Report
Information Retrieval
Apache Nutch
2014 - 15

Nostradamus

Team Members :

Arkanath Pathak
Pramesh Gupta
Pranjal Pandey
Sanyam Agarwal
Nevin Valsaraj
Aseem Patni
Sabyasachee Baruah

INTRODUCTION

This project was undertaken as part of completion of the course curriculum of Information Retrieval, CS60092. We extend our acknowledgements to Professor Sudeshna Sarkar for giving us the opportunity to work on this project. We also thank our teaching assistants Ayan Das, Yetesh Chaudhary and Abhik Jana.

As part of the initial work we were asked to choose among Dspace, Nutch and IBM Watson. After looking through each of them and analysing their pros and cons, we chose Apache Nutch to perform our project on. The reason behind was the much easier integration with our backend database and user control over the query settings. Dspace required the user to specify the keywords along with the data that has been submitted. That made it a bad option for indexing products in general where we come across thousands of parameters. The domain space of the data was also limited. With Nutch we could not only leave this technicalities to be handled by Solr, but at the same time we could impose out own tf-idf settings to the scoring which was helpful in sentiment analysis. We did not choose IBM Watson as it required to be trained and the training data could only be chosen from specific fields that did not match our requirements.

OBJECTIVE

The aim of our project is to make a product search engine and return the sentiment score of the products for the users to view. We have crawled data from the website boschtools.com that contains a detailed catalog, along with user reviews, of power tools and accessories. Each of the returned results contain links to their official product page of the main website.

The rest of the report has been divided into the following sections:

I. **Crawling** : Here we describe the steps we followed to crawl data with the help of Nutch and store it in Solr
II. **Storage and Indexing**: Here we show the usage of Apache Solr and the basic frontend API that it provides.
III. **Search**: Here we show the use of Solr's eDismax feature, and the use of NLP libraries in our python scripts.
IV. **Semantic Analysis**: Here we show the use of Semantria
V. **Frontend**: Here we show the the frontend GUI features
VI. **Working**: Here we show how our query is processed in all the above stages and the result comes back to us as a score, review and link tuple.
VII. **Conclusion**: Our conclusion of the project and what we learnt from it.

I. CRAWLING - Apache Nutch

We have used **Nutch** 1.9 for our purpose and have followed the installation instructions as given in the following tutorial. The **boschtools.com** website arranges their products as categories that have each under them a collection of products. To crawl only the category and product pages we used the following regular expressions in our Nutch configuration files -

- *http://www.boschtools.com/Products/Tools/Pages/BoschProductDetail.aspx(\?)pid=(.+)*$*
- *http://www.boschtools.com/Products/Tools/Pages/BoschProductCategory.aspx(\?)catid=(.+)*$*
- *http://www.boschtools.com/Products/Tools/Pages/ItemResults.aspx(\?)catid=(.+)*$*

Apache Nutch also lets us specify a depth and iteration number for the crawl command that was set to 100 each allowing us to crawl as many as 254 product pages. Depth here means the level of URL links to which the crawl goes and iteration number means the number of rounds the crawler goes through the page.

Another change in settings that was made was in the url-filters file where consecutive three forward slashes was ignored by default in the url. This option was disabled as boschtools.com urls did not follow this constraint.

Apart from changing the proxy port and ip in the xml files we increased the content.limit size values from 64 KB to 64000 KB.

One shortcoming we faced while working with the website boschtools.com was that Nutch was not able to extract the review section for the products. The reason behind this was the use of AJAX that sent an asynchronous POST request to the server for getting the user review and ratings. As such we had to resort to the use of python scripts and use mysql in the backend just for the storage of reviews that will be used later for sentiment score. We followed a simple schema of pid, review and score as our attributes, with pid serving as our primary key.

The extracted data were stored in Apache Solr, which we discuss next.

II. STORAGE and INDEXING - Apache Solr

Apache Nutch is made to work along with Apache Solr. We have used Apache Solr 4.0 for our purpose and have followed this tutorial. Solr is an open source search platform built on Apache Lucene. Data is stored in Solr in the form of collections that can be unloaded and reloaded at will. Added to this, Solr provided its own basic query syntax and a number of query parsers of which we have used **eDismax**. We will discuss eDismax in the next section. For now we will discuss about the meaning of different fields we came across while working with Solr.

- **q**

It is the main query text field. Here we can specify queries in the form of *attribute : content* where attribute can be field keywords like id, content, title, etc and content is a regular expression.

- **fq**

fq stands for filter query and is used to restrict the set of documents returned for query. This has no effect on the score returned.

- **sort**

Solr returns set of documents for the query specified in the **q** field. The order is on the basis of score that Solr assigns. By default it is set to desc but we can also set it to asc.

