Homework 3

Edward Hernández

College of William & Mary

Homework 3

Dependency Resolution

This document is a literate Python program written inset within a LaTeX document, woven and executed using Pweave and typeset using pdfTeX

This code is meant for use under Python 3, and is tested only under version 3.5.1. It uses BeautifulSoup and nltk to analyze corpora built using Scrapy ($\geq$1.1.0rc1).[1]

Resolving the dependencies for this code are non-trivial. If you have a Python distribution installed already, you may not want to attempt to alter it to accomodate this code, as its dependencies are somewhat unstable. The best solution is virtualization. The following Dockerfile is sufficient to create a suitable environment. The same is trivially easy to achieve with virtualenv.[2]

```
FROM ubuntu
RUN apt-get update &&                       \
        apt-get install -y                  \
                python3                     \
                python3-pip                 \
                python3-bs4                 \
                python3-twisted-experimental \
                python3-w3lib               \
                libxml2-dev                 \
                libxslt1-dev                \
                libz-dev
RUN pip3 install scrapy==1.1.0rc1
```

This Dockerfile installs the following dependencies, against which the code is run on Ubuntu 14.04:

[1] Any other version of Scrapy will likely break this code. Scrapy's stable release does not support Python 3, and much of this code is likely not valid for that release.

[2] Under OS X, you may still have an issue with libxml. It can usually be solved by (re)installing Xcode. If you cannot satisfy the libxml dependency, then virtualenv is not a possible solution, and Docker is likely your only recourse.

```
PAM==0.4.2
PyDispatcher==2.0.5
Scrapy==1.1.0rc1
Twisted==13.2.0
attrs==15.2.0
beautifulsoup4==4.2.1
chardet==2.2.1
colorama==0.2.5
cssselect==0.9.1
html5lib==0.999
lxml==3.3.3
parsel==1.0.1
pyOpenSSL==0.13
pyasn1==0.1.9
pyasn1-modules==0.0.8
pyserial==2.6
queuelib==1.4.2
requests==2.2.1
service-identity==16.0.0
six==1.5.2
urllib3==1.7.1
w3lib==1.14.2
wheel==0.24.0
zope.interface==4.0.5
```

Building Corpora with Scrapy

Import stuff:

```
from scrapy import signals, Spider, Item, Field, Request
from scrapy.crawler import Crawler
from scrapy.exporters import XmlItemExporter, JsonLinesItemExporter
from scrapy.loader import ItemLoader
from scrapy.loader.processors import Join, MapCompose, TakeFirst
from scrapy.settings import Settings
from scrapy.utils import log
from twisted.internet import reactor

from bs4 import BeautifulSoup
from w3lib.html import remove_tags
import re
```

```
<class 'ImportError'>
No module named 'bs4'
```

Scrapy provides an `Item` class which is used to collect scraped data. The following code defines an `Item` class to hold information about stories. Each chapter of each story in the corpus will be stored as a separate text (`body`) with a `title`, an `author`, a description or summary (`desc`), the topic or `theme` of the story, its `page` or chapter number within a larger work, the `site` it was scraped from, and the `url` from which it was scraped.

```
class StoryItem(Item):
    title  = Field()
    author = Field()
    desc   = Field()
    body   = Field()
    url    = Field()
    site   = Field()
    theme  = Field()
    page   = Field()
```

Define how to load the above item with raw data from the sites.

```
class StoryItemLoader(ItemLoader):
    default_input_processor  = MapCompose(str.strip)
    default_output_processor = TakeFirst()
    body_in  = MapCompose(remove_tags)
    body_out = Join()

class FFItemLoader(StoryItemLoader):
    # desc_out = Join()
    desc_in  = MapCompose(remove_tags)
    desc_out = Join()

class AOItemLoader(StoryItemLoader):
    body_out  = Join()
    theme_out = Join(', ')
```

```
<class 'NameError'>
name 'remove_tags' is not defined
```

