MEng Final Project
Super Seminar Scrapper

Advisor - David Bindel, Student - Batu Inal [bi49]
Cornell University

Abstract

Every major university runs seminars and colloquia, and data about speakers, titles, and abstracts is often posted online. The topics and speakers often say a great deal both about the venue and about what is hot in the field in general; but while the information is available "in the wild", it is not in a form that is amenable to machine parsing. This project will address the issue of scraping colloquium information from top departments into a consistent, machine-readable format in order to support data mining.

1 Introduction

1.1 Previous Work

This is a project that has been carried out, under the supervision of David Bindel, by four students up-to-date; Jiankun Lu, Jing Jing, Patrick Chen and most currently Batu Inal . The contributions from the previous three students range in different ways; thus it is one of the objectives of the project to glue these pieces together to create a unified functioning system. A data-driven website has been built by Jiankun Lu, using JavaScript and HTML. The front-end handles the data parsing by Angular.js and the animations by jQuery. Once implemented, the back-end will process the data into a machine-readable format in order to support data mining from the websites and write the processed data to a file called alltalks.json which will be requested by the Angular.js http on the user's browser. One of the contributors, Jing Jing has created API-like Python modules that use Machine-Learning techniques such as k-means clustering, Logistic Regression with Gradient Descent and Latent Dirichlet Allocation to identify the latent structures within a list of seminars or colloquiums through document clustering and topic modeling. In specific, to differentiate colloquiums from seminars, cluster seminars into different groups based on their subareas and extract the common topic words among colloquiums or seminars with the same subarea.

1.2 Current Work

As a basic task, we will create scrapers for the CS colloquia at Cornell, Berkeley, CMU, UW, and UIUC. We believe this is a diverse enough set of data formats to give us some feel for what would have to go into a more general framework. This is the primary task, and we believe it will be completed by early or mid April. We will then incorporate the scraped data feeds into an existing web front-end in order to get a complete system to display and query these colloquia in a consistent manner. This task is aimed to be done before the end of the semester, May 2016. If time permits, we will explore three additional tasks: a more general framework for building scrapers for new seminars and colloquia, a framework to disambiguate speaker information in order to tell (for example) whether the same speaker is giving job talks at multiple institutions and integrating CS colloquia at MIT and(or) Stanford to the aforementioned system.

2 Implementation

2.1 Main Module

Since the main task is to retrieve the data from university colloquia pages, we chose the process of crawling websites utilizing the Scrappy[3] library in python. After we identified the colloquia sites of the 5 universities[1], we crawled these sites in parallel. According to the response url we handled each university separately, scrapping the HTML by utilizing the xpath function from the HtmlResponse library in the Scrappy.HTTP module. Once a pattern that uniquely identifies the page is found by the programmer (for e.g; the page is in table format thus extracting all the features in /table[2]/tr/td using xpath lets us get relevant data in a list format), the relevant content is extracted. The relevant information may either be a list of extracted nodes from the plain html or .ical urls which contains all the relevant information for the colloquia in the respective university. If the retrieved content is from a plain HTML, rather than an .ical file, the pattern is to create an entry for every element containing a date. Though; we do realize that not every date might represent a colloquia, we have the system initially assume all dates are colloquia entries. Filtering out the non-colloquia entries, in a later refinement-process, seems to work as the best heuristic. Then, once again the plain-HTML response is filtered, this time for the information on the speaker, topic and url of the colloquia. Afterwards, the correct entries are inserted under the correct dates within the list created. The dummy dates that got through our initial filtering are identified; thus deleted from the main list. If the extracted content is an .ical url. We simply parse the perfectly aligned .ical file using the icalendar library.

CourseWebPageCrawler.py The structure of the main module is as follows:

```
def parse(self, response):
        import cornell
        import ical
        #Cornell Handler (plain-html example)
        if (response.url == shared.cornell_url):
            #Extracts Dates of the Colloqiuas
            content = response.xpath("//h2").extract()
            #Builds the correct_size-Vector with so_far known info and date_handler
            shared.date_handler('cornell', content, "Day", 1, "", "", "",
            "G01 Gates Hall - Mentor's Lecture Hall", "Cornell University", "", "", "")
            #Extracts the speaker name, the topic and the url of the speaker
            cornell.speaker_topic_url_handler(response.xpath("//h4").extract())
            #Cleans the vector of dummy elements
            cornell.clean_no_colloqiums()
            #Send Processed Records to be Written to .json file
            shared.add_to_all_records(cornell.records)
        #CMU Handler (.ical example)
        elif (response.url == shared.cmu_url):
            #get possible urls for.ical links
```

```
        possible_links = response.xpath('//div[@class="calendar-admin-menu"]').extract()
        #get the .ical file
        link = ical.get_calendar_url(possible_links, 0, 'https://www.scs.cmu.edu')
        #get the contents of the .ical file
        content = ical.get_content(link, 0, 'cmu')
        #create a list of records from the data
        cmu.create_records(content)
        #Send Processed Records to be Written to .json file
        shared.add_to_all_records(cmu.records[1:])
```

2.2 Finding Common Patterns-Shared, iCal

While creating scrapers for the CS colloquia sites we tried creating a general framework that could be further extended and easily adjusted in the future for other schools, perhaps even different projects. In order to create the for-mentioned general framework, along with the structure of the main-module described in the section above, we have created two shared classes (Shared.py and ical.py), which are utilized while crawling university sites. The two classes can be summarized as the following:

Shared.py The shared class acts like an API to parse the plain-html response, from any crawled website, create records, with correct meta-data, for each colloquia and write to a shared JSON file "alltalks.json". Using the class one can; check whether a "date" exists in the plain text, then create a record for a colloquia with the correctly parsed/created date(in the form of datetime python object). The class is also utilized to strip meta-data from the plain-text such as the abstract, speaker, topic and insert into the correct record with the correct datetime object.

