Jaka Krajnc in Janez Škrlj

Poročilo ekstrakcije podatkov spletnih strani

Seminarska naloga pri predmetu iskanje in ekstrakcija podatkov s spleta

MENTOR: doc. dr. Slavko Žitnik

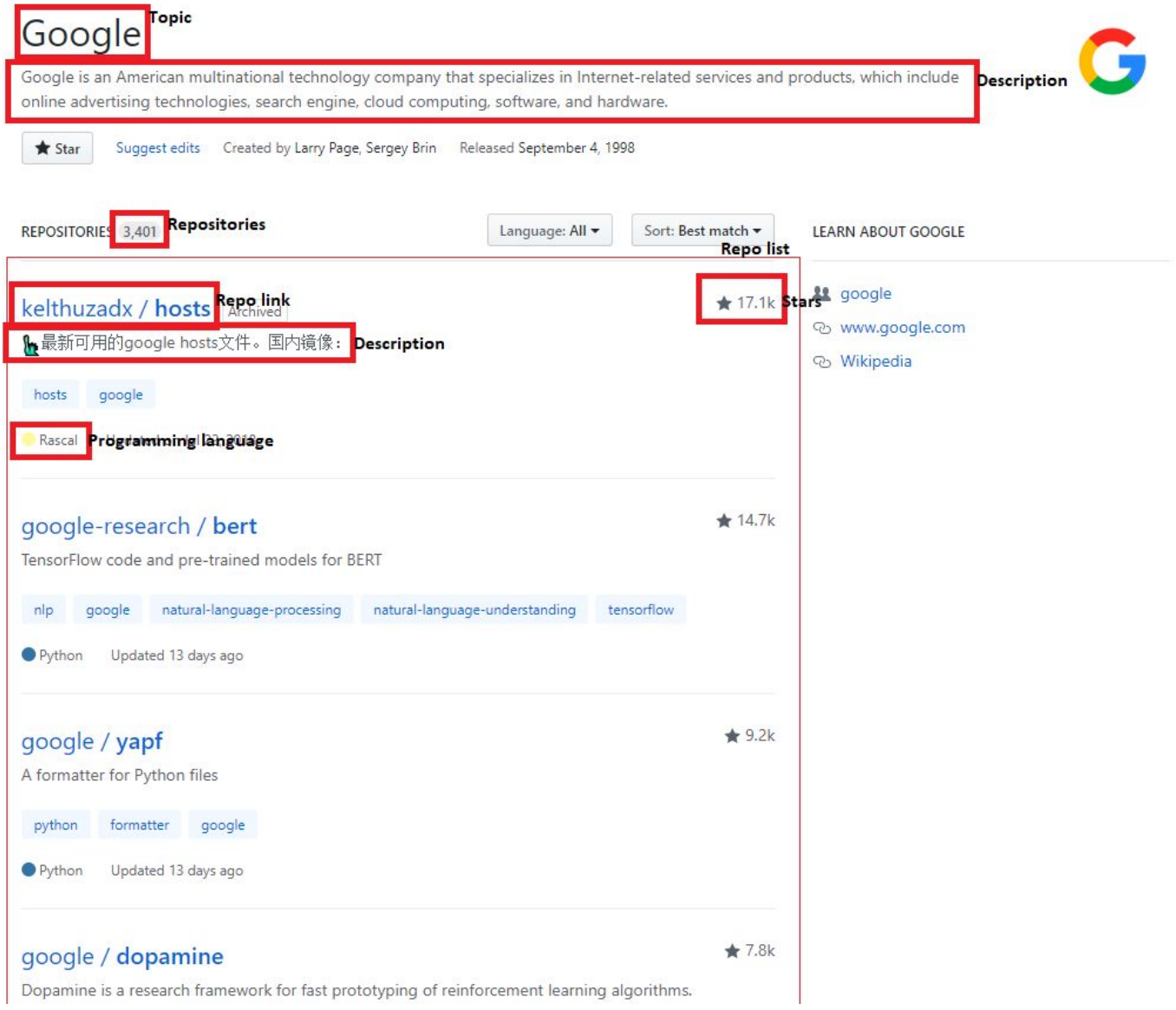
V poročilu smo predstavili tri implementacije strukturirane ekstrakcije podatkov iz spleta.

