SugConf: A Conference Classification and Meta Extraction System

Mandar Rahurkar and Maryam Karimzadehgian
http://130.126.122.126:8080/sugconf4.jsp

Categories and Subject Descriptors: [Web Information Retrieval SVM Language Model]: Project Report

Additional Key Words and Phrases: web, retrieval, SVM, Language Model

Authors’ address: University of Illinois at Urbana-Champaign.
© University of Illinois at Urbana-Champaign

CS498-CXZ Introduction to Text Information Systems
Contents

1 Introduction 1

2 System: Training and Testing 2

3 Entity Extraction 3

4 Classification 4
   4.1 SVM: Support Vector Machines 4
   4.2 Language Model 4

5 Results 5

6 Conclusion 6

7 Implementation 7
1. INTRODUCTION

The rapid growth of the World Wide Web in recent years has created a vast repository of information. It is estimated that the Internet contains at least 100 terabytes of text. The popular search engine Google indexes about 9 terabytes of the Web to perform its search. The crawler is used to retrieve webpage’s systematically and pages which are updated frequently are marked. However considering the amount of data to be crawled, it is not possible to sample a large subset of web, with high dynamic content, at high enough rate to acquire the updated content. This which may be detrimental for several applications. Another problem that we have is most of the text is unstructured, meaning unlike the structured database simple entities like name and address have to be identified and extracted using machine learning techniques.

The recall of a web search engine, i.e. the ratio of the number of relevant pages returned to the number of all relevant pages on the web, is typically rather low. Another problem is low precision, which is the ratio of the number of relevant answers to the number of all answers. This is due to the fact that the keywords specified by the user may occur in several contexts and most of them may be irrelevant to the user. Consequently, often a large number of irrelevant web pages are returned with a relatively small number of relevant pages. Though this is not important for general search search applications as long as first few pages are relevant, this may not be true for all applications. Focused crawlers overcome the above drawbacks of web search engines, i.e. they yield good recall as well as good precision by restricting themselves to a limited domain [Ester et al. 2001]. Large-scale, topic-specific information gatherers are called focused crawlers [Aggarwal et al. 2001; Chakrabarti et al. 1999; Diligenti et al. 2000]. In contrast to normal crawlers which must process large portions of the Web in a centralized manner, a distributed federation of focused crawlers can cover specialized topics in more depth and keep the crawl fresher, because there is less to cover for each crawler. In its simplest form, a focused crawler consists of a supervised topic classifier controlling the priority of the unvisited frontier of a crawler. The classifier is trained a priori on document samples embedded in topic taxonomy. This set is called seed set in focused crawlers. It thereby learns to label new documents as belonging to topics in the given taxonomy [Chakrabarti et al. 1998]. The goal of the focused crawler is to start from nodes relevant to a focus topic in the Web graph and explore links to selectively collect pages about that topic.

However, as in our case of collecting all conference pages, there is no reason to believe that one conference page will lead you to another and so on. Thus even if you start with some seed of conference pages probability that you will end up with a large set of conference documents is very less. Another approach that we propose is to improve the search itself by identifying the relevant documents in a search generated by relevant queries, and use these documents to further identify more relevant query-words. These classified relevant documents may also be used as feedback. Without loss of generality we propose to use this framework to solve the problem of conference mining and entity or meta-data extraction from conference home pages.

This report is organized as follows. Section 2 introduces the system and describes
its components leading to feature extraction. In the following section we talk about the Entity or meta-tag extraction algorithm. In section 4 we discuss the classification algorithms that we used, results of which are elaborated in section 5. Implementation details are provided in section 7 followed by some screen shots of the system.

2. SYSTEM: TRAINING AND TESTING

![System Block Diagram](image1)

Overall functioning of the system has been shown in figure 1. Once the web content has been retrieved it is processed to convert the noisy HTML data into feature vectors. Thus as shown in the figure 2, during first stage of preprocessing html tags are removed. In HTML documents it is not mandatory for an opening tag to have a closing tag. For eg., `<h1>` need not be closed with `<h1>` for heading1 formatting. Also the presence of nested tasks makes the task of parsing an HTML page non-trivial. We used the publicly available library for html parsing¹. Let \( T = \{p, H1, H2, H3, Title\} \) be a set of relevant html tags. Text in each of the

¹http://htmlparser.sourceforge.net/
fields has different degree of importance. Since author of the web page has felt the need to attach more significance to these words, our weighing method is justified. At next stage all extracted text is indexed under for all fields in set \( T \) using Lucene\(^2\).

