خزش سایت های خبری

گروه ۵

شیوا رادمنش، سحر شیخ‌الاسلامی، سیدکسری کرمی‌نژاد، علی نصیری سروی، حامد واعظی

اطلاعات گزارش	چکیده
تاریخ: ۱۳۹۹/۹/۳	در این تکلیف با استفاده از framework محبوب scrapy یک خزنده‌ی برای خزش سایت Truth or Fiction ساخته شده است. و در ادامه پارسری نوشته شده که اطلاعاتی مانند عنوان خبر، آدرس سایت، متن خبر، تاریخ انتشار خبر، برچسب خبر و موضوع خبر را استخراج کرده و در یک فایل .csv می‌نویسد.
واژگان کلیدی: خزنده پارسر spider	

۱- نصب ابزار و تغییر تنظیمات آن

در این پروژه برای خزش وب از scrapy استفاده شده است. Scrapy یک framework محبوب برای خزش وب می‌باشد. برای نصب scrapy از کامند زیر استفاده شده است.

```
$ pip install scrapy
```

سپس برای شروع پروژه از کامند زیر استفاده می‌کنیم.

```
$ scrapy startproject crawler
```

این دستور فایل ها و دایرکتوری‌هایی به صورت زیر ایجاد می‌کند.

scrapy.cfg

crawler/

    __init__.py

items.py

middlewares.py

pipelines.py

settings.py

spiders/

__init__.py

فایل items.py مدل فیلدهایی که قرار است scrape شوند را تعریف می‌کند.

فایل setting.py تنظیماتی مانند crawl agent و crawl delay را تعریف می‌کند.

دایرکتوری spider محل ذخیره‌ی کد scraping و crawling می‌باشد.

همچنین scrapy از scrapy.cfg برای پیکربندی و از pipelines.py برای پردازش فایل‌های scrape شده و middlewares.py استفاده می‌کند.

محتوای فایل‌های items.py و middlewares.py و pipelines.py، محتوای پیش فرضی است که خود scrapy ایجاد کرده است و فولدر پروژه قابل مشاهده می‌باشد.

تنها بخش‌هایی از فایل setting.py را برای پروژه‌ی خود customize کرده ایم:

- در حالت پیش فرض scrapy از یک صف LIFO برای ذخیره‌ی درخواست های معلق شده استفاده می‌کند(جستجوی اول عمق). اگر بخواهیم به صورت جستجوی اول سطح crawling را انجام دهیم، باید تنظیمات زیر را انجام دهیم.

```
DEPTH_PRIORITY = 1
SCHEDULER_DISK_QUEUE = 'scrapy.squeues.PickleFifoDiskQueue'
SCHEDULER_MEMORY_QUEUE = 'scrapy.squeues.FifoMemoryQueue'
```

- برای مودب بودن crawler مقدار download delay را اضافه کردیم تا سرعت crawl کردن کاهش یابد.

```
DOWNLOAD_DELAY = 0.3
```

- برای این‌که مقدار delay همواره یکسان نباشد و pattern ثابتی برای خزش نداشته باشیم (زیرا ممکن است سایت ما را به عنوان بات مخرب ، قطعه کد زیر را نیز اضافه می‌کنیم:

```
RANDOMIZE_DOWNLOAD_DELAY = True
```

- برای پیروی از درخواست‌های سایت مورد نظر در رابطه با نحوه‌ی خزش، ROBOTSTXT_OBEY را true می‌کنیم.

```
ROBOTSTXT_OBEY = True
```

- برای این‌که حین دریافت برخی کدهای html مانند 403 خزشگر تلاش مجدد داشته باشد، قطعه کد زیر را اضافه می‌کنیم:

```
RETRY_TIMES = 4
RETRY_HTTP_CODES = [500, 502, 503, 504, 522, 524, 408, 429, 403]
```

۲- ایجاد یک spider

حال می‌توانیم کد مربوط به crawling و scraping را که با نام spider در scrapy شناخته می‌شود را ایجاد کنیم.

Spider ها کلاس‌هایی هستند که ما تعریف می‌کنیم و scrapy از آنها برای scrape کردن و crawl کردن اطلاعات یک وب سایت یا مجموعه‌ای از وب سایت‌ها(به عنوان seed)، استفاده می‌کند. این کلاس ها باید زیر کلاس scrapy.spider قرار گیرند. و برخی attribute ها و متدهای اولیه را تعریف کنند.