1. Uvod

Na spletnih straneh lahko najdemo veliko uporabnih podatkov. Pogosto si želimo iz določene strani pridobiti nekatere podatke, zato nas zanima kako so strani strukturirane, da bomo iz strukture lažje izvlekli podatke. Specifične podatke, ki jih vsebuje določena spletna stran lahko iz nje ekstrahiramo v kolikor poznamo strukturo strani. Tipično nas bo zanimala ekstrakcija podatkov iz dveh podobnih strani. V tem primeru lahko za določen tip strani napišemo ekstraktor, ki bo znal prebrati določeno stran. Podatke lahko izvlečemo na več načinov. Najprej bomo prikazali ekstrakcijo s pomočjo regularnih izrazov. Regularen izraz je zaporedje znakov, ki definira iskalni vzorec. Tipično je vzorec uporabljen kot string za iskalni algoritem ali za iskanje in zamenjavo določenih znakov. Koncept regularnih izrazov izvira že iz leta 1950 ko je bil formaliziran prvi opis regularnega izraza. Drugi način iskanja, ki ga bomo uporabili je iskanje s pomočjo jezika XPath. XPath je poizvedovalni jezik za izbor vozlišč iz drevesa XML. Tako za spletno stran najprej zgradimo drevo, nato pa s pomočjo jezika XPath pridobimo vozlišče, ki nas zanima v okviru ekstrakcije podatkov. Prav tako se bomo posvetili implementacije ‘Road Runnerja’, ki bo na podlagi dveh strani vrnil rezultat primerjave strani, ter nam tako omogočil lažji pristop k ekstrakciji podatkov na za nas novi strani, s katere bi želeli pridobiti podatke.

2. Opis spletnih strani po izbiri

Za dodatne spletne strani smo si izbrali dve strani na spletni strani github. Izbrali smo si strani na naslovih: https://github.com/topics/google, https://github.com/topics/react-native. Podatke, ki smo jih želeli pridobiti so prikazani na spodnji sliki.



Slika: Označeni podatki, ki smo jih ekstrahirali.

```
{
  "Topic": "Google",
  "Description": "Google is an American multinational technology company th
  "Repositories": "3,376",
  "RepoList": [
    {
      "RepoLink": "https://github.com/kelthuzadx/hosts",
      "Description": "\ud83d\uddfd\u6700\u65b0\u53ef\u7528\u7684google host
      "Stars": "17.1k",
      "ProgrammingLanguage": "Rascal"
    },
    {
      "RepoLink": "https://github.com/google-research/bert",
      "Description": "TensorFlow code and pre-trained models for BERT",
      "Stars": "14.5k",
      "ProgrammingLanguage": "Python"
    },
    {
      "RepoLink": "https://github.com/google/yapf",
      "Description": "A formatter for Python files",
      "Stars": "9.1k",
      "ProgrammingLanguage": "Python"
    },
    {
      "RepoLink": "https://github.com/google/dopamine",
```


Slika: Oblika zbranih podatkov strani github.

Podatki sestojijo iz imena, opisa, števila repozitorijev ter seznama vseh repozitorijev.

2. Implementacije

Ekstrakcijo podatkov smo implementirali v programskem jeziku Python. Glavni deli programa so sestavljeni iz ekstrakcije z uporabo regularnih izrazov, ekstrakcije z uporabo xpath izrazov ter implementacije roadrunner algoritma. Najprej smo izvedli ekstrakcijo podatkov nad podanimi spletnimi stranmi (rtvslo, overstock). Nato pa smo dodali še dve svoji (github).

