# Report for Review Crawler

**Background**

With the rapid development of information technology, data is generated every day, anytime and anywhere. For example, there are plenty of hot topics on social network platforms like Twitter and Facebook, which attract people to exchange their views. If we can make good use of the data, we can get valuable information. In machine learning and data mining, data is the basic thing which could be used to train and test a model to do some predictions. Web crawler, which is an Internet bot that systematically browses the World Wide Web, typically for the purpose of Web indexing is widely used to collect the specific and meaningful data from the websites.

In this project, I implemented two web crawlers to grabs the whole hotel list on Hotels.com (HK) and corresponding comments from customers. Hotels.com is the world's leading provider of hotel accommodation services, provides reservation services through its own network of localized websites. This project also limited the scope to hotels and their comments in Hong Kong, China. The result data set will be used for further research about hotel industry in Hong Kong, China.

This project is based on a certain development environment, which is shown as below:

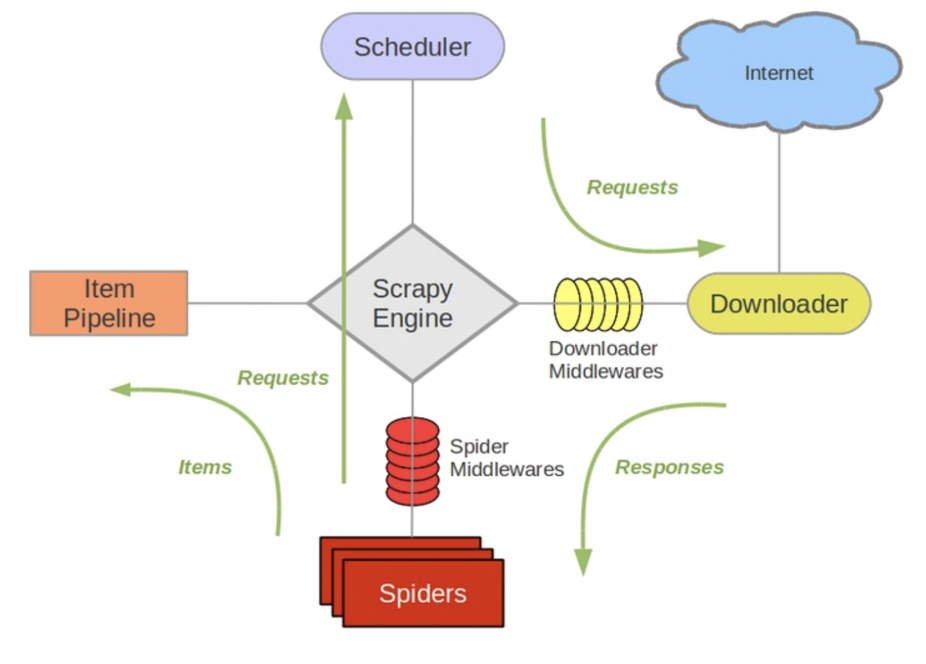
* Python 3.7.4
* Scrapy 1.7.3
* Pandas 0.25.2

**Scrapy Framework**

Web scraping is being used as a means for gathering and analyzing data across the web. To support this process, there have been numerous frameworks that have come up to satisfy different requirements for various use-cases. After a brief survey, I choose Scrapy as the framework. Scrapy is a free and open-source web-crawling framework written in Python. Originally designed for web scraping, it can also be used to extract data using APIs or as a general-purpose web crawler.

Scrapy project architecture is built around "spiders", which are self-contained crawlers that are given a set of instructions. Following the spirit of other don't repeat yourself frameworks, such as Django, it makes it easier to build and scale large crawling projects by allowing developers to reuse their code. Scrapy also provides a web-crawling shell, which can be used by developers to test their assumptions on a site’s behavior.

Here is a diagram of the architecture of Scrapy.



