Seminarska naloga 2 - Ekstrakcija podatkov

Katjuša Jaklič, Luka Kuzman, Tjaša Mlakar
Fakulteta za računalništvo in informatiko, Univerza v Ljubljani
Maj 2022

1. Uvod

Sistemi za ekstrakcijo podatkov s spleta so orodja, ki samodejno izvlečejo podatke iz spletnih strani s spreminjajočo se vsebino ter le-te shranijo v podatkovno bazo ali katero koli drugo strukturirano obliko zapisa. Cilj druge seminarske naloge je bila implementacija treh različnih pristopov za pridobivanje strukturiranih podatkov s spleta. Seminarske naloge se lotimo na tri načine:

1) s pomočjo regularnih izrazov
2) s pomočjo XPath
3) avtomatska ekstrakcija podatkov s pomočjo algoritma RoadRunner

Omenimo, da gre pri zadnji le za implementacijo ustvarjanja ovoja, na podlagi katerega lahko nadalje implementiramo ekstraktor.

2. Implementacija

Za implementacijo rešitev smo uporabili programski jezik Python. Ekstrakcijo podatkov smo izvajali na dveh člankih iz rtvslo.si ter na dveh seznamih izdelkov iz overstock.com. Dodatno pa smo si izbrali še spletno stran mimovrste.com, kjer smo si izbrali dva izdelka, iz njunih strani pa ekstrahirali podatke o imenu izdelka (Title), ceni (Price), številki izdelka (Number), zadovoljstvu kupcev (RatingPercent, v %) ter o številu podanih mnenj (ReviewNumber). Izbrani podatki so označeni na sliki 1.

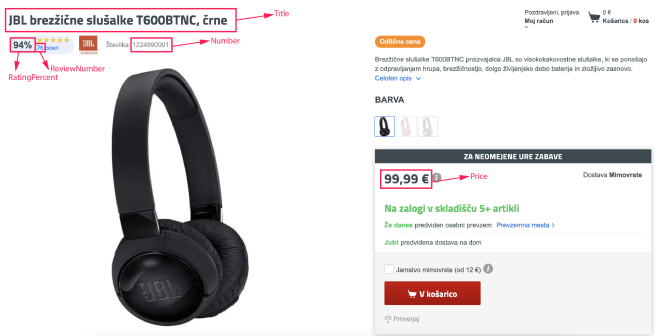

Figure 1: Iskani podatki na mimovrste.com

2.1. Implementacija s pomočjo regularnih izrazov

V prvem delu smo se ekstrakcije podatkov lotili z uporabo regularnih izrazov, ki smo jih prilagodili za vsak par strani ločeno. Znotraj Python datoteke regex_extraction.py smo implementirali 3 funkcije, kjer vsaka poskrbi za ekstrakcijo podatkov iz posameznega para spletnih strani. Vsaki izmed teh 3 funkcij je podana html datoteka, nad katero se nato izvajajo regularni izrazi, ki iščejo podatke. Pri tem smo uporabili knjižnico re, s katero smo v html vsebini našli dotične podatke. Določeno prejeto vsebino je bilo potrebno še dodatno urediti z operacijami, in sicer, da smo se znebili odvečnih html značk in morebitnih nepotrebnih podatkov. Urejene podatke smo nato pretvorili v JSON obliko, ki jih naredi bolj berljive in pregledne. Ob klicu funkcije, da se izvrši uporaba regularnih izrazov, se zaženejo vse 3 funkcije in podatki se nato izpišejo na standardnem izhodu.

Regularni izrazi za ekstrakcijo podatkov iz strani overstock.com so prikazani na sliki 2.

```
title = """<td\s+valign=\"top\">\s+<a\s+href=\"\S*\"><b>(.*)</b>"""
listPrice =  """<b>List\sPrice:<\/b><\/td><td\salign=\"left\"\s+nowrap=\"nowrap\"><s>(.*)</s></td>"""
price = """<td\salign=\"left\"\snowrap=\"nowrap\"><span\sclass=\"bigred\"><b>(.*)</b></span>"""
saving = """<td\salign=\"left\"\snowrap=\"nowrap\"><span\sclass=\"littleorange\">(\$[0-9\.,]*).*</span></td>"""
savingPercent = """<td\salign=\"left\"\snowrap=\"nowrap\"><span\sclass=\"littleorange\">.*\((.*)\)</span></td>"""
content = """<td\svalign=\"top\"><span\s+class=\"normal\">([\s\S]*?)<br>"""
```