2.1 Z uporabo regularnih izrazov

Rtvslo

Najprej smo si temeljito ogledali spletno stran in identificirali značke, ki vsebujejo iskane nize. Na podlagi tega smo izdelali regularne izraze, ki vrnejo ustrezne nize. Stran rtvslo je bila manj zahtevna za analizo, saj ni vsebovala seznama vrednosti.

```
def parse_rtvslo(html):
    # Author name
    author = re.findall('<div class="author-name">(.+?)</div>', html)[0]
    # Title
    title = re.findall('<h1>(.+?)</h1>', html)[0]
    # Subtitle
    subtitle = re.findall('<div class="subtitle">(.+?)</div>', html)[0]
    # Lead
    lead = re.findall('<p class="lead">(.+?)</p>', html)[0]
```

Slika: Primer tvorjenja regularnih izrazov za stran rtvslo

```
return json.dumps({
    "Author": author,
    "PublishedTime": pub_time,
    "Title": title,
    "SubTitle": subtitle,
    "Lead": lead,
    "Content": content,
}, indent=2, ensure_ascii=False)
```

Slika: Rezultat metode, ki ekstrahira vsebino strani rtvslo

Overstock

Stran overstock je od nas zahtevala nekaj več dela, saj smo imeli opravka s seznamom artiklov. HTML vsebino smo dobro analizirali in na podlagi te napisali regularen izraz, ki nam je vrnil vsebino. Prav tako je bilo pomembno da smo zelo dobro definirali kdaj gre za artikel, saj stan vsebuje zelo podobne vrstice, ki ne predstavljajo dejanskega artikla.

```
# Match rows
matched = re.findall('<tr bgcolor="(#ffffff|#dddddd)">'
                     '\n<td valign="top" align="center">'
                     '\n<table>((.|\n|\r)+?)</table></td>' # unused data
                     '<td valign="top">((.|\n|\r)+?)<b>((.|\n|\r)+?)</b></a><br> \n'  # last value is title
                     '<table>((.|\n|\r)+?)<table> \n<tbody>'
                     '<tr>((.|\n|\r)+?)<td align="left" nowrap="nowrap"><s>((.|\n|\r)+?)</s></td></tr> \n'  # list price
                     '<tr>((.|\n|\r)+?)<td align="left" nowrap="nowrap"><span class="bigred"><b>((.|\n|\r)+?)</b></span></td></tr>
                     '<tr>((.|\n|\r)+?)<td align="left" nowrap="nowrap"><span class="littlearange">((.|\n|\r)+?) \(((.|\n|\r)+?)\)'
                     '</span></td></tr>'  # you save
                     '((.|\n|\r)+?)'
                     '<td valign="top"><span class="normal">((.|\n|\r)+?)</td>'  # description
                     '((.|\n|\r)+?)'
                     , html)
```

Slika: Regularen izraz, ki zajame en artikel.

```
parsed_rows = []
# Process each row
for row in matched:
    parsed_rows.append({
        "Title": row[5],
        "ListPrice": row[11],
        "Price": row[15],
        "Saving": row[19],
        "SavingPercent": row[21],
        "Content": clean_html(row[25])
    })
return json.dumps(parsed_rows, indent=2)
```

Slika: Rezulat metode parse_overstock

Github

Za ekstrakcijo podatkov po našem izboru smo izbrali podobni spletni strani iz spletnega mesta github. Najprej smo za vsako stran pridobili osnovne podatke, kot so ime, opis, število repozitorijev, nato pa preiskali še vse repozitorije na strani.

```
def parse_github(html):
    # Page name
    topic = re.findall('<h1 class="text-normal mb-1">(.+?)</h1>', html)[0]
    description = re.findall('<p>(.+?)</p>', html)[0]
    repos = re.findall('<span class="Counter">(.+?)</span>', html)[0]

    programming_lngs = re.findall('<span itemprop="programmingLanguage">(.+?)</span>', html)
    stars = re.findall('<svg class="octicon octicon-star mr-1" viewBox="0 0 14 16"(?:(?:.|\n|\r)+?)</svg>((?:.|\n|\r)+?)</a>', html)
    descriptions = re.findall('<div class="text-gray mb-3 ws-normal">((?:.|\n|\r)+?)</div>', html)
    repo_links = re.findall('<h3 class="f3">(?:(?:.|\n|\r)+?)<a href="((?:.|\n|\r)+?)">(?:(?:.|\n|\r)+?)</h3>',html)
```