Pipe the scraped text into Json items on the lines of a file (one file per spider).

```
class JsonLinesExportPipeline(object):
    def __init__(self):
        self.files = {}

    @classmethod
    def from_crawler(cls, crawler):
        pipeline = cls()
        crawler.signals.connect(pipeline.spider_opened,
signals.spider_opened)
        crawler.signals.connect(pipeline.spider_closed,
signals.spider_closed)
        return pipeline

    def spider_opened(self, spider):
        file = open('%s_products.jl' % spider.name, 'w+b')
```

```
            self.files[spider] = file
            self.exporter = JsonLinesItemExporter(file)
            self.exporter.start_exporting()

        def spider_closed(self, spider):
            self.exporter.finish_exporting()
            file = self.files.pop(spider)
            file.close()

        def process_item(self, item, spider):
            self.exporter.export_item(item)
            return item
```

A spider to scrape `fanfiction.net`:

```
    class FFSpider(Spider):
        name = "ff"
        allowed_domains = ["fanfiction.net"]
        # start_urls = [
        #     "https://www.fanfiction.net/%s/" % c for c in
        #     (
        #         'anime',
        #         'book',
        #         'cartoon',
        #         'comic',
        #         'game',
        #         'misc',
        #         'movie',
        #         'play',
        #         'tv'
        #     )
        # ]
        start_urls = [
            "https://www.fanfiction.net/misc/"
        ]

        def parse(self, response):
            for href in response.xpath('//td[@valign="TOP"]/div/a/@href'):
                url = response.urljoin(href.extract())
                yield Request(url, callback = self.parse_tag)

        def parse_tag(self, response):
            for href in response.xpath('//div/center[1]/a[contains(text(),
    "Next")]/@href'):
                url = response.urljoin(href.extract())
                yield Request(url, callback = self.parse_tag)

            for href in
    response.xpath('//div[contains(@class,"z-list")]/a[1]/@href'):
                long_url = response.urljoin(href.extract())
                url      = '/'.join(long_url.split('/')[0:-1])
                yield Request(url, callback = self.parse_story)

        def parse_story(self, response):
            header  = response.xpath('//span[@class="xgray
    xcontrast_txt"]/text()')
            head    = header.extract()[1]
            chapter = int(response.url.split('/')[-1])
            more    = re.search('Chapters: [0-9]*', head)
            if more and chapter == 1:
                chapters = int(more.group(0).split()[1])
                base_url = '/'.join(response.url.split('/')[0:-1])
                urls = [
                    base_url + '/' +  str(x) for x in range(2, chapters+1)
                ]
                for url in urls:
                    yield Request(url, callback = self.parse_story)

            loader = FFItemLoader(StoryItem(), response=response)
            loader.add_xpath('title', '//*[@id="profile_top"]/b/text()')
            loader.add_xpath('author',
    '//*[@id="profile_top"]/a[1]/text()')
            loader.add_xpath('desc', '//*[@id="profile_top"]/div')
            loader.add_xpath('body', '//*[@id="storytext"]')
            loader.add_value('url', response.url)
            loader.add_value('site', 'fanfiction.net')
            loader.add_value('theme', '')
            loader.add_value('page', str(chapter))
            yield loader.load_item()
```

Spider for Literotica:

```
class LESpider(Spider):
    name = "le"
    allowed_domains = ["literotica.com"]
    start_urls = [
        "https://www.literotica.com/c/%s/1-page" % c for c in
        (
            'adult-how-to',
            'adult-humor',
            'adult-romance',
            'anal-sex-stories',
            'bdsm-stories',
            'bdsm-stories',
            'celebrity-stories',
            'chain-stories',
            'erotic-couplings',
            'erotic-horror',
            'erotic-letters',
            'erotic-novels',
            'erotic-poetry',
            'exhibitionist-voyeur',
            'fetish-stories',
            'first-time-sex-stories',
            'gay-sex-stories',
            'group-sex-stories',
            'illustrated-erotic-fiction',
            'interracial-erotic-fiction',
            'lesbian-sex-stories',
            'loving-wives',
            'masturbation-stories',
            'mature-sex',
            'mind-control',
            'non-consent-stories',
            'non-erotic-poetry',
            'non-erotic-stories',
            'non-human-stories',
            'reviews-and-essays',
            'science-fiction-fantasy',
            'taboo-sex-stories',
            'transsexuals-crossdressers'
        )
    ]

    def parse(self, response):
        for href in
response.xpath('//*[@id="content"]/div/div/h3/a/@href'):
            url = response.urljoin(href.extract())
            yield Request(url, callback=self.parse_story)
        for href in response.xpath('//*[@class="b-pager-
next"]/@href'):
            url = response.urljoin(href.extract())
            yield Request(url, callback=self.parse)

    def parse_story(self, response):
        loader = StoryItemLoader(StoryItem(), response=response)
        loader.add_xpath('title', '//h1/text()')
        loader.add_xpath('author',
'//*[@id="content"]/div[2]/span[1]/a/text()')
        loader.add_value('desc', '')
        loader.add_xpath('theme',
('//*[@id="content"]/div[1]/a/text()'))
        loader.add_xpath('body', '//*[@id="content"]/div[3]/div')
        loader.add_value('url', response.url)
        loader.add_value('site', 'literotica.com')
        loader.add_xpath('page', '//*[@class="b-pager-
active"]/text()')
        yield loader.load_item()

        for href in response.xpath('//*[@class="b-pager-
next"]/@href'):
            url = response.urljoin(href.extract())
            yield Request(url, callback=self.parse_story)
```

