Course Project

1 Introduction

The following document aims to illustrate and explain some of the details of TiNews, my tool. In the next sections, I explain some of the design choices, websites crawled, as well as some architectural choices.

To get a better overview of how to run the tool, please read the `README` embedded in the project folder.

Before you go any further, if not yet done, it is recommended to download the repository at `https://github.com/pennal/TiNews`

2 Components

2.1 Crawling

The first step was to obtain the information required for the tool. To do so, I used Apache Nutch. The configuration was left quite standard, only for a few additions that are explained in the next sections. In general, the number of rounds was left at 5, which I found to be a rather acceptable parameter. This allowed me to gather enough information to have a working tool.

Crawling was done independently for each of the websites. This was done to parallelize the work, and therefore speed up the process. Also, this allows me to keep an eye on exactly what comes from where, and have a better overview on the size of the crawling process.

Once done, the segments for each of the websites were taken and dumped to an html file. This made it easy to extract the information needed from each of the websites. This is better explained in the next section.

2.1.1 Ticino Online

I started by using one of the obvious ones. The structure of the website is quite simple, and easy to crawl. The only issue I encountered were the external links that are embedded in almost every page through the use of widgets. One of the external websites hosted by Tio and operated by them is Tuttojob, a platform for people that are seeking jobs. In the end, I modified the regex to exclude any website that did not start with the core domain tio.ch.

2.1.2 Corriere del Ticino

Unfortunately, this website does not allow bots to crawl their content, unless you are Google. Therefore, I had to exclude it from my list.

2.1.3 Il Caffé

Il Caffé was another easy choice, and its structure is also very simple to crawl. Luckily, the website is quite self contained, and does not have any links to external pages like Ticino Online. I anyways modified the regexes to only follow URLs that were on the same domain, as to reduce the search space.

2.1.4 Ticinonews

Ticinonews was a bit of a last minute addition. Same as for all the other websites, I forced the crawler to follow only links which were on the same root domain. In the end, I was able to retrieve quite a few documents.

2.2 Filtering

Instead of relying on Nutch to do the heavy lifting, I decided to write an external program that would ingest the content, filter it and extract the required content. To do so, I take the plain text dumps from the crawling phase, parse them, and then depending on the website that the content belongs to, extract the information using an HTML parser.

To do so, I implemented a strategy pattern, where each website must implement an interface that I declared. This interface contains methods that are common to all, such as one for getting the title, getting the content, etc. and delegates the actual implementation to another class, hiding the implementation from the user of the system.

Once such filtering is performed, the data is dumped into a unique JSON, which contains all valid articles crawled by Nutch and filtered by my utility. For each of the article, the following fields are present:

- Title: Title of the article
- Content: Main content of the article
- URL: URL of the article
- ID: Unique id required by Solr, and implemented using a simple hash on the content

These fields are then sent to Solr, which takes care of indexing the data, and storing it. This is done using the `post` command provided by Solr (for more details, see the `README`).

2.3 Solr

To index and serve the content, I used Apache Solr. After the filtering phase, the data is sent to Solr which is used as a simple web server which server the content based on the user query. To send the data to Solr, I used the data obtained from the filtering phase which was cleaned and included all the data I needed.

2.4 Frontend

In order to present the results to the user, I created a very simple webpage with a single field on it. Once the user has inserted the query, the data is sent over so Solr which returns the documents related to the search. The results are then shown to the user in a very Google-like style, allowing them to click in the result and get taken to the page of the article, independently from the source.

The design of the page was left quite simplistic on purpose. In the end, I decided to use what Bootstrap calls “cards” which are essentially containers. This way, the user has the impression of unrelated stories, where each result is split into its own “component”. It is also made very clear that the title of a website can be clicked, which will take the user to the desired page.

To implement the frontend, I used basic HTML and CSS, with some JavaScript to make the requests to Solr. Other than that, I used the Bootstrap framework to make the website as responsive as possible, as well as jQuery for DOM manipulation, FontAwesome for the icons, and Handlebars for the template generation.

3 Querying

As previously explained, I used Solr to host all of the information. Luckily there is a build in query engine that allows us to obtain the information even if stored in custom fields. Therefore, I used the normal query URL, simply plugging in the information required by the user. A modification was made to the parameters, in such a way that the title was more important than the content itself. This is because most news website tend to give a summary of the content with a few keywords as the title, and it is quite easy to find information. Therefore, I boosted the importance of the title, but left the importance of the content as default. The end URL looks as follows:

```
http://localhost:8983/solr/nutch/select?defType=edismax&df=content&indent=on
&q=<QUERY_WORDS>&qf=title^2%20content&rows=100&wt=json
```

Solr will then return a payload as JSON, which can be easily traversed and the required information extracted.

4 Running the tool

In order to repeat the entire process, please check the `README` file.

5 Evaluation

In order to evaluate the tool, I asked a few classmates to look for articles that were both recent and not. No further instructions were given, in order to see how user friendly the tool is. All that was asked was to talk during the evaluation, and to narrate any doubt or questions.

The positive impressions were as follows:

- Simple to use UI
- Results are coherent with the search query
- The highlighting of the terms was a nice addition
- The system was responsive, never hanging too much in the query phase

On the other hand, there were a few remarks:

- The results were mostly from one site (Ticinonline)
  The reason behind this is quite simple. Tio is the leading web based news website in Ticino, and tends to have a few hundred articles per day. The other two websites are still quite important, but Il Caffé tends to deliver most of its content in PDF and Ticinonews simply does not have the dimensions to compete with the other two
- The date would have been nice to see
  Unfortunately, this is an issue in how the different sites implement this feature. For example Tio tends to write something along the lines of "2 weeks ago" instead of the actual date, for recent article. This does not give any clear indication for the actual date, and therefore I decided to leave out this piece of information.
- The content text is not expandable on the results page
  This is due to a limitation in time, and could be interesting to implement.

6 Conclusion

In conclusion, I am quite happy with the tool. It performs as expected, and the results are quite accurate. If I were to do it again, I probably would spend more time in the crawling phase to really optimise this part, both in speed and accuracy.