IEPS - 2. seminarska naloga

Delfina Bariša, Nejc Smrkolj Koželj, Žiga Simončič

7. maj 2019

1 Uvod

Pri seminarski nalogi je bilo potrebno implementirati tri načine za ekstrakcijo strukturiranih podatkov s spletnih strani. Podatke smo ekstrahirali z uporabo jezika XPath, regularnih izrazov (*RegEx*) in z lastno implementacijo algoritma RoadRunner.

Vse načine smo testiralih na dveh primerih spletnih strani iz domen rtvslo.si, overstock.com in naše izbrane domene - amazon.com.

V nadaljevanju so v kratkem predstavljene podrobnosti posameznega načina ekstrakcije. V naslednjem poglavju podrobneje opišemo uporabo XPath, v poglavju 4 se osredotočimo na uporabo regularnih izrazov, poglavje 5 podrobneje opisuje implementacijo algoritma RoadRunner, zaključek pa se nahaja v poglavju 8.

2 Domena Amazon

Dodali smo domeno Amazon iz katere smo izluščili dve podstrani[1][2] za dva različna produkta. Za posamezen produkt smo izluščili naslov (*Title*), ceno (*Price*), oceno produkta (*Stars*), število ocen (*Number of Reviews*) in opis produkta (*Description*).

3 XPath

Za vsako stran smo za vsako zahtevano polje najprej izluščili tekstovni del najglobje značke s pomočjo XPath izraza. Dobljenemu besedilu smo nato odstranili morebitne JavaScript skripte in odvečne bele presledke. Strani RTV SLO in Amazon sta vsebovali le po en zadetek na posamezno kategorijo medtem ko je stran Overstock vsebovala več zadetkov. Pri tej strani smo pridobili seznam vseh zadetkov nato pa se z zanko sprehodili skozi rezultate ter oblikovali končni

[1] Darth Vader propagandni poster: https://www.amazon.co.uk/Merchandising-669490-Your-Empire-Science-Fiction/dp/B00QF67NKI/ref=$sr_1_fkmr0_1?keywords = EMPIRE+Merchandising+669490+Your+Empire+Needs+You+Star+Wars%2C+Vader%2C+Science+Fiction+Sci+Fi%2C+Film+Poster+61+x+91.5+cm_+Amazon.co.uk_+Amazon.co.uk_htmlqid = 1557262281s = gatewaysr = 8-1-fkmr0$

[2] Anti-kapitalistični propagandni poster: https://www.amazon.co.uk/Vintage-Anti-Capitalist-CAPITALIST-Art-Reproduction/dp/B01N91LC68/ref=$sr_1_fkmr0_1?keywords = Vintage+Anti-Capitalist+PYRAMID+OF+THE+CAPITALIST+SYSTEM+c1911+250gsm+Gloss+Art+Card+Reproduction+Poster_+Amazon.co.uk_+Amazon.co.uk_htmlqid = 1557262288s = gatewaysr = 8-1-fkmr0$

JSON objekt. JSON struktura vsebuje vse zahtevane podatki za posamezno stran. Na strani Overstock smo posamezne zadetke zapisali v seznam.

3.1 RTV SLO

Besedilo smo izluščili s pomočjo spodnjih XPath poti. Dobljeno besedilo smo nato še očistili s pomočjo regularnih izrazov.

- Title: //h1/text()
- Subtitle: //div[@class='subtitle']/text()
- Author: //div[@class='author-name']/text()
- PublishedTime: //div[@class='publish-meta']/text()
- Lead: //p[@class='lead']/text()
- Content: //div[@class='article-body']//*[not(name()='script')]/text()

3.2 Overstock

Besedilo smo izluščili s pomočjo spodnjih regularnih izrazov. Dobljeno besedilo smo nato še očistili.

- Title: //td[@valign='top']/a/b//text()
- ListPrice: //tr/td[@align='left' and @nowrap='nowrap']//text() - vsak prvi element
- Price: //tr/td[@align='left' and @nowrap='nowrap']//text() - vsak drugi element
- Saving in SavingPercent: //tr/td[@align='left' and @nowrap='nowrap']//text() - vsak tretji element
- Content: //td[@valign='top']/span[@class='normal']/text()

Saving in SavingPercen smo nato razdelili na polovico s pomočjo regularnih izrazov glede na oklepaje in znak $.