Slika: Regularni izrazi, ki zajamejo podatke na strani github

```
for i in range(len(programming_lngs)):
    repoList.append({
        'RepoLink': repo_links[i],
        'Description': clean_html(descriptions[i].strip()),
        'Stars': stars[i + 1].strip(),
        'ProgrammingLanguage': programming_lngs[i],
    })

data = {
    "Topic": topic,
    "Description": description,
    "Repositories": repos,
    "RepoList": repoList,
}
return json.dumps(data, indent=2, ensure_ascii=False)
```

Slika: Rezulat metode parse_github

2.2 Z uporabo xpath

Implementacija zajemanja besedila je bila z uporabo xpath-a še lažja kot z regularnimi izrazi, ker je xpath zasnovan za delo na dokumentih drevesnih oblik, kakršen je tudi html, xml, ...

Rtvslo

```
def parse_rtvslo(html_in):
    tree = html.fromstring(html_in)
    # # Author name
    author = tree.xpath('//div[@class="author-name"]/text()')[0]
    # # Published time
    pub_time = tree.xpath('//div[@class="publish-meta"]/text()')[0]
    pub_time = re.findall('\n\t\t(.+?). (.+?) ob (.+)', pub_time)
    pub_time = pub_time[0][0] + ". " + pub_time[0][1] + " ob " + pub_time[0][2]
    # # Title
    title = tree.xpath('//h1/text()')[0]
    # # Subtitle
    subtitle = tree.xpath('//div[@class="subtitle"]/text()')[0]
    # # Lead
    lead = tree.xpath('//p[@class="lead"]/text()')[0]
    # Content
    content_list = tree.xpath('//article/p/text()')
    content = ""
    for p_txt in content_list:
        content = content + p_txt

    return json.dumps({
        "Author": author,
        "PublishedTime": pub_time,
        "Title": title,
        "SubTitle": subtitle,
        "Lead": lead,
        "Content": content,
    }, indent=2, ensure_ascii=False)
```

Slika: zajemanje podatkov z uporabo xpath

Za zajem avtorjevega imena smo izbrali element div, ki je imel za razred "author-name". Na enak način smo dobili tudi čas objave, le da smo za razred izbrali "publish-meta". Ker je končna oblika podatkov od nas zahtevala datum v posebni obliki, smo z regularnim izrazom izbrali dva dela datuma in ure, ter ju nato shranili v drugačnem zaporedju kot sta bila izvlečena. Na enak način (z določanjem razredov) smo dobili tudi podnaslov in krajši opis članka. Vsebino članka smo dobili z izbiro elementa article. Ker smo dobili element article v seznamu po več vrsticah, smo ga morali na koncu še združiti v en niz, katerega smo nato shranili v json datoteko skupaj z ostalimi podatki.

Overstock

```
def parse_overstock(html_in):
    tree = html.fromstring(html_in)

    title       = tree.xpath('//tr/td[@valign="top"]/a/b/text()')
    list_price = tree.xpath('//table/tbody/tr/td[@align="left"]/s/text()')
    price       = tree.xpath('//table/tbody/tr/td[@align="left"]/span/b/text()')
    save        = tree.xpath('//table/tbody/tr/td[@align="left"]/span/text()')
    content     = tree.xpath('//td[@valign="top"]/span[@class="normal"]/text()')
    for i in range(len(content)):
        content[i] = content[i].replace("\n", " ")
        save[i] = re.findall("(.*) \((.*)\)",save[i])
    parsed_rows = []
    for i in range(len(title)):
        parsed_rows.append({
                "Title": title[i],
                "ListPrice": list_price[i],
                "Price": price[i],
                "Saving": save[i][0][0],
                "SavingPercent": save[i][0][1],
                "Content": content[i]
            })
    return json.dumps(parsed_rows, indent=2)
```

Slika: Zajem podatkov strani overstock z uporabo xpath.