Figure 2: Regularni izrazi za spletno stran overstock.com

Regularni izrazi za ekstrakcijo podatkov iz strani rtvslo.si so prikazani na sliki 3.

```
author = """<div\s+class=\"author-timestamp\">\s+<strong>(.*)</strong>"""
publishedTime = """<div\s+class=\"publish-meta\">\s+(.*)<br>"""
title = """<meta\sname=\"title\"\s+content=\"(.*)\">"""
subtitle = """<div\s+class=\"subtitle\">(.*)</div>"""
lead = """<p\s+class=\"lead\">(.*)</p>"""
content = """<\/div>[\s]*<\/figure>[\s]*<p([\s\S]*?)(?=(<figure\sclass=\"mceNonEditable)|(<div\sclass=\"gallery\">))"""
```

Figure 3: Regularni izrazi za spletno stran rtvslo.si

Regularni izrazi za ekstrakcijo podatkov iz strani mimovrste.com so prikazani na sliki 4.

```
title = """<title>(.*)|\s+(?=mimovrste=\)</title>)"""
price = """price__wrap__box__final\">\n*(.*)€"""
ratingPercent = """class=\"badge-rating__percent\">\\n(.*)"""
number = """<span\sdata-sel=\"catalog-number\">(.*)</span></span>"""
reviewNumber = """data-testid=\"badge-rating-count\">\\n(.*)\s+ocen"""
```

Figure 4: Regularni izrazi za spletno stran mimovrste.com

2.2. Implementacija s pomočjo XPath

V drugem delu smo ekstracijo podatkov izvedli s pomočjo XPatha. Za vsak par strani smo znotraj datoteke xpath_extraction.py implementirali funkcijo, ki poskrbi za ekstrakcijo želenih podatkov. Vhodni podatek je HTML datoteka v obliki niza, s pomočjo lxml knjižnice in XPath izrazov pa iz nje izluščimo želeno vsebino. Nekatere podatke smo za lepši izpis primerno obdelali in iz njih odstranili redundanco, nato pa jih shranili v slovar in pred izpisom na standardni izhod shranili v JSON obliko.

XPath izrazi za ekstrakcijo podatkov iz strani rtvslo.si so prikazani na sliki 5.

```
author = '//*[@class="author-name"]/text()'
publish_time = '//*[@class="publish-meta"]/text()'
title = '//*[@id="main-container"]/div[3]/div/header/h1/text()'
subtitle = '//*[@class="subtitle"]/text()'
lead = '//*[@class="lead"]/text()'
content = '//*[@class="article-body"]/article/p/text()|//*[@class="article-body"]/article/p/strong/text()'
```

Figure 5: XPath izrazi za spletno stran rtvslo.si

XPath izrazi za ekstrakcijo podatkov iz strani overstock.com so prikazani na sliki 6.

```
title = '//*/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/a/b/text()'
list_price = '//*/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]' + \
             '/table/tbody/tr/td[1]/table/tbody/tr[1]/td[2]/s/text()'
price = '//*/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]' + \
        '/table/tbody/tr/td[1]/table/tbody/tr[2]/td[2]/span/b/text()'
saving = '//*/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]' + \
         '/table/tbody/tr/td[1]/table/tbody/tr[3]/td[2]/span/text()'
saving_percent = '//*/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]' + \
                 '/table/tbody/tr/td[1]/table/tbody/tr[3]/td[2]/span/text()'
content = '//*/span[@class="normal"]/text()|//*/span[@class="normal"]/a/span/b/text()'
```

Figure 6: XPath izrazi za spletno stran overstock.com

XPath izrazi za ekstrakcijo podatkov iz strani mimovrste.com so prikazani na sliki 7.

```
title = '//*[@id="main-content"]/div/div/div[2]/article/h1/text()'
price = '//*[@class="price__wrap__box__final"]/text()'
item_number = '//*[@class="additional-info__warranty-and-catalog-number"]/span/span/text()'
number_of_reviews = '//*[@class="reviews__ratings"]/h2/text()'
rating_p = '//*[@class="product-ratings__overall"]/span/text()'
```