- **start**

It indicates the offset in the complete result set.

- **rows**

It indicates the maximum number of documents returned from the result set.

- **fl**

We can specify the set of fields whose information we want to see.

- **df**

It is the default field and only takes effect when the **qf** field is not specified and left blank

- **wxt**

We can specify the format we want to see the returned result set for e.g.. python, html, csv etc.

In addition to these Solr provides **core Admin** and **Logging** features that we can use to manage, modify and delete our collections as well as to view exceptions and errors respectively. Each of the collections again gives us a variety of options like **Analysis, Ping, Schema** and **Plugins** which are for advanced usage and were not of much use for us in the present project**.**

III. SEARCH - Apache Solr and NLP

Here we discuss the NLP libraries that we used for our query preprocessing stage and the use of eDismax, a feature of Solr that returned us the result set of out project.

We first used **NLTK**, a suite of natural language processing libraries designed for the python programming language. We initially input a query string and it goes through the following phases :

- **Stopword Removal**

We remove common stop words like *“if”,”a”,”is”,”the”*, etc.

- **Tokenisation**

Users while inputting the names of products might not be sure of the market name. As such it will be a common occurrence to come by descriptive words which is essential to be tokenised to get the correct score.

- **Synonyms**

This is a feature we tested out that gave us better results than in its absence. Instead of picking up a single query term we pick a related set of words for each token we get after the tokenisation phase and provide it as additional input to Solr’s eDismax plugin.

Now we turn our attention to Solr’s eDismax. eDismax stands for extended maximum disjunction. We have a **qf** or query field in eDismax that we have set to *“id title^6 content^2”* for our purpose. The working is simple. The user query that comes as an output from the python script after the preprocessing phase is fed into the q field of Solr. What eDismax does is that it tries to find all the query terms in all the fields of **qf** which in our case is id, title and content. Now we have already noted above that Solr has its own scoring method. Here is where eDismax scores above the basic query parser. We can input our own weights to the various fields allowing zoning to be incorporated into out project. So a word appearing in the title is 3 times (= 6/2) as more “*important*” as when it appears in the content body.

However for the purpose of our project we also had to include sentiment scores as well which we did with the help of Semantria.

IV. SEMANTIC ANALYSIS - Semantria

Semantria is a **cloud-based API** that returns a model sentiment score for input text in the range [-1,1]. The first 20,000 transactions are free given a registered secret and key number. However being cloud-based using it when the query was presented from the user at run time would slow done our software's response time. As such we used our own backend mysql database that was mentioned earlier.

The number of products we had crawled was static and the software was not be used for commercial purposes. As such we stored the sentiment scores of each of the products before hand in our tables.

The sentiment score of Semantria was halved and then added to the score Solr returned. The intuitive behind this was to give more relevance to Solr's scoring schema as sentiment score is only computed on user reviews and not on actual content.

V. FRONTEND

We used **AJAX** request in the client side to request search results from the server, which responds with **JSON-P** response.
The search results were filtered on the client side according to the category,
The response included all the product details including the Solr score and the score calculated by semantic analysis.
Another kind of query was the query for product details. The response was all the product details including the reviews and the score for semantic analysis for the reviews.

VI. WORKING

The user first inputs query term(s) describing the product he wants to search for. The query string is first parsed by a python script which removes stop words, tokenises the string and converts the free text into a set of synonymous words. This is fed to Apache Solr.

In Solr each of these words is fed separately to the fields we have specified in the **q** field. Sold returns a result set along with the score for each of the results. Now for each of these results we lookup the sentiment score in our database and add it to our Solr score and return the whole as a json body to our front end.

At the front end the json gets parsed and depending upon the user filter settings and sort order the products are displayed along with their views. In case the user wants to navigate to the official product page we have also included the link for that. The user also has an option to turn off sentiment analysis in the case when he or she is only interested to browse through the products for the given query.

VII. CONCLUSIONS

All in all the project has been an enlightening experience for us as we got introduced to Nutch, Solr, Semantria and also got to use some NLP libraries. We were introduced to the stages of how a query is processed till the stage when results are returned. Most importantly we uncovered some shortcomings of Apache Nutch in processing AJAX elements which will prove useful in future.

We also realise that we can farther extend our project and software in many aspects. Firstly, we can include a more detailed usage of the various parsing engines that Solr provides.

Secondly, we can farther improve our scoring method to give different weights to different zones. We have already implemented a prototype of it and can improve upon it.

Thirdly, we can build upon Apache Nutch. Nutch like Solr provides a host of configuration options in its xml files that needs to be tweaked to crawl different webpages and sites.

We conclude by stating once again our acknowledgements to the aforementioned faculty members as well as to our whole team without whose combined effort this project would not have been possible.