Below are some examples of the functions from shared.py:

```
"""
Given an element from the html response;
extracts, creates and returns the correct date

Args:
    date_type (str): Month or Day of the Week found in element
    elt (str): The plain-html element from response
    dist (int): The offset from the searched element
    typ (str): <Day> or <Month> The date_type that is en-grained in the element

Returns:
    datetime: The found date in the element in a datetime object format
"""
def extract_date(date_type, elt, dist, typ):

"""
Given an element from the html response and an element to find,
```

```
    Finds the given element, then advances index by the specified index, building
    a return string until the delimeter character is met.

    Args:
        to_find (str): The element to be searched
        elt (str): The plain-html element from response
        dist (int): The offset from the searched element, serves to eliminate the tag.
        for e.g. if the tag was <xyz> we would have dist=5, to get past the tag
        delimiter (str): A delimiter character, indicating the end of the element
                default (str): If to_find is not found, the return value
        is_URL (bool): If the element to be retrieved is an URL set to true.
        reverse (bool): Set to true if plain-text needs to be searched from reverse

    Returns:
        str: the element found otherwise returns the default value specified
    """
    def retrieve_element(to_find, elt, dist, delimeter, default, is_URL, reverse):

    """
    Given a needle within the haystack
    returns the index of the nth occurrence of the element within the haystack

    Args:
        haystack (str): The plain-HTML element from response
        needle (str): The element to be retrieved
        n (int): Occurrence number

    Returns:
        int: the index of the nth occurrence of the element within the haystack
    """
    def find_nth(haystack, needle, n):

    """
    Given a file name, writes to the specified
    file under the "data" directory in JSON format

    Args:
        fileName (str): File to write to
    """
    def writeToFile(fileName):
```

```
    #Given the correct meta-data as input creates a record
    def create_record(Topic, Speaker, Time, Venue, University, URL, Description, Tags):
    #Adds the record to the list of records for the correct University
    def add_to_all_records(records):
```

ical.py Once the user has identified that the website contains an .ical file, the user may use this API to extract data from .ical files and create the according records.

There are two main functions in the class:

```
    """
    Retrieves the URL of the .ical file

    Args:
        content ([str]): list of URLs in string format
        index (str): The index of the correct URL within list
beg (str): String to append before the URL (for e.g http://), else leave ""

    Returns:
        str: a URL with <beg> appended to the front
    """

    def get_calendar_url(content, index, beg):
    """
    Given a .ical URL retrieves the relevant data by parsing the encoded string
    and returning an instance of the corresponding Python type
    Args:
        URL (str): .ical URL
        broken (bool): If the .ical url is broken, set to true
        class_name (str): If broken is set, call the .ical fixer
        from relevant school's class

    Returns:
        Calendar: A Calendar object where meta-data is stored in a struct-like object.
    """
    def get_content(URL, broken, class_name):
```

Some .ical files may be uploaded in a broken format on-purpose, for crawlers to not easily extract data. However, our API has a work around for the issue; where if the .ical file is malformed, a customized fixer is called from the class of the correct University. The fixer, reads the malformed Calendar file byte-by-byte and writes it onto a new file, while fixing the broken parts. Once the data from the newly created file is used to create new records, the file is removed.

3 Discussion

We believe that the steps to create a robust and flexible general-framework to scrape colloquia sites has been established. In order to make the API more modular and robust, we plan to revise the code from scratch while also creating a documentation for future use. The front-end has been successfully connected with the newly-created back-end scrapper; however, in order to maintain the front-end functionality, some of the hard-coded features on the front-end will need to be revised. In order to facilitate the process, one of the previous authors, Jiankun Lu, will be contacted on minor modifications to the front-end.

The current system is yet to be connected with the Machine Learning modules implemented by Jing Jing, it is one of our goals to integrate these modules to our system in order to better analyze and understand the data from colloquia sites. Thus, the framework could be used to disambiguate speaker information in order to tell (for example) whether the same speaker is giving job talks at multiple institutions.

If time permits, we also plan on integrating the CS colloquia at other schools (primarily MIT and(or) Stanford) to the aforementioned system using the API's created; to be able to understand how to add an $i'th$ random-school to the system without much effort in the future.

References

[1] University Colloquia Sites Crawled *Cornell University- https://www.cs.cornell.edu/events/colloquium, University of Illinois- Urbana Champaign-http://cse.illinois.edu/news-events/seminars, UCal.Berkeley- http://www.eecs.berkeley.edu/Colloquium/, University of Washington- https://www.cs.washington.edu/events/colloquia, Carnegie Mellon University- https://www.scs.cmu.edu/calendar*

[2] David Bindel, Jinakun Lu, Batu Inal "Super Seminar Scrapper" *https://github.com/jl-git/talkstream*. GitHub Page for Super Seminar Scrapper Project

[3] Scrapy *An open source and collaborative framework for extracting the data you need from websites. In a fast, simple, yet extensible way.* http://scrapy.org/