متد ها و attribute های تعریف شده در کلاس spider در این پروژه عبارتند از:

- **name**: رشته‌ای برای شناسایی spider. این مقدار در این پروژه "news" می‌باشد.

```
name = "news"
```

- **متد check_domain**: این متد چک می‌کند که متدهای خارج از دامنه‌ی سایت مورد نظر خزش نشوند.

```
def check_domain(self,link):
        domain = urlparse(link).netloc
        if domain == 'www.truthorfiction.com':
            return True
        return False
```

- **متد start_request**: این متد فیلدی به نام urls دارد که از آن به عنوان seed استفاده می‌شود. مقدار این فیلد برابر با سایت www.truthorfiction.com می‌باشد. و در ابتدا آن

را به یک لیست global به نام seen_links اضافه می‌کند. و متد scrapy.Request را فراخوانی می‌کند.

Scrapy خودش Request Object هایی که این متد بازمی‌گرداند را با توجه به تنظیمات انجام شده زمان بندی می‌کند و با دریافت response و متد callback مربوط به request را فراخوانی می‌کند که این متد در اینجا parse می‌باشد.

```
def start_requests(self):
        urls = [
        'https://www.truthorfiction.com/'
        ]
        for url in urls:
            #this function returns responses from requests of
urls.
            seen_links.add(url)
            yield  scrapy.Request(url=url, callback=self.parse)
```

- **متد parse:** با دریافت response برای هر request این متد فراخوانی می‌شود. و response مورد نظر را به عنوان ورودی می‌گیرد. در اینجا response یک صفحه‌ی html می‌باشد. آن را به متد scrape_data که متعلق به ماژول parser است، پاس می‌دهد.(جزئیات این ماژول در بخش بعدی به طور مفصل توضیح داده شده است). در ادامه یک LinkExtractor تعریف شده است که متعلق به کتابخانه‌ی scrapy می‌باشد. این LinkExtractor صفحه‌ی html (همان response) را به عنوان ورودی می‌گیرد و لینک‌ّای آن را استخراج می‌کند تا عمل خزش روی آنها انجام شود و یک لیست از لینک‌ها بر می‌گرداند. در ادامه این لیست پیمایش می‌شود و اگر یک لینک در دامنه‌ی سایت مورد نظر باشد و قبلا مشاهده نشده باشد، آن را به seen_links اضافه کرده و تابع scrapy.Request را برای آن فراخوانی می‌کنیم.

همانطور که گفته شد، scrapy این Request Object ها را زمانبندی می‌کند و با دریافت response متد parse را برای آن فراخوانی می‌کند.

```
def parse(self, response):
        global seen_links
        scrape_data(str(response.body),response.url)
        le = LinkExtractor()
        links = le.extract_links(response)
        for link in links:
            if not self.check_domain(link.url):
                continue
            elif link.url in seen_links:
                continue
            seen_links.add(link.url)
            #print(link)
            yield scrapy.Request(link.url, callback=self.parse)
```

۳- ماژول پارسر

در سوال از ما خواسته شده است که پارسر مناسب برای هر سایت ایجاد کنیم که اطلاعاتی نظیر «عنوان خبر» و «آدرس سایت» و «متن خبر» را بازیابی کند.

برای این کار از کتابخانه lxml استفاده کرده ایم. این کتابخانه ابزارهایی برای پردازش و parse کردن فایل های xml و html ارائه می دهد.

این کتابخانه حتی توانایی parse کردن فایل های html ای که tag های آن به درستی قرار نگرفته را نیز دارد.

تابع scrape_data با استفاده از این کتابخانه و برای استخراج داده ها از سایت Truth or fiction نوشته شده است.ورودی این تابع یک url است که آدرس صفحه html مورد نظر است و یک رشته که شامل قسمت body آن صفحه html است.

برای ساختن درخت پارسر، از تابع fromstring کتابخانه xml استفاده می کنیم.

برای استفاده از این تابع باید دقت شود که ورودی این تابع page.content است و نه page.tex.

زیرا ورودی این تابع یک حتما باید رشته شامل byte ها باشد

خروجی این تابع نیز تمام فایل html است، که در یک ساختار درختی مرتب شده است.

برای نگهداری مولفه های مختلف، از ساختمان داده dictionary در پایتون استفاده می کنیم.