Figure 7: XPath izrazi za spletno stran mimovrste.com

2.3. Samodejni spletni algoritem za ekstrakcijo RoadRunner

RoadRunner (in njemu podobne implementacije) je pristop k ekstrakciji podatkov, ki deluje na avtomatiziran način. Le ta s pomočjo nabora spletnih strani ustvari ovoj, s pomočjo katerega lahko nato izluščimo podatke iz spletnih strani. Delovanje pristopa temelji na tesni korespodenci vgnezdenih tipov ter regularnih tipov brez unije (ang. *union–free regular expressions*, nadalje UFRE). Podrobnosti le-tega, kot tudi teoretično ozadje, na katerem algoritem stoji, so opisane v delu [1]. Avtorji članka s pomočjo omenjenega teoretičnega ozadja uvedejo algoritem `match`(σ_1, σ_2), ki izračuna najmanjšo zgornjo mejo UFRE σ_1 in σ_2. V bistvu gre pri algoritmu za primerjanje znakov oznak ter nizov dveh ali več spletnih strani, na podlagi česar se nato gradi ovoj spletne strani. Naša implementacija RoadRunner algoritma je v veliki meri povzeta iz prej omenjenega dela [1], vendar pa smo se ponekod odločili za drugačen pristop; razlogi so opisani v ustreznih razdelkih. Psevdokoda algoritma je prisotna na sliki 12.

Podane spletne strani smo prebrali s pomočjo knjižnice BeautifulSoup4[1] (krajše BS4), pri tem pa smo se odločili, da spletno stran pretvorimo v obliko, ki je skladna s predpisi XHTML. S tem mitigiramo morebitne nevšečnosti, do katerih bi morda prišlo zaradi neskladnosti, nekonsistentnosti oziroma nepravilnosti spletnih strani, do katerih prihaja zaradi precej manj omejujočega HTML. Knjižnjica BS4 nam kot rezultat branja vrne drevesno strukturo. Preden datoteki poženemo skozi RoadRunner, smo se odločili še za izbris določenih oznak, za katere smo menili, da k ekstrakciji ne doprinesejo pomembnih podatkov (te so `head`, `script`, `noscript`, `style`, `svg`, `img`[2], `Doctype`, ter komentarji). Prav tako smo se zaradi boljše berljivosti odločili za izbris vseh atributov razen atributov `class` in `id`, katerih uporabo omenimo kasneje.

S globinskim (ang. *depth first*) sprehodom skozi drevo vsake HTML datoteke izvajamo primerjanje HTML datotek in iščemo neujemanja v drevesih. Ločimo dva različne neujemanja vozlišč: neujemanje oznak in neujemanje nizov, ki sta opisana v nadaljevanju. Za označevanje ovoja uporabimo atribut `roadrunner_optional`, vrednost katerega določimo glede na to, kakšnega tipa je (`?` če gre za poljuben/opcijski element, `*` če gre za 0 ali več-kratno pojavitev elementa in +, če gre za 1 ali več-kratno pojavitev elementa). Primer oznake `div`, za katero je bilo ugotovljeno, da je opcijska, je prikazana na sliki 8.

```
<div roadrunner_optional="?">...</div>
```

Figure 8: Primer opcijske oznake.

Pri sprehodu skozi drevo in spreminjanjem le-tega za potrebo gradnje ovoja spreminjamo tudi drugo drevo in sicer enako kot prvo. S tem si olajšamo primerjavo vozlišč na ustreznih pozicijah.

2.3.1. Neujemanje nizov (string mismatch). Do neujemanja nizov pride, ko pride do neujemanja vsebine niza v soležnih vozliščih obeh dreves. V tem primeru vrednost niza spremenimo v `#PCDATA`.

2.3.2. Neujemanje oznak (tag mismatch). Pri primerjanju smo se poslužili določenih hevristik. Omembe vredno je primerjanje atributov razreda `class` ter identifikatorja `id`. Za to smo se odločili zaradi opažanja, da so lahko zaporedoma prisotne oznake, katerih ime je sicer enako, vendar pa je njihova vsebina različna, na podlagi podanih primerov spletnih strani odločili, da za ujemanje dveh oznak zadošča veljanje vsaj enega od sledečih pogojev:

1. https://pypi.org/project/beautifulsoup4/
2. Za izpust oznak `svg` in `img` se odločimo, ker se v našem primeru osredotočamo na tekstovne podatke. Preprosto pa bi kodo spremenili, da v upoštev jemljemo tudi le-te, in sicer tako, da vrstice, ki so zadolžene za brisanje tovrstnih oznak, zakomentiramo.

