# Information Retrieval

COMP 479 Project 4 Report

Lin Ling (40153877)

A report submitted in partial fulfilment of the requirements of Comp479.

Concordia University

# INDEX

## 1. Crawl and scrape web pages

Web crawling is the method of traversing through the World Wide Web to download information related to a particular topic. For this project, the topic is to use the text of web pages to cluster the documents using k-mean and assign a sentiment score to each cluster. It is always a good practice to refer to the web pages' robot exclusion before attempting to crawl the page.

Source code: concordia_crawler.py

To put the spider to work, go to the project's top level directory and run:

scrapy  runspider concordia_crawler.py

```
(base) D:\0.Information Retrieval\Project\P4\Test1>scrapy runspider concordia_crawler.py
```

a. Crawling tools: Scrapy

Spiders are classes that I define and that Scrapy uses to scrape information from a website (or a group of websites). They must subclass Spider and define the initial requests to make.

As you can see, the Spider subclass scrapy. spiders and defines some attributes and methods:

- name: "concordia"
- file_limit: 50
- start_urls:  'https://www.concordia.ca/ginacody.html'
- rules:

  Link Extractor:  is an object that extracts links from response.

  Parameters: deny: a list of regular expressions that the (absolute) urls must match to be excluded (i.e., not extracted).

```python
rules = (
    Rule(
        LinkExtractor(
            deny=(
                r'^(?!https://www.concordia.ca).+',  # stay in concordia domain
                r'^(https://www.concordia.ca/fr).+',  # don't go to french page
                r'([A-z]+=[0-9]+)$',  # don't go over pages 9 (empty pages)
                r'([A-z]+=[A-z]+)$',  # don't click on filters (e.g sort=title)
            ),
        ),
        callback='parse_item',
        follow=True),
)
```

b. Obey the standard for robot exclusion

Scrapy shell command does respect robots.txt configuration defined in setting.py

Scrapy respects the robots.txt file by default (ROBOTSTXT_OBEY = True).

```python
# obey robots.txt and robots meta tags
custom_settings = {
    "ROBOTSTXT_OBEY": True,
    "ROBOTS_META_TAG_OBEY": True,
}
```

Since there is no robot.txt for Concordia domain, the exclusion is in a meta tag, so it will check the robot content to obey the standard.

Line 37: gets the robot content

Line 44: check the content and scrapping and download the web which is allowed to crawl.

```python
33     def parse_item(self, response):
34         if self.docID >= int(self.file_limit):
35             raise CloseSpider("Limit reached.")
36
37         self.robot_content = response.xpath("//meta[@name='robots']/@content").extract_first()
38
39         url = response.url
40         if url in self.visited_urls:
41             self.logger.info(f'Already scrapped: {url}')
42             yield
43
44         if self.robot_content == "" or self.robot_content == "index,follow":
45             self.logger.info(f'Scrapping doc #{self.docID + 1}: {url}')
46             filename = response.url.split("/")[-1]
47             with open("testData/" + filename, 'wb') as f:
48                 f.write(response.body)
49
50         self.docID += 1
51         self.visited_urls.add((url, self.docID))
52
```

c. Set upper bound on the total number of files to be downloaded

Line 34: If the docID is bigger than file limit (which is set up to 50 at concordia spider class)

Line 35: When the total number of files to be downloaded reaches the file limit, The spider will be closed.

Line 45-48: save the HTML files in folder testData for testing and debugging.