3.3 Amazon

- Title: //span[@id='productTitle']/text()
- Price: //span[@id='priceblock_ourprice']/text()
- Stars: //a[@class='a-popover-trigger a-declarative']/i/span/text()
- NumberOfReviews: //span[@id='acrCustomerReviewText']/text()
- Description: //ul[@class='a-unordered-list a-vertical a-spacing-none']//span/text()

4 RegEx

Postopek izljuščevanja je bil podoben kot pri XPath. Vsako zahtevano polje smo izluščili z le enim RegEx nizom. Pri strani Overstock je bil RegEx malo bolj kompleksen saj strani manjkajo atributi na HTML značkah po katerih bi se lahko orientirali.

4.1 RTV SLO

Besedilo smo izluščili s pomočjo spodnjih XPath poti. Dobljeno besedilo smo nato še očistili s pomočjo regularnih izrazov.

- Title: <h1>.+?</h1>
- Subtitle: <div class=\"subtitle\">.+?
- Author: <div class=\"author-name\">.+?
- PublishedTime: <div class=\"publish-meta\">.+?
- Lead: <p class=\"lead\">.+?</p>
- Content: <div class=\"article-body\">.+?<div class=\"article-column\">

4.2 Overstock

Besedilo smo izluščili s pomočjo spodnjih regularnih izrazov. Dobljeno besedilo smo nato še očistili.

- Title: <td valign=\"top\">\s*<a href=\"[^\"]+\">\s*[^<]+
- ListPrice: <s>\$[^<]+</s>
- Price: \$[^<]+
- Saving: <span class=\"littleorange\">\$[^(]+
- SavingPercent: \([0-9]*%\)
- Content: <span class=\"normal\">[^<]+

Zadetke smo s pomočjo zanke razporedili v končno JSON strukturo.

4.3 Amazon

- Title: <span id=\"productTitle\".+?
- Price: <span id=\"priceblock_ourprice\".+?
- Stars: <i class=\"a-icon a-icon-star.+?</i>
- NumberOfReviews: <span id=\"acrCustomerReviewText\".+?
- Description: <ul class=\"a-unordered-list a-vertical a-spacing-none\">.+?

5 Algoritem RoadRunner

5.1 Opis

V nasprotju z načinoma XPath in RegEx, kjer točno določimo pot in podatke za ekstrakcijo, algoritem RoadRunner pomembne podatke in strukturo strani določi sam. Na podlagi več primerkov spletnih strani (z iste domene/podstrani), ki jih med seboj primerja, zgradi ovojnico s pomočjo katere lahko izluščimo pomembne podatke. Ovojnica je lahko zapisana na različne načine, vendar mora nazorno opisati/predstaviti strukturo spletne strani.

Algoritem je opisan v kar nekaj člankih. Najbolj smo si pomagali s člankom [1], saj je jasen in jedrnat.

RoadRunner deluje na osnovi sprehoda po drevesu in zaznavanja ujemanj med HTML značkami in nizi (vsebino). Ujemanje nizov (vsebine) predstavlja fiksne podatke (npr. naslov strani), neujemanje nizov podatke za ekstrakcijo, ujemanje značk pa splošno ujemanje strukture dveh ali več strani.

Algoritem zazna tudi opcijsko vsebino in sezname.

5.2 Implementacija

Alogoritem smo implementirali v programskem jeziku Python. močno smo si pomagali s knjižnico BeautifulSoup, ki iz vira strani sestavi drevo po katerem se lahko enostavno premikamo.

Za sprehod po drevesu smo uporabili rekurzijo. Rekurzivno se sprehodimo skozi drevesi dveh strani, sproti pa preverjamo ujemanja značk ali nizov (vsebine) ter gradimo ovojnico. Kadar se znaka ali niza spletnih strani ne ujemata, skušamo to neujemanje rešiti z generalizacijo. V poglavju 5.2.2 bomo podrobneje opisali reševanje neujemanja nizov. V poglavju 5.2.2 pa bomo pojasnili reševanje neujemanja znakov.

5.2.1 Neujemanje nizov

Neujemanje nizov se zgodi, ko se se na enakem položaju spletnih strni pojavita različna niza. Primer 5.1 prikazuje niza, ki se ujemata in generirano ovojnico.

