova. Začnemo pri prvem sinu in ga primerjamo z naslednjim. Obe vozlišči primerjamo med seboj.

Primerjava poteka tako, da poskušamo obe drevesi čimbolje poravnati, potem preštejemo število ustrezno poravnanih elementov in jo delimo s številom neopcijskih elementov največjega izmed dreves. Če je podobnost dovolj visoka, višja od nastavljene vrednosti *tau*, potem vozlišči smatramo kot podobni ter njuni drevesi združimo in ga označimo kot ponavljajoči element. V naslednjem koraku primerjamo združeno vozlišče z naslednjim sinom.

Če pa si vozlišči nista dovolj podobni, potem preidemo na naslednjega sina in nadaljujemo s primerjavam. Celoten algoritem je opisan s pseudo kodo 3.

Tako v končnem rezultatu dobimo generalizirano drevo, ki ima v vsakem vozlišču zapisano oznako poleg tega pa je označeno, če je tam prišlo do spremembe besedila, ali je element opcijski in ali gre za ponavljajoči element.

Algorithm 1 Overview of the algorithm.

1: $htmlTree1 \leftarrow BuildHtmlTree(html1)$
2: $htmlTree2 \leftarrow buildHtmlTree(html2)$
3: $w, AlignedTree \leftarrow SimpleTreeMatching(htmlTree1, htmlTree2)$
4: $gt \leftarrow GenerelizeTree(gt)$

Algorithm 2 GenerelizeTree(AlignedTreeNode)

1: $CombineAlignedNodes$
2: **for all** children **do**
3: $AddChild(GenerelizeTree(child))$
4: **end for**
5: MDR(0)

Algorithm 3 MDR(index, tau)

1: **if** $index < NumberOfChildren$ **then**
2: $c1 \leftarrow child[index]$
3: $c2 \leftarrow child[index + 1]$
4: $similarity, tree \leftarrow CompareNodes(c1, c2)$
5: **if** $sim > tau$ **then**
6: $Replace([c1, c2], tree)$
7: $MDR(index, tau)$
8: **else**
9: $MDR(index + 1, tau)$
10: **end if**
11: **end if**

Izhod algoritma Algoritem proizvede generalizirano drevo. To strukturo lahko predstavimo na več načinov. Najpreprostejši način je, da drevo shranimo v strukturo XML. Generalizirana vozlišča, ki predstavljajo tekst, opcijske elemente ali seznam označimo s posebnimi atributi. Tak XML dokument lahko potem za ekstrakcijo uporabimo na več načinov. Lahko ga enostavno pretvorimo v regularni izraz, tako da nadomestimo generalizirana vozlišča z ustreznimi konstrukti za

opcijske in ponavljajoče elemente. Drug način je, da za generalizirana vozlišča določimo ustrezne XPath poizvedbe.

V okviru seminarske smo implementirali dva tipa ovojnice – generaliziran XML in XPath poizvedbe. V osnovni konfiguraciji se izvozi XML ovojnica, generalizirano drevo pa omogoča tudi izvoz XPath poti. Celotna XML ovojnica pride za obravnavane strani precej velika, zato smo se odločili, da v poročilo vključimo kratek izsek zanimivih delov. Preostanek ovojnic je na voljo v repozitoriju. Izjema je testni primer, ki je dovolj majhen, da ga lahko prikažemo. Za testni primer podamo ovojnico tudi v obliki XPath poti.

Testni primer Za testiranje algoritma smo naredili dve preprosti testni strani, ki vsebujeta tekstonve elmente, sezname in opcijske elemente. Izhoda algoritma je naslednja ovojnica.

```
<html>
  <body>
    <div>
      <div list="list">
        <ul>
          <li text="text" list="list"/>
        </ul>
      </div>
      <div optional="optional">
        <p text="text"/>
      </div>
      <div>
        <h1/>
      </div>
      <p text="text"/>
    </div>
  </body>
</html>
```

XPath poti generaliziranih elementov pa so take. Znaki pred XPath poizvedbo označujejo tip vozlišča. L predstavlja seznam, T predstavlja tekst, O pa predstavlja opcijski element.

```
L  : //html/body/div/div
LT : //html/body/div/div/ul/li
O  : //html/body/div/div
T  : //html/body/div/div/p
T  : //html/body/div/p
```

rtvslo.si

```
[...]
<article>
    <figure optional="optional">
      <a>
        <img/>
      </a>
      <figcaption>
        <span/>
      </figcaption>
    </figure>
    <figure>
      <div text="text">
        <a optional="optional">
          <img/>
        </a>
      </div>
      <div text="text">
        <img optional="optional"/>
        <div optional="optional">
          <a>
            <img/>
          </a>
        </div>
        <div optional="optional">
          <img/>
        </div>
      </div>
    </figure>
[...]
```

overstock.com

```
[...]
<td>
  <br/>
  <table>
    <tbody>
      <tr>
        <td>
          <table>
            <tbody>
              <tr list="list">
                <td text="text">
                  <span>
                    <b text="text"/>
                  </span>
```

```
                </td>
                <td text="text">
                  <b text="text" list="list"/>
                </td>
              </tr>
            </tbody>
          </table>
        </td>
      </tr>
    </tbody>
  </table>
</td>
[...]
```

themoviedb.org

```
[...]
<div>
    <section>
      <div>
        <h2>
          <a text="text" list="list"/>
        </h2>
        <div>
          <span text="text" list="list"/>
          <span>
            <a text="text" list="list"/>
          </span>
          <span text="text"/>
        </div>
      </div>
      <ul>
        <li>
          <div>
            <div>
              <div>
                <div>
                  <span/>
                </div>
              </div>
            </div>
          </div>
          <div>
[...]
```

4 Zaključek

Pripravili smo regularne izraze in XPath poizvedbe za izločanje podatkov iz izbranih strani. Prišli smo do spoznanja, da je s pomočjo xpath izrazov v večini primorv lažje generirati stavke za pridobitev informacije iz html strukture. Tudi celoten izraz izgleda bolj pregleden in robusten.

Razvili smo tudi algoritem za avtomatsko generacijo ovojnice, ki je sposoben zaznati generalizirana vozlišča, ki predstavljajo besedilo, opcijske elemente in tudi sezname.

Možna izboljšava avtomatskega generiranja ovojnica bi bila boljše iskanje ponavljajočih delov 3. Preprosti možnosti sta eksperimentiranje z omejitvijo *tau* oziroma drugačno utežitev podobnosti, ki upošteva tudi globino poleg velikost poddreves. Možna izboljšava bi bila tudi implementacija bolj sofisticiranega poravnavanja dreves, kljub temu da se je preprosto ujemanje dreves s sledenjem izkazalo za natančno in robustno.

Prav tako bi lahko ob poravnavanju html elementov upoštevali tudi njihove atribute, kar bi verjetno pripeljalo do bolj robustnih rešitev.

Izboljšati bi se dalo tudi pripravo in izpis ovojnice. Sedaj se v ovojico zajame celotna struktura drevesa. Če kakšno izmed vozlišč nima nobenega generaliziranega naslednika, potem njegovih naslednjikov ne bi potrebovali izpisovati v ovojnico. Za nepomemnbe dele bi tako ohranili zgolj najvišje vozlišče.

Literatura

1. Crescenzi, V., Mecca, G., Merialdo, P., et al. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB* (2001), vol. 1, pp. 109–118.
2. Zhai, Y., and Liu, B. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web* (2005), pp. 76–85.