## 2. Cluster the document collection

Source code: extract_html_text.py

a. Extract the text from web pages

Read the files from folder testData and use BeautifulSoup to extract the text from the files and store in a list named documents for document collection.

b. Use scikit learn to cluster the document collection

This project uses TfidfVectorizer as text vectorizer. TdidfVectorizer uses an in-memory vocabulary (a Python dict) to map the most frequent words to features indices and hence compute a word occurrence frequency(sparse) matrix. The word frequencies are then reweighted using the Inverse Document Frequency (IDF) vector collected feature-wise over the corpus.

c. Two different clustering runs

    i. k = 3

```
n_samples: 64, n_features: 4748
Top terms per cluster:
Cluster 0: school student concordia servic academ calendar graduat univers art scienc studi campu class centr colleg cours event resourc link health
Cluster 1: tuition loan fee school scholarship servic financi student concordia budget bank bursari 2022 academ rate univers aid calendar pay account
Cluster 2: school concordia servic student calendar scienc graduat academ univers art studi class comput colleg research 2022 gina codi engin news
```

Print out 20 index terms for each cluster that most informative, from the information we could generate a name for each cluster.

Cluster 0: Service and Calendar

Cluster 1: loan and bursary

Cluster 2: Campus news

Output file: 3_clusters.txt

    ii. k = 6

```
n_samples: 64, n_features: 4748
Top terms per cluster:
Cluster 0: dec p.m. 2 3 4 school 1 oral - apprentic exam phd concordia servic student a.m. day yr. 14 wednesday
Cluster 1: school student concordia servic calendar academ graduat univers scienc art studi class colleg campu event cours centr resourc gina codi
Cluster 2: engin softwar school comput scienc beng student option csse cours section servic program bachelor concordia elect calendar sequenc gina codi
Cluster 3: concordia 2022 school citi research student servic 2021 calendar institut scienc ’ univers novemb studi campu art graduat academ sustain
Cluster 4: cybersecur network consortium cyber canadian concordia ” “ univers innov debbabi research lead train school director economi mourad ncc not-for-profit
Cluster 5: dr. system network machin process antenna model wireless signal control analysi imag electr energi ece video circuit algorithm electromagnet comput
```

Print out 20 index terms for each cluster that most informative, from the information we could generate a name for each cluster.

Cluster 0: Exam Phd

Cluster 1: Service and Calendar

Cluster 2: Software and Computer Science option

Cluster 3: Citi research

Cluster 4: Cybersecurity

Cluster 5: network machine

Output file: 6_clusters.txt

### 3. Sentiment analysis and derive cluster sentiment scores

The AFINN lexicon is a list of English terms manually rated for valence with an integer between -5 (negative) and +5 (positive) by Finn Årup Nielsen between 2009 and 2011. For the cluster sentiment scores, write a function to collect the detailed scores of each document in the cluster, and get the total sentiment scores and average sentiment score which is total sentiment scores divided the number of documents in the cluster.

Function and the result shows as follows:

```python
# derive cluster sentiment scores
def calculate_afinn_score(docs, index_list):
    total_afinn_score = 0
    average_afinn_score = 0
    afinn_scores = []
    for index in index_list:
        afinn_score = afinn.score(docs[index])
        total_afinn_score = total_afinn_score + afinn_score
        afinn_scores.append(afinn_score)
        average_afinn_score = total_afinn_score / len(index_list)
    return total_afinn_score, average_afinn_score, afinn_scores
```

```
(288.0, 144.0, [157.0, 131.0])
(454.0, 45.4, [77.0, 43.0, 38.0, 43.0, 36.0, 56.0, 73.0, 34.0, 41.0, 13.0])
(866.0, 66.61538461538461, [111.0, 62.0, 52.0, 105.0, 72.0, 81.0, 65.0, 47.0, 52.0, 61.0, 62.0, 62.0, 34.0])
(2934.0, 75.23076923076923, [37.0, 98.0, 66.0, 82.0, 90.0, 60.0, 73.0, 61.0, 48.0, 63.0, 137.0, 43.0, 135.0, 62.0, 70.0, 86.0, 184.0, 74.0, 54.0, 11
(365.0, 73.0, [100.0, 78.0, 64.0, 66.0, 57.0])
(707.0, 88.375, [53.0, 71.0, 59.0, 114.0, 98.0, 76.0, 173.0, 63.0])
```

First is total afinn score, second is average afinn score and rest is the list of each documents afinn scores.