Scrapy Engine is responsible for controlling the flow of data in all components of the system and triggering events when corresponding actions occur. This component is equivalent to the brain of the reptile. The scheduler receives requests from the engine and queues them so that they are provided to the engine later when it requests them. The initial crawling URL and the subsequent URL to be crawled in the page will be put into the scheduler, waiting for crawling. At the same time, the scheduler will automatically remove duplicate URLs (if a specific URL does not need to be de duplicated, it can also be implemented by setting, such as the URL requested by post). The Downloader is responsible for obtaining the page data and providing it to the engine, and then providing it to the spider. Spider is a class written by a scrapy user to analyze a response and extract an item (that is, the item obtained) or an additional follow-up URL. Each spider is responsible for handling a specific (or some) website. Item pipeline is responsible for processing items extracted by spider. Typical processes are cleanup, validation, and persistence (such as access to a database). After the page is stored in the item by the crawler, it will be sent to the project pipeline, processed in several specific orders, and finally stored in the local file or database.

The process of a task is the combination of those The general work flow of Scrapy is:

1. The engine opens a domain, finds the spider that processes the website and sends a request about the first URL (s) to be crawled from the spider;
2. The engine gets the first URL to crawl from the spider and schedules it by Request in the scheduler.
3. The engine requests the scheduler for the next URL to crawl.
4. The scheduler returns the next URL to be crawled to the engine, which forwards the URL to the downloader through the download middleware (request direction).
5. Once the page is downloaded, the downloader generates a response for the page and sends it to the engine through the download middleware (response direction).
6. The engine receives the response from the downloader and sends it to spider for processing through spider middleware (input direction).
7. Spider processes the response and returns the crawled item and the (followed up) new request to the engine.
8. The engine gives the crawled item (returned by spider) to the item pipeline and the request (returned by spider) to the scheduler.
9. Repeat from step 2 until there are no more requests in the scheduler, and the engine shuts down the site.

**Implementation**

**Hotel Crawler**

The first crawler aims to collect all the Hong Kong hotels in Hotels.com. The start of the crawler is a certain URL from which the crawler begins to collect data. If you select Hong Kong as your destination and do not set a limit to the other search condition on the search-form, then the page will jump to a list of all the hotels. The output file should contain the list of hotels, every item of the list should contain a hotel id and a hotel name in order to get the comments.

During the process of developing this crawler, there is a problem that the hotel list is dynamic loaded through JavaScript and there are not displayed pagination buttons. I used the built-in developer tools in Chrome to analysis the action about how the HTTP request is generated and what form it has. Then I find every time when a request is sent, the URL contains a parameter which represents the page number of the list the request wants to get. Therefore, I defined a global variable that add one every time before call the HTTP request for the next page.

**Comments Crawler**

Comments crawler aims to collect all the comments of every hotel. The hotel list is provided by the Hotel Crawler as json in advance. The difficulty of this crawler is that we have to control whether to turn page of comments or to change to another hotel. I used the exception mechanism and try to turn page first. If the result is an error or an exception, it will be catch and turn to another hotel. When I first wrote code to implement, I missed the point that the Python keyword yield will call the multithreaded working mode automatically, causing unexpected output. This could be confusing at first, especially when I didn’t realize the special mode.

**Result and Validation**

In the end, I crawled 195,589 comments from 1319 hotels and saved them into json. The item in comment file contains the hotel name, rating score, rating badge, description and comment date. As for validation, I didn’t come up with a good way to test whether the result is completely correct, so I just sampled from the website manually and compare the comment detail with what I crawled. The project is managed by Github at <https://github.com/fitwill/MSBD5014.git>.

**Future Work**

The crawlers still cannot grab reviews periodically. A solution to this problem is using crond to run a timed task in Linux. Also, the former comments crawler should be modified to only update the new comments.

**Appendix**

1st meeting

Date: 3rd October

Place: LG1, Library

People: Dr. JIAN Xun, ZHANG Zihan

Approval: Approved

Discussion: 1. The framework to use. 2. The fundamental development demand. 3. Website to crawl.

2nd meeting

Date: 14th October

Place: LG1, Library

People: Dr. JIAN Xun, ZHANG Zihan

Approval: Approved

Discussion: 1. Show the learning output of Scrapy. 2. Problem solving. 3. Next step plan.

3rd meeting

Date: 6th November

Place: LG1, Library

People: Dr. JIAN Xun, ZHANG Zihan

Approval: Approved

Discussion: 1. Discuss the problems Zihan met in implementation process. 2. Finish grabbing the hotel and begin to grab comments.

4th meeting

Date: 4th December

Place: Online (Due to the special arrangement of campus)

People: Dr. JIAN Xun, ZHANG Zihan

Approval: Approved

Discussion: 1. Discuss the result of the project. 2. Discuss the future work.