Homework 3

Edward Hernández
College of William & Mary

Homework 3

The Program

This document is a literate Python program inset within a LaTeX document, woven and executed using Pweave and typeset using pdfTeX This code is written in/for Python 3, and is intended for and tested under only version 3.5.1. It uses nltk to analyze corpora built using Scrapy (≥1.1.0rc1).[1]

Resolving the dependencies for this code is non-trivial. Ideally, pip should be able to handle the dependencies:

```
pip3 install                \
        bs4                 \
        chatterbot          \
        python-Levenshtein  \
        nltk                \
        vaderSentiment      \
        scrapy==1.1.0rc1
```

However, if you are running OS X, pip will likely fail building lxml, since OS X does not ship libxml by default. This can usually be solved by (re)installing the Xcode command line tools: `xcode-select install`. However, this is not always successful. Additionally, if you have a Python distribution installed already, you may not want to attempt to alter it to accomodate this program, especially if you make use of a stable version of any of its dependencies.

In either case, the best solution is probably virtualization. The following Dockerfile is minimally sufficient to create a suitable environment.[2]

```
FROM ubuntu:xenial
ENV LC_ALL C.UTF-8
```

[1] Scrapy's stable release (1.0.5) does not support Python 3, so this program currently runs against only development versions after 1.1.0rc1. All code is tested to run against 1.1.0rc1 but is primarily run against 1.2.0dev2, the current development branch on github.

[2] Virtualenv is another good solution to avoid altering your Python installation, but cannot be used to satisfy the dependence on libxml, since it manages only Python-internal dependencies.

```
RUN apt-get update &&                       \
        apt-get install -y                  \
                python3                     \
                python3-pip                 \
                python3-boto                \
                python3-cookies             \
                python3-cssselect           \
                python3-bs4                 \
                python3-future              \
                python3-fuzzywuzzy          \
                python3-levenshtein         \
                python3-lxml                \
                python3-nltk                \
                python3-responses           \
                python3-requests-oauthlib   \
                python3-pydispatch          \
                python3-pymongo             \
                python3-queuelib            \
                python3-twisted             \
                python3-w3lib
RUN pip3 install          \
        chatterbot        \
        scrapy==1.1.0rc1  \
        vaderSentiment
CMD pip3 freeze
```

This Dockerfile installs the following dependencies, against which the code has been tested to run on Ubuntu 16.04:

```
attrs==15.2.0
beautifulsoup4==4.4.1
```

```
blinker==1.3
boto==2.38.0
chardet==2.3.0
ChatterBot==0.3.6
cookies==2.2.1
cryptography==1.2.3
cssselect==0.9.1
funcsigs==0.4
future==0.15.2
fuzzywuzzy==0.10.0
html5lib==0.999
idna==2.0
jsondatabase==0.1.1
lxml==3.5.0
mock==1.3.0
nltk==3.1
numpy==1.11.0
oauthlib==1.0.3
PAM==0.4.2
parsel==1.0.1
pbr==1.8.0
pyasn1==0.1.9
pyasn1-modules==0.0.7
PyDispatcher==2.0.5
PyJWT==1.3.0
pymongo==3.2.2
pyOpenSSL==0.15.1
pyserial==3.0.1
python-forecastio==1.3.4
python-Levenshtein==0.12.0
```

```
python-twitter==3.0rc1
queuelib==1.1.1
requests==2.9.1
requests-oauthlib==0.4.0
responses==0.3.0
Scrapy==1.1.0rc1
service-identity==16.0.0
six==1.10.0
textblob==0.11.1
Twisted==16.0.0
urllib3==1.13.1
vaderSentiment==0.5
w3lib==1.11.0
zope.interface==4.1.3
```

Building Corpora with Scrapy

Import stuff:

```
from scrapy import signals, Spider, Item, Field, Request
from scrapy.crawler import Crawler
from scrapy.exporters import 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 chatterbot import ChatBot
from w3lib.html import remove_tags
import re
import json
```

Scrapy provides an `Item` class which is used to collect scraped data. The following code defines an `StoryItem` 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()
```

Scrapy provides an API by which data can be loaded into an `Item` via an `ItemLoader`. The code below specifies a `StoryItemLoader` (which inherits the `ItemLoader` class defined by Scrapy) to load story data generally and two classes which inherit it, to load particular data from specific websites.

In cases where both the `StoryItemLoader` and one of the child classes specify a field (e.g. `body_out`), the child class supersedes the parent.

These loaders cannot be written before the `Spiders` defined below, as they operate on the output of those Spiders (and thus depend necessarily for their design on the design of those Spiders). They are defined here only because they must exist to be called by the Spiders.

```
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(', ')
```

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_stories.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'
        )
    ]

    def parse(self, response):
        tags = response.xpath(
            '//td[@valign="TOP"]/div/a/@href'
```

```
        )
        for href in tags:
            url = response.urljoin(href.extract())
            yield Request(url, callback = self.parse_tag)

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

        stories = response.xpath(
            '//div[contains(@class,"z-list")]/a[1]/@href'
        )
        for href in stories:
            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):
        next = response.xpath(
            '//*[@class="b-pager-next"]/@href'
        )
        for href in next:
            url = response.urljoin(href.extract())
            yield Request(url, callback=self.parse)

        stories = response.xpath(
            '//*[@id="content"]/div/div/h3/a/@href'
        )
        for href in stories:
            url = response.urljoin(href.extract())
            yield Request(url, callback=self.parse_story)

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

```
            yield Request(url, callback=self.parse_story)


        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()
```

Spider for AO3:

```
class AOSpider(Spider):
    name = "ao"
```

```
    allowed_domains = ["archiveofourown.org"]
    start_urls = [
        "https://archiveofourown.org/media"
    ]

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

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


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

        stories = response.xpath('//h4/a[1]/@href')
        for href in stories:
```

```
            extension = '?view_full_work=true&view_adult=true'
            url = response.urljoin(href.extract()) + extension
            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

Training a ChatterBot

```
chatbot = ChatBot(
    "Mary Sue",
    io_adapter = "chatterbot.adapters.io.NoOutputAdapter"
)
# chatbot.train("chatterbot.corpus.english")

file = open('le_products.jl')
```

```
for line in file.readlines():
    conversation = []
    story = json.loads(line)
    body = story['body']
    graphs = body.split('\n')
    for graph in graphs:
        speech = graph.split('"')[1::2]
        utterance = ' '.join(speech)
        if utterance:
            conversation.append(utterance)
    if conversation:
        chatbot.train(conversation)

print(
    chatbot.get_response("Hello")
)
```

Unfortunately, once the chatbot has been trained on a decent amount of data, it becomes unusably slow. This is almost certainly a result of its storage method (a flat database stored to disk) and the nature of its implementation (in Python).