Primer 5.1:

Listing 1: Stran 1

```
<HTML>
    <BODY>
        <B>Title:</B>
    </BODY>
</HTML>
```

Listing 2: Stran2

```
<HTML>
    <BODY>
        <B>Title:</B>
    </BODY>
</HTML>
```

Listing 3: Ovojnica

```
<body>
    <b>Title:<\/b>\s*
<\/body>\s*
```

Primer 5.2 prikazuje niza, ki se ne ujemata in generirano ovojnico.

Primer 5.2:

Listing 4: Stran 1

```
<HTML>
    <BODY>
        <B>Title1:</B>
    </BODY>
</HTML>
```

Listing 5: Stran2

```
<HTML>
    <BODY>
        <B>Title2:</B>
    </BODY>
</HTML>
```

Listing 6: Ovojnica

```
<body>
    <b>(.*)<\/b>\s*
<\/body>\s*
```

V primeru 5.2.2 sta se niza "Title1:"in "Title2:". To neujemanje smo rešili tako, da smo v ovojnici označili, da je na tem mestu lahko poljuben niz. Slednje smo označili s "(.*)".

5.2.2 Neujemanje znakov

Med neujemanje znakov štejemo:

- neujemanje med različnima znakom
- neujemanje med enim znakom in enim nizom

Kljub temu, da ne ujemanje znakov delimo na zgoraj napisana primera, bomo zaradi lažjega razumevanja v nadaljevanju pisali kar neujemanje znakov.

Ko se dva znaka med spletnima stranema ne ujemata ločimo dve možnost:

1. Ena stran vsebuje znak, druga pa ne. Takšnim znakom rečemo opcijski znaki in jih bomo podrobneje pojasnili v poglavju 5.2.2.1.
2. Obe strani vsebujeta znak, a se na eni strani pojavi večkrat kot na drugi. Takšnim znakom pravimo ponavljajoči se znaki ali krajše iteracije. Njih bomo več povedali v poglavju poglavju 5.2.2.2.

5.2.2.1 Opcijski znaki Opcijski znaki sto tisti, ki se pojavijo samo v eni izmed spletnih strani. Ko smo prišli do ne ujemanja dveh znakov, smo za vsakega posebej preverili, ali se nahaja na drugi spletni strani. Če se ni nahajal na drugi spletni strani, je pomenilo, da je opcijski. Tako samo ga ovili v "()?". Znak lahko vsebuje več znakov, zato smo morali rekurzivno pridobiti tudi ovojnico njegove vsebine.

Primer 5.3 prikazuje opcijski znak na Strani 1 in generirano ovojnico.

Primer 5.3:

Listing 7: Stran 1

```
<HTML>
    <BODY>
        <B>Title:</B>
        <H1>Shrek 1</H1>
        <IMG>Shrek1.jpg</IMG>
    </BODY>
</HTML>
```

Listing 8: Stran 2

```
<HTML>
    <BODY>
        <B>Title:</B>
        <H1>Shrek 2</H1>
    </BODY>
</HTML>
```

Listing 9: Ovojnica

```
<body>(.*)
    <b>Title:<\/b>\s*
    <h1>(.*)<\/h1>\s*
    (<img\/>\s*)?
<\/body>\s*
```

5.2.2.2 Ponavljanje znake (iteracija) Ponavljajoči se znak je tisti, ki se večkrat pojavi zaporedoma. Ko se na eni strani znak zaporedoma pojavi več krat kot na drugi, pride do ne ujemanja. Takrat smo preverili, če je znak res enako predhodnem/u znaku/om in če se po njem pojavlja še kateri znak. V ovojnici smo takšen znak prikazali kot en znak, ki smo ga obdali z "()+". Znak lahko vsebuje več znakov, zato smo se odločili, da se znaka ujemata samo, če se ujema tudi njuna vsebina. Tak kot pri opcijskih znakih, smo tudi tu morali rekurzivno pridobiti ovojnico vsebine znaka.

Primer 5.4 prikazuje ponavljajo zanko, ki se na Strani 2 pojavi več krat kot na Strani 1 in generirano ovojnico.