Spider for AO3:

```
class AOSpider(Spider):
    name = "ao"
    allewed_domains = ["archiveofourown.org"]
    start_urls = [
        "https://archiveofourown.org/media"
        #
"http://archiveofourown.org/works/6508453/chapters/14893933?view_full_work=true&vi
```

```
    ]

    def parse(self, response):
        for href in response.xpath('//h3/a/@href'):
            url = response.urljoin(href.extract())
            yield Request(url, callback=self.parse_genre)

    def parse_genre(self, response):
        for href in response.xpath('//li/ul/li/a/@href'):
            url = response.urljoin(href.extract())
            yield Request(url, callback=self.parse_tag)



    def parse_tag(self, response):
        for href in
response.xpath('(//ol[@role="navigation"])[1]/li[last()]/a/@href'):
            url = response.urljoin(href.extract())
            yield Request(url, callback=self.parse_tag)

        for href in response.xpath('//h4/a[1]/@href'):
            url = response.urljoin(href.extract()) +
'?view_full_work=true&view_adult=true'
            yield Request(url, callback=self.parse_story)

    def parse_story(self, response):
        loader = AOItemLoader(StoryItem(), response=response)
        loader.add_xpath('title', '//h2/text()')
        loader.add_xpath('author', '//a[@rel="author"]/text()')
        loader.add_xpath('desc', '(//*[@class="summary
module"])[1]//p/text()')
        loader.add_xpath('body', '//*[@id="chapters"]//div/p/text()')
        loader.add_value('url', response.url)
        loader.add_value('site', 'archiveofourown.org')
        loader.add_xpath('theme', '//dd[@class="fandom
tags"]//a/text()')
        yield loader.load_item()
```

I haven't yet figured out what signal the spiders should send to avoid shutting down the reactor (which cannot be restarted) before all three are finished (if they are all run together.

```
# callback fired when the spider is closed
def callback(spider, reason):
    stats = spider.crawler.stats.get_stats()  # collect/log stats?

    # stop the reactor
    reactor.stop()

# instantiate settings and provide a custom configuration
settings = Settings()
settings.set(
    'ITEM_PIPELINES', {
        '__main__.JsonLinesExportPipeline': 100,
    }
)
# settings.set('USER_AGENT', 'Mozilla/5.0 (Windows NT 6.3; Win64;
x64)')

# instantiate a spider
ff_spider = FFSpider()
le_spider = LESpider()
ao_spider = AOSpider()

# instantiate a crawler passing in settings
# crawler = Crawler(settings)
ff_crawler = Crawler(ff_spider, settings)
le_crawler = Crawler(le_spider, settings)
```

```
ao_crawler = Crawler(ao_spider, settings)
# configure signals
ff_crawler.signals.connect(callback, signal=signals.spider_closed)
le_crawler.signals.connect(callback, signal=signals.spider_closed)
ao_crawler.signals.connect(callback, signal=signals.spider_closed)

# configure and start the crawler
# crawler.configure()
# crawler.crawl(spider)
ff_crawler.crawl()
# le_crawler.crawl()
# ao_crawler.crawl()

# start logging
# log.start()
log.configure_logging()

# start the reactor (blocks execution)
reactor.run()
```

Analysis with NLTK