Najprej zgradimo drevo z funkcijo fromstring, kateri za parameter podamo niz vsebine celotnega html dokumenta. Naslove dobimo na tak način da se osredotočimo na vse td elemente z atributom valign "top", znotraj tr elementov. Iz njih si potem izberemo tudi znotraj ležeče a in b elemente (/a/b). Ker nas zanima besedilo znotraj teh elementov na koncu dodamo še /text(). Na podoben način dobimo tudi cene, prihranke in vsebino opisa izdelka. Pri vsebini nato še zamenjamo vse \n elemente s praznim nizom in na tak način očistimo besedilo. Ker bi radi dobili ceno in prihranek posebej, smo uporabili tudi regularni izraz s katerim iz vrstice, ki vsebuje ceno in prihranek, ločeno dobimo ceno in prihranek. Vse te vrednosti na koncu shranimo v json obliko.

Github

```
def parse_github(html_in):
    tree = html.fromstring(html_in)

    topic = tree.xpath('//h1[@class="text-normal mb-1"]/text()')[0].strip()
    description = tree.xpath('//div[@class="col-md-10 text-gray"]/p/text()')[0].strip()
    repos = tree.xpath('//h2[@class="h6 text-uppercase mb-2 mb-md-0"]/span/text()')[0].strip()

    repo_links_names = tree.xpath('//h3[@class="f3"]/a/@href')
    descriptions = tree.xpath('//article[@class="border-bottom border-gray-light py-4"]/div[@class="text-gray mb-3 ws-normal"]')
    stars = tree.xpath('//a[@class="d-inline-block link-gray"]')
    programming_languages = tree.xpath('//span[@itemprop="programmingLanguage"]/text()')
    parsed_rows = []

    for i in range(len(descriptions)):
        descriptions[i] = descriptions[i].xpath("normalize-space()")
    for i in range(len(stars)):
        stars[i] = stars[i].xpath("normalize-space()")

    for i in range(len(programming_languages)):
        parsed_rows.append({
            "RepoLink":            repo_links_names[i],
            "Description":         descriptions[i],
            "Stars":               stars[i],
            "ProgrammingLanguage": programming_languages[i],
        })


    data = {
        "Topic": topic,
        "Description": description,
        "Repositories": repos,
        "RepoList": parsed_rows,
    }

    return json.dumps(data, indent=2)
```

Slika: Zajem podatkov strani github z uporabo xpath.

Ravno tako kot prej zgradimo drevo z uporabo metode fromstring. Na enak način kot smo delali do sedaj, predvsem z izbiro specifičnih razredov elementov, ki nas zanimajo, shranimo vrednosti v spremenljivke. Na koncu nad zbranimi podatki opisov repozitorijev in števila zvezdic, pokličemo funkcijo normalize-space, ki odstrani prazne vrstice, \n in prazne prostore pred ter za besedilom.

2.3 Opis roadrunner algoritma

Na začetku se osredotočimo le na elemente, ki se nahajajo v elementu body. Z algoritmom road runner želimo zgraditi strukturo, ki nam bo pomagala identificirati podobne elemente med dvemi spletnimi stranmi.
V algoritmu tako zgradimo drevo elementov, ki so si enaki glede na pozicijo v DOM strukturi. To storimo tako da primerjamo elemente med sabo in rekurzivno enako ponovimo za vse otroke v vsakem vozlišču. Pomembno je, da preverimo tudi ujemanje značk.
Ker želimo pridobiti tudi mesta, ki so potencialno dobra in vsebujejo želene podatke, nam algoritem vrne poti, ki jih lahko kasneje uprabimo za pridobitev podatkov.
Algoritem nam vrne JSON ujemajoče se strukture ter poti, ki so nam znimive. Kot že omenjeno se omejimo le na značke znotraj oznake body.

```
def wrapper(base_html, input_html):
    base_tree  = BeautifulSoup(base_html, "html.parser")
    input_tree = BeautifulSoup(input_html, "html.parser")

    # Use only body tags for analysis
    base_body  = base_tree.find('body').findChildren(recursive=False)
    input_body = input_tree.find('body').findChildren(recursive=False)

    res = construct('', {}, base_body, input_body)

    return {
        "Matching_Construction": res,
        "Potential_Paths": potential_paths
    }
```