Primer 5.4:

Listing 10: Stran 1

```
<HTML>
    <BODY>
        Title:
        <B>Shrek 1</B>
        <IMG>Shrek1.jpg</IMG>
        <UL>
            <B>Directed by:</B>
            <LI>
                <I>Andrew Adamson</I>
            </LI>
            <LI>
                <I>Vicky Jenson</I>
            </LI>
        </UL>
    </BODY>
</HTML>
```

Listing 11: Stran 2

```
<HTML>
    <BODY>
        Title:
        <B>Shrek 2</B>
        <UL>
            <B>Directed by:</B>
            <LI>
                <I>Andrew Adamson</I>
            </LI>
            <LI>
                <I>Kelly Asbury</I>
            </LI>
            <LI>
                <I>Conrad Vernon</I>
            </LI>
        </UL>
    </BODY>
</HTML>
```

Listing 12: Ovojnica

```
<body>
    Title:
    <b>(.*)<\/b>\s*
    (<img\/>\s*)?
    <ul>\s*
        (<li>\s*
            <i>(.*)<\/i>\s*
        <\/li>\s*)+
    <\/ul>\s*
<\/body>\s*
```

5.3 Hevristike

Pri predprocesiranju iz vseh HTML značk odstranimo vse HTML atribute, razen atributa *id*. Poleg ujemanja značk po imenu preverjamo tudi ujemanje po id-ju, saj lahko v nasprotnem primeru pride do napačne poravnave strani.

To se lahko zgodi, če imata strani isto strukturo, vendar se na eni od straneh pred enakim blokom pojavi identični blok, ki je unikaten tej strani. Brez upoštevanja id-jev, bi se zaznalo ujemanje navadnega bloka prve strani s 'posebnim' blokom druge strani, zato bi bila nadaljnja ujemanja zamaknjena, kar bi nas privedlo do napačne ovojnice. Z upoštevanjem id-jev pa poskrbimo, da vsebinsko ustrezne bloke pravilno poravnamo in tisti 'poseben' blok zaznamo kot *optional*.

5.4 Psevdokoda

Listing 13: Psevdokoda

```
RoadRunner(tag1, tag2):
    wrapper = ""

    if tag1.name == None or tag2.name == None:
        return wrapper

    if tag1.name == tag2.name and tag1.id = tag2.id:
        if tags end inline:
            wrapper = <tag1.name/>
            return wrapper
        if tags have id attribute:
            wrapper = <tag1.name id=tag1.id>
        else:
            wrapper = "<tag1.name>"

        # Check for string match
        if tag1.content == tag2.content:
            # Match, add text to wrapper
            wrapper += tag1.content
        else:
            # Mismatch, interesting text, extract
            wrapper += "(.*)"


        loop through children of tag1:
            loop through children of tag2:
                w = RoadRunner(child of tag1, child of tag2)
                if w == "":
                    continue
                else:
                    if children of tag2 were skipped:
                        w = Mismatch(tag1, tag2, skipped)
                    wrapper += w
            #notFound: child of tag1 not found in children od tag2
            if notFound:
                wrapper += Mismatch(tag2, tag1, notFound)
        #notChecked:not checked children of tag2
        if notChecked:
                wrapper += Mismatch(tag1, tag2, notFound)

        wrapper += </tag1.name>

    return wrapper

Mismatch(tag1, tag2, miss):
        loop through skipped children:
            if skipped child in children of tag1:
                # Check their content and iterators
                w = (corrected(w))?
            else:
                FindIterators:
                    # Look for all repeated tags
                    w = (corrected(w))+
```

6 Čiščenje besedila in preprocesiranje

Pri XPath in RegEx smo besedilo tudi očistili tako, da smo najprej odstranili vse JavaScript skripte. Nato smo besedilu odstranili morebitne HTML značke. Na koncu smo odstranili še vse bele presledke.

Pri RoadRunnerju smo odstranili znački *script* in *style*, komentarje in vse atribute, razen atributa *id*. Ovojnico smo generirali samo za *body*.

7 Rezultati

Rezultate za XPath in RegEx smo v standardnem JSON formatu priložili izvorni kodi. Ovojnico, zapisano v obliki regularneta izraza, ki je rezultat algoritma RoadRunner smo prav tako priložili k izvorni kodi.

8 Zaključek

Poročilo opisuje podrobnosti implementiranih načinov ekstrakcije podatkov s spletnih strani - XPath, RegEx in algoritem RoadRunner.

Literatura

[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.