در قطعه کد زیر، علاوه بر ساخت درخت برای html، آدرس سایت را نیز ذخیره می کنیم.

```
def scrape_data(html, url):
    # print("downloading " + url)
    tree = fromstring(html)
    d = dict()
    d['url'] = url
```

برای طی کردن این درخت دو روش موجود است:

- استفاده از Xpath
- استفاده از CSSSelect

ما در این پروژه از Xpath استفاده کرده ایم. Xpath روشی مناسب برای یافتن اطلاعات در ساختار هایی مانند html و xml می باشد.

با توجه به اینکه همه صفحات خبری در این سایت از الگویی مشخص پیروی می کنند، پس از Xapth برای هر مولفه ی آن، مثلا عنوان خبر، ثابت است. برای پیدا کردن xpath هر مولفه می توان به شکل زیر عمل کرد.

1. ابتدا یک صفحه دلخواه را با استفاده از مرورگر کروم باز می کنیم.
2. با cntrl+I به صفحه inspect هدایت می شویم.
3. روی مولفه دلخواه، مثلا عنوان خبر کلیک می کنیم و tag متناسب با آن را پیدا می کنیم.
4. با کلیک راست کردن، در قسمت copy و copy XPath می توان xpath آن مولفه را بدست آورد.

عنوان خبر:

به طور مثال برای «عنوان خبر»، xpath شکلی مانند زیر دارد.

```
//*[@id="post-125242"]/div/header/div[2]/h1
```

قسمت اولیه، [@id="post-125242"]، شماره post را نشان می دهد که با توجه به اینکه ما صفحات html را به عنوان ورودی به تابع می دهیم، نیازی به شماره post در xpath نیست. همانطور که قطعه کد زیر مشاهده می شود، درخت برگردانده شده با گرفتن xpath مقدار آن مولفه را می دهد.

```
try:
    title = tree.xpath('//div/header/div[2]/h1/text()')[0]
    d['title'] = str(title).strip("'")
```

متن خبر:

برای استخراج متن خبر، xpath را با روش بالا پیدا میکنیم که به شکل زیر است:

```
//*[@id="post-125242"]/div/div/p[1]/text()
```

Xpath بالا مربوط به پاراگراف اول خبر است. برای اینکه بخواهیم کل پاراگراف ها در نظر بگیریم کافی است به جای عدد ، از * استفاده کنیم که شامل همه پاراگراف های ممکن خواهد شد.

```
        try:
            text_list = tree.xpath('//div/div/p[*]/text()')
            text = "\n".join(text_list)
            d['text'] = text
        except:
            d['text'] = None
```

تاریخ خبر:

برای استخراج تاریخ خبر نیز، با روش توضیح داده شده در بالا xpath را پیدا می کنیم که به شکل زیر است:

```
//*[@id="post-125242"]/div/header/div[2]/div/span[3]/span[1]
```

که این مولفه شامل سه قسمت ماه، روز، و سال است که به ترتیب اولین، دومین و سومین داده این مولفه هستند.

```
        try:
            publish_date =
str(tree.xpath('//div/header/div[2]/div/span[3]/span[1]/text()')
[0]).strip().split(" ")
            d['publish month'] = publish_date[0].strip(',')
            d['publish day']  = int(publish_date[1].strip(','))
```

```
            d['publish year'] = int(publish_date[2].strip(','))
        except:
            d['publish year'] = None
            d['publish month'] = None
            d['publish day'] = None
```

نویسنده خبر:

Xpath نویسنده خبر نیز به شکل زیر است:

```
//*[@id="post-125242"]/div/header/div[2]/div/span[2]/a/span
```

این مقدار را نیز نگهداری می کنیم.

```
try:
            author =
tree.xpath('//div/header/div[2]/div/span[2]/a/span/text()')[0]
            d['author'] = str(author)
        except:
            d['author'] = None
```

برچسب خبر:

در این سایت هر خبر یک برچسب دارد،که صحت آن را بررسی می کند. این برچسب میتواند مقدار true یا not_true داشته باشد.

Xpath برچسب خبر نیز به شکل زیر است:

```
//*[@id="post-125243"]/div/div/div[2]/div/span
```

این مقدار را نیز ذخیره می کنیم:

```
         try:
            label =
str(tree.xpath('//div/div/div[2]/div/span/text()')[0])
            d['label'] = label
```

```
        except:
            d['label'] = None
```

موضوع خبر:

در این سایت، هر خبر قسمتی دارد که موضوع خبر را مشخص می کند.این موضوعات شامل Disinformation, reporting, social media, fact checks و . . . هستند.