Slika: Metoda, ki poskrbi za ekstrakcijo želenih delov dokumenta HTML, ter prične z konstruktom drevesa ujemanja.

```
def construct(path, map, chlds_first, chlds_sec):
    # Create construction until there are some childs
    if len(chlds_first) == 0:
        return {}
    # We will now iterate though both childs of both DOM nodes. Each node should be at same index
    # If nodes are not on same index, they are not matching same structure, so they will be ignored
    # Save size of childs for second page
    sec_size= len(chlds_sec)
    for i in range(len(chlds_first)):
        # Iterate thorough all childs on base page. But do not proceed if child with same does not exists on other side.
        if i < sec_size:
            # We need both correlated nodes
            base = chlds_first[i]
            comp = chlds_sec[i]
            # Build path name, and pass it to next level of recursion
            curr_path = path+base.name+'/'
            if base.name == comp.name:
                # If nodes are correlated and of same type, then construct inner Tree recursive
                # Add some additional random value, because JSON does not support attributes of same name on same level
                map[base.name+'_'+str(randint(0,1000))] = construct(curr_path,{},base.findChildren(recursive=False), comp.findChildren(recursive=False))
            if base.name in INTERESTING_TAGS:
                # If current path is interesting, then store this in potential path and display at the end
                potential_paths.append(curr_path)

    # Returning generated matching tree with potential paths
    return map
```

Slika: Konstrukcija drevesa, ter potencialno dobrih poti

```
# Potential tag list.
INTERESTING_TAGS = ['p']
```

Slika: Seznam potencialnih značk, ki nas zanimajo

Ker algoritem vrne strukturo JSON in ker atributi na istem nivoju ne smejo imeti enakega imena, smo poleg značke dodali naključno število, ki poskrbi, da zajamemo vse značke.


```
"Potential_Paths": [
  "div/header/div/details/details-menu/div/div/details/details-dialog/form/div/div/div/div/p/",
  "div/main/div/div/div/div/p/",
  "footer/div/div/div/p/"
]
```


Slika: Potencialne poti, ki vsebujejo podatke


```
{
  "Matching_Construction": {
    "div_673": {
      "a_286": {},
      "div_301": {
        "div_662": {}
      },
      "header_755": {
        "div_986": {
          "a_966": {
            "svg_229": {
              "path_923": {}
            }
          }
        },
```


Slika: Ujemajoči se vzorec spletnih strani

3. Rezultati

Izhodni podatki

Ker je izhodnih podatkov preveč, da bi jih vstavili v poročilo, smo jih shranili v repozitorij (https://github.com/KrajncJ/web-data-extraction), v mapo **outputs**.

4. ZAKLJUČEK

Pri tej seminarski nalogi smo se naučili pridobivati podatke dokumentov na različne načine. Začeli smo z uporabo regularnih izrazov, ki na prvo žogo izgledajo zelo nerazumljivo. Potem, ko smo se poglobili v njihovo delovanje, pa smo videli, da gre za zelo močno in fleksibilno orodje, ki ga lahko uporabljamo skoraj kjerkoli. Sledila je uporaba xpath izrazov, ki so zelo primerni za strukturirane dokumente, takšne kakršni so bili naši html dokumenti iz katerih smo morali pridobiti podatke. Zato je bila njihov uporaba še lažja od regularnih izrazov. Na koncu pa je sledila implementacija približka roadrunner algoritma, s katerim si lahko na podlagi primerjave zgradimo sliko iz kje je smiselno pridobivati podatke. Za seznanitev smo prebrali članeke [1][2]. Nato pa smo napisali svojo poenostavljeno različico roadrunner algoritma.

LITERATURA

[1] http://vldb.org/conf/2001/P109.pdf (Dostopano 7.5.2019)

[2] https://dl.acm.org/citation.cfm?doid=1017460.1017462 (Dostopano 7.5.2019)