For feature representation we use the vector-space model. We use bag of words approach, and the phrases or tokens in vector space represent the weight for each word in the given document. Each document is represented as a feature vector of size \(|V|\), where \( V \) is vocabulary. Vocabulary or dictionary was created by combining together all the unique words from positive and negative training set. \(|V|\) is about 13000 words. Each point in the feature vector is the weight assigned to respective word in the vocabulary. Eg., \( F_d = f_1 \ldots f_{|V|} \) where \( f_i \) represents the weight assigned to \( i \)th word in \( V \) for document \( d \). Thus the words which are in vocabulary but not in the document get assigned the weight value ‘0’. We used the weighting function in eqn. 2.

\[
tf(w, d) = \sum_{field \in T} Wt(field)c(w, d, field) \forall field \in T
\]

\[
f(w, d) = \frac{tf(w, d)}{\log(df(w) + 1)}
\]

where \( tf(w, d) \) and \( df(w, d) \) represent the term frequency and document frequency respectively for word \( w \) in document \( d \) in the data set. \( c(w, d, field) \) is count of word \( w \) for field \( \in T \) in document \( d \). Weight for each field was assigned empirically. During the training phase these features are used to generate a classifier using SVM. During testing phase web-search for given query is done using a popular search engine and retrieved contents are processed in a similar manner as described in the training phase. The features are then used to classify a page as conference page or not.

3. ENTITY EXTRACTION

Extracting entities from the collection of text is a well researched problem of Named Entity identification. However even extracting common name entities like date is not very fruitful. We experimented with an open source toolkit\(^3\) to extract dates from the processed web-content, however it was computationally very expensive. Even with the computational expense, the results were discouraging. We observed that URL’s pointing to these entities themselves are very characteristic. We exploit this to our benefit to instead identify URL’s which point to relevant entities rather than entities itself. Meta information which we extract is:

— Important Dates.
— Venue of the Conference.
— Workshops related to the conference.
— Tutorials related to the conference.
— Registration Information.

\(^2\)http://lucene.apache.org/

\(^3\)http://opennlp.sourceforge.net/
For a given query we have a queue of URL’s returned by the search engine. We process each URL in this search result queue in a sequential manner. The problem that we face here is that content of page pointed to by given URL, even if a conference page, might not contain the relevant meta data. This happens especially when URL points to a page which used frames. To deal with this situation we crawl each URL in a search result queue up to a depth of 2 and build another queue of URL’s obtained by crawling a URL from search result queue. We check for binary presence of keywords in these URL’s. If a URL is not a match for any of the above mentioned relevant meta data categories it will be discarded. Mapping between meta data categories and URL is one to one. Duplicate URL’s are removed by maintaining URL queues in hash table where hash key is a function of URL string. Thus duplicate URL’s will generate the same hash key and will be overwritten. We keep track of number of meta tags found in each search result URL. This information as described later is used for ranking.

4. CLASSIFICATION

4.1 SVM: Support Vector Machines

Support Vector Machines have been shown to work very well for text classification tasks and hence we decided to try SVM’s as our first approach. Inherently SVM’s try to find a linear classifier to maximize the margin between the examples of positive and negative data samples. Optimizing the parameters of SVM is arduous task. We experimented with using polynomial functions and RBF kernels. The results are presented in the Section 5.