1) oznaki imata atribut razreda, vendar je oznaka obeh prazna
2) oznaki imata atribut razreda, ujemata pa se v vsaj v prvem navedenem razredu
3) oznaki imata atribut identifikatorja, le-ta pa sta enaka
4) oznaki nimata ne atributa razreda ne atributa identifikatorja

V primeru ujemanja smo ujemanje preverjali rekurzivno na podrejenih vozliščih vsake oznake (oziroma če je bila oznaka seznam smo v otrocih odkrivali iteratorje). V primeru neujemanja pa smo iskali opcijske elemente.

2.3.3. Odkrivanje opcijskih oznak (discovering optionals). V primeru, da ni zadoščen noben izmed zgoraj navedenih pogojev, se lotimo iskanja oznak (s sprehodom naprej po oznakah), kjer bi vsaj eden od le-teh veljal. V ta namen uporabimo svojo implementacijo navkrižnega iskanja (ang. *cross-search*), ki deluje po sledečem principu:

1) oznaki v trenutnem vozlišču obeh dreves se skladata: dodaj vsa dosedanja vozlišča (od odstopanja naprej) prvega in drugega drevesa kot opcijske oznake
2) trenutna oznaka prvega drevesa se je že prej pojavila v drugem drevesu. V tem primeru dodaj kot opcijske elemente dosedanje elemente prvega drevesa ter elemente drugega drevesa do omenjenega indeksa
3) enak razmislek kot 2), le da se zamenjata drevesi

V nasprotnem primeru se premaknemo na naslednji element v obeh drevesih in ponovimo postopek. V primeru, da smo v obeh drevesih prišli do konca, le dodamo vse oznake kot opcijske oznake (in sicer najprej oznake prvega drevesa, nato drugega).

Omenimo še primer, ko je v enem vozlišču zmanjkalo podrejenih vozlišč, v drugem pa ni. V tem primeru dodamo preostala vozlišča daljšega preostalega poddrevesa kot opcijske elemente.

2.3.4. Odkrivanje iteratorjev (discovering iterators). Odkrivanja iteratorjev se v naši implementaciji lotimo na drugačen način, kot je omenjen v delu [1]. Opazimo namreč, da iste oznake lahko pojavljajo ena za drugo, četudi le-teh ne bi razumeli kot iteratorji. Zato smo za iterativne elemente vzeli le elemente, ki se pojavljajo znotraj struktur `ul`, `ol` (katerih elementi so oznake `li`) ter `thead`, `body` ali `tfoot` (katerih elementi so oznake `tr`).

Sprva preverimo, ali je število elementov obeh seznamov enako. V tem primeru primerjamo vsak element enega drevesa z ustreznim elementom drugega. V primeru, da seznama nista enako dolga, sprva preverimo, kateri izmed seznamov je daljši. Nato naredimo primerjavo le-tega s prejšnjim elementom in ugotovimo, ali sta elementa (razen nizov, ki jih vsebujeta), popolnoma enaka. Tu naša implementacija ponovno odstopa od uradne. Za le-to smo se odločili, ker smo opazili, da lahko pride do neželenega spajanja elementov uporabniškega vmesnika, ki so podani v takšni strukturi (recimo element na `overstock.com`, ki prikazuje, koliko izdelkov izmed vseh je prikazanih na trenutni strani, in je prisoten le na prvi strani).

V primeru, da sta elementa popolnoma enaka (razen potencialno vsebovanih nizov), element označimo kot element iteracije z atributom `roadrunner_optional="+"`, če vsaka stran vsebuje vsaj en tovrsten element, ter `"*"`, če ena od strani ne vsebuje tovrstnega elementa. V primeru, da se elementa razlikujeta, elemente do ujemajočega indeksa primerjamo enega z drugim, nato pa preostale elemente daljšega sezama označimo kot opcijske.

2.3.5. Rezultati. Za prikaz delovanja naše implementacije RoadRunnerja program poženemo na delno modificiranem primeru, prisotnem v [1], prikazanih na sliki 9 in sliki 10.

Rezultat (torej ovoj), ki ga dobimo, ko skozi naš program poženemo omenjeni `HTML` datoteki, je prikazan na sliki 11. Pozorni smo na to, da smo nekatere oznake pred zagonom filtrirali. Tako vidimo odsotnost oznake `img` v rezultatu. Za demonstracijo opcijskih oznak zato vpeljemo `footer`, katere prisotnost kot opcijska oznaka zasledimo v rezultatu.

Rezultati za preostale spletne strani in testne primere so prisotni v repozitoriju[3].