Xpath این قسمت به شکل زیر است:

```
//*[@id="post-125259"]/div/header/div[2]/div/span[1]/a[1]
```

حال همه اطلاعات مورد نیاز را از صفحه html استخراج کرده ایم، آن را در یک فایل csv ذخیره می کنیم.

به این منظور چک می کنیم که آیا چنین فایلی وجود دارد؟ در صورت وجود پس از باز کردن آن فایل یک سطر مشخصات زیر به آن اضافه می کنیم. در صورتی که چنین فایلی موجود نباشد، باید فایل را ایجاد کرده و header هر ستون را در csv اضافه کنیم. پس از انجام این کار اولین سطر را نیز به فایل csv اضافه می کنیم.

```
columns = ['title', 'url', 'publish day', 'publish month',
'publish year', 'label', 'author', 'subject', 'text']
        file_path =
os.path.join(file_parent_dir,'output',file_name)
        # file_path =
os.path.join(os.path.abspath(__file__),'output',file_name)
        file_exists = os.path.isfile(file_path)

        # if not os.path.exists('output'):
        if not
```

```
os.path.exists(os.path.join(file_parent_dir,'output')):
            # os.mkdir('output')
            os.mkdir(os.path.join(file_parent_dir,'output'))
        with open(file_path, 'a+', newline='') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=columns)
            if not file_exists:
                writer.writeheader()
            writer.writerow(d)
```

۴- Run کردن spider

در حالت عادی که از پارسر کتابخانه‌ی scrapy استفاده شود، می‌توان با استفاده از کامند زیر spider مورد نظر را run کرد.

```
$ scrapy crawl <spider name>
```

اما در اینجا چون متد parse در این کلاس، customize شده است و خودمان آن را نوشته‌ایم. به همین دلیل از قطعه کد زیر در spider برای اجرای آن استفاده شده است. این کد ابتدا یک crawlerProcess با مشخصات تعیین شده برای user-agent ایجاد می‌کند و سپس spider تعریف شده توسط خودمان را به آن داده و در نهایت آن process را start می‌کنیم.

```
if __name__ == '__main__':

    process = CrawlerProcess({
        'USER_AGENT': 'Mozilla/4.0 (compatible; MSIE 7.0;
Windows NT 5.1)'
    })

    process.crawl(NewsSpider)
    process.start()  # the script will block here until the
crawling is finished
```

برخی اطلاعات اضافی کسب شده حین دیباگ

در اولین ران spider پس از crawl کردن حدودا ۲۰۰۰ صفحه خبری به timeout برخوردیم. راه حل ما برای رفع کردن این مشکل استفاده از delay در scrapy بود. اما در طی این مسیر برخی راه حل های جالب دیگر نیز پیدا کردیم که چون مشکل ما با delay حل شد نیازی به تست کردن آن‌ها نبود اما این راه حل‌ها را ثبت می‌کنیم:

۱- یکی از راه‌های هندل کردن این error تغییر user agent می‌باشد. user agent پیش فرض scrapy عبارت است از "Scrapy/{version}(+http://scrapy.org)". آن را مطابق آنچه در ادامه نشان داده شده است تغییر دادیم.

```
'USER_AGENT': 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)'
```

۲- یکی دیگر از راه حل‌های حل این error به کمک exception handling می‌باشد. برای این‌که بتوانیم در scrapy از این موضوع استفاده کنیم، باید یک تابع errback برای Request مان تعریف کنیم تا زمانی که به error خوردیم این تابع فراخوانی شده و در آن بتوانیم نحوه هندل کردن error را پیاده سازی کنیم.(که در این مورد ارور ما می‌توان در آن تابع تا زمانی که دوباره ارتباط برقرار شود، retry کنیم)

۳- یکی دیگر از راه حل ها استفاده از caching می‌باشد. می‌توان با اضافه کردن این خط به فایل settings.py از این قابلیت بهره برد:

```
DOWNLOADER_MIDDLEWARES = {
'scrapy.contrib.downloadermiddleware.httpcache.HttpCacheMiddlewa
re': 300, }
```