4.2 Language Model

We used CMU’s statistical language modeling toolkit\(^4\) to generate language model. Perplexity was the distance measure used to compare the web-page content to determine if a given page is conference related page or not. Perplexity is information theoretic measure which roughly speaking gives the distance between two distributions. Consider sample points for \(X, Y = (y_1 \ldots y_N)\). Likelihood for \(X\) can be written as:

\[
\log(L(Y)) = \sum_{i=1}^{N} \log(P(X = y_i))
\]

Thus perplexity is defined as:

\[
2^{-\frac{\log(L(Y))}{N}}
\]

The threshold for cutoff was determined by using the development set which was independent of the training set. In order to allow words not seen in training set to occur with non-zero probability discounting was used. We experimented with Absolute Discounting, Witten Bell Discounting [Witten and Bell 1991], Good Turing Estimate [Katz 1987] and Linear discounting [Ney et al. 1994]. The results using each of these methods are presented in the Section 5.

\(^4\)http://mi.eng.cam.ac.uk/~prc14/toolkit.html
5. RESULTS

For building a training set, around 100 conference URL’s were manually obtained and content of these URL’s is used as positive samples. To obtain negative samples, content of 100 URL’s spread over web search for arbitrary queries like “fruits”, “dance” etc were saved. Development set for positive samples was built over manually selected 25 conference URL’s and negative results were obtained similarly as for the training set but with different queries. Training and development set did not have any overlap.

We did not get very good results with SVM. Percentage Accuracy on test set was 92% whilst on development set accuracy did not exceed 36%. One of the reasons for bad performance could be lack of discounting. We believed that with training data we have, most of the conference related words should be in the dictionary and if these words are absent in the test page, it is unlikely to be a conference related page.

Results using language model are summarized in table I. Column 2 and 3 give the average perplexity computed for the positive samples and negative samples respectively in the development set. Thus these numbers can be used to determine the threshold for deciding if a given page conference related page or not. However instead of using hard thresholding we deciding to do soft thresholding in a way to rank the documents. Let $NT$ be number of tag entries found in a given page and $P$ be its computed perplexity. Overall score for a given document $d$ is computed as in equation 3. Idea is we would like to have less perplexity and more number of entity matches; with entity matches being the more dominant factor. Perplexity is ensured to be always greater than 1. Adding 5 in the denominator and 1 is the numerator is to give it a sort of discounting flavor. As seen in table I the difference between the average perplexity is very large especially when using Witten Bell smoothing which demonstrates a good discriminative capability.

$$score(d) = \log(P) + 1 \over NT^3 + 5$$  \hspace{1cm} (3)

<table>
<thead>
<tr>
<th>Discounting Method</th>
<th>Average Positive Perplexity</th>
<th>Average Negative Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>608</td>
<td>2810</td>
</tr>
<tr>
<td>Absolute</td>
<td>594.97</td>
<td>3995.58</td>
</tr>
<tr>
<td>Witten Bell</td>
<td>696.28</td>
<td>3408.71</td>
</tr>
<tr>
<td>Good Turing</td>
<td>516.46</td>
<td>2453.26</td>
</tr>
</tbody>
</table>

Table I. Results from Using Language Model

Some of top occurring n-grams are:

- Call for Papers 136
- are not limited 16
- include but are 15
- The conference will 14
- Call for Participation 13

...
Thus these phrases can be used as query words to search for more conference pages and thereby increase the recall rate.

6. CONCLUSION

A real time system to identify conference related pages and extract entities was created and tested. It was tested for several queries and found to be working effectively. The system is currently not multi-threaded and hence multiple queries cannot be issued at one given time. Some errors are introduced due to ineffective html parsing. It seems domain specific stemming would play a vital role in improving the results. Portal stemmer was tried however it transformed ‘conference’ into its stem ‘confer’ which is not a good idea. Some of the things that we would like to do in future would to use mixture modeling or/and transductive learning.

REFERENCES


7. IMPLEMENTATION

We implemented following parts during the course of our project:

— HTML parser using the publicly available library. (JAVA)
— Creating and Reading Inverted Index using Lucene. (JAVA)
— Stop word removal, punctuation removal (PERL)
— Feature Extraction. (JAVA)
— Dictionary Generation using Index. (JAVA)
— Web search and query processing. (PERL)
— Web page generation. (PERL)
— Interface (JSP)

The implementation is extremely modular and can deal with large datasets. It is running on a Celeron machine and hence is not very fast.
Fig. 3. Screenshot for query “eccv 2006”
Fig. 4. Screenshot for query “sigir”