Če bi želeli iz naših ovojev pridobiti podatke, bi se najverjetneje tega lotili, da bi sprva prebrali vse nize, ki se nahajajo na pozicijah, kjer se pojavi `#PCDATA`, nato pa bi verjetno ekstrahirali tudi opcijske podatke; menimo, da bi le-ti morda nosili neko vrednost.

3. Zaključek

Vsak izmed pristopov ima svoje slabosti, kot tudi prednosti. Implementaciji s pomočjo regularnih izrazov oz. XPath izrazov sta preprosti in hitri metodi, vendar je njuna pomanjkljivost ta, da je izraze v primeru spreminjanja same spletne strani potrebno nenehno posodabljati. Hiba RoadRunner algoritma je dejstvo, da je brez prisotnosti pomoči umetne inteligence težko učinkoviti ugotoviti, kaj so pravzaprav zares podatki; potrebne so tudi določene hevristike, ki morda na določenih spletnih straneh delujejo bolje kot na drugih. Prav tako z avtomatizacijo pridobljenim podatkom brez (že prej omenjene) omoči umetne inteligence težko določimo pomen (kaj točno te pridobljeni podatki pravzaprav so). Prednost v pristopu pa je očitna, namreč za enkstrakcijo podatkov nam ni več potrebno izdelovati ovoja za vsako stran posebej; le-ta se ustvari sam.

Seminarska naloga se nam je zdela zanimiva. Kot probleme, na katere smo naleteli, omenimo precej težavno in pogosto neintuitivno urejanje BS4 drevesa ter, po našem mnenju, slabo dokumentiranost za izvajanje le-tega.

References

[1] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, et al. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, volume 1, pages 109–118, 2001.

3. https://github.com/lukak85/s.pyderman/tree/master/pa2

```
<HTML>
Books of:
<B>
    John Smith
</B>
<UL>
    <LI>
        <I>Title:</I>
        DB Primer
    </LI>
    <LI>
        <I>Title:</I>
        Comp. Sys.
    </LI>
</UL>

</HTML>
```

Figure 9: Primer spletne strani 1.

```
<HTML>
Books of:
<B>
    Paul Jones
</B>
<IMG src="#" />
<UL>
    <LI>
        <I>Title:</I>
        XML at Work
    </LI>
    <LI>
        <I>Title:</I>
        HTML Scripts
    </LI>
    <LI>
        <I>Title:</I>
        Javascript
    </LI>
</UL>
<footer>TestFooter</footer>

</HTML>
```

Figure 10: Primer spletne strani 2.

```
<html>
 <body>
  <p>
   Books of:
   <b>
    #PCDATA
   </b>
  </p>
  <ul>
   <li roadrunner_optional="+">
    <i>
     Title:
    </i>
    #PCDATA
   </li>
  </ul>
  <footer roadrunner_optional="?">
   TestFooter
  </footer>
 </body>
</html>
```

Figure 11: Rezultat RoadRunnerja, ovoj (wrapper), pridobljen s primeri.

```
def RoadRunner(s, c):
    i = 0
    while i < len(s) or i < len(c):
        if eno izmed vozlišč nima več elementov:
            # Dodaj manjkajoče elemente kot "?"
        elif ujemanje oznak:
            if element je seznam:
                s[i] = iterator_matching(s, c)
            else:
                s[i] = RoadRunner(s[i], c[i])
        elif neujemanje nizov:
            s[i] = "#PCDATA"
        elif ujemanje nizov:
            pass
        elif neujemanje oznak:
            # Navkrižno ujemanje, dodaj ustrezne elemente kot "?"
    return s

def iterator_matching(s, c):
    if len(s) == len(c):
        for i=0:len(s):
            s[i] = RoadRunner(s[i], c[i])
    else:
        if len(s) > len(c) and s[len(s) - 1] != s[len(s) - 2]:
            # Primerjamo istoležne elemente dreves, ostale dodamo kot "?"
            # Podobno storimo, če do situacije pride v drugem drevesu
        elif n1_len > 0 and n2_len > 0:
            # Dodaj manjkajoče elemente kot "+"
        elif (len(s) == 0 or len(c) == 0) and not len(s) == len(c):
            # Dodaj manjkajoče elemente kot "*"
    return s
```

Figure 12: Psevdokoda RoadRunner algoritma. `s` in `c` sta sicer vozlišči posameznega drevesa, vendar se zaradi preglednosti nanju sklicujemo kot otroci le-teh.