A Vector Space Engine for Web Services [1]

Adarsh Mohata, Ajith P S, Ashish Kedia

September 22, 2014

Outline

1. Vision
2. Abstract
3. Vector Space Model
4. Implementation
5. Progess
6. References

Vision

- Finding already existing web services on the internet easily
- Getting as much semantic information as possible from the files used to describe the services

How to achieve the goal ??

- Mining of Meta (WSDL) Files for web services
- Indexing as many WSDL files as possible
- Extracting keywords from the files indexed so that the method of vector space model can be used to represent web services
- Implementing an algorithm which allows us to join detached web service repositories to a single one
- Execute queries in the resultant vector space

The Term Space

- Create a vector space of n - dimensions where each dimension is a keyword extracted from the meta-files indexed
- Dimension grows when a previously undiscovered term is encountered
- Each document is represented by a vector :

$$d = (d_1, d_2, d_3....d_n) \tag{1}$$

where d_i is a real number indicating the degree of importance of term t_i in describing the document d

Weighting of Terms

Instead of binary weightage the authors of the paper have used the inverse document frequency of a document to assign weightage to occurence of each term in a given document

$$idf_k = ld\left(\frac{N}{n_k} + 1\right) \tag{2}$$

$$w_{ik} = tf_{ik} * ld\left(\frac{N}{n_k} + 1\right) \tag{3}$$

where, T_k = term k in Document D_i,
tf_{ik} = frequency of tem T_k in Document D_i,
idf_k = inverse document frequency of term T_k,
N = total number of documents,
n_k = the number of documents that contain T_k

Dynamic Modeling

- In this system the values of each vector reflect the overall number of documents and the particular weights to each term.
- Only possible in a centralized model where the values for N and N_k are already known.
- When two vector spaces are to be combined this knowledge is not available prior
- All necessary data has to be stored individually to enable weighting at runtime
- Additional overhead in query processing but adding new documents is extremely fast

Adding New Document

When a new document is added to the index:

1. The raw term frequencies are calculated for the documents. The vector space is expanded if new terms are found
2. The raw term frequencies are stored for each term that occurs in the new document
3. Values of N and N_k are updated

Data Structure proposed for implementation = Hash Table

Document Rating Algorithm

- Once the term weights for a document or a query has been assigned the similarity to other documents within the same term space can be rated and the method proposed is Cosine Coefficient
- It takes two vectors p and q and generates the cosine of the angle between them
- Documents with no common terms have cosine of 0 and those documents which are very similar have a cosine value near to 1

$$cos\left(p, q\right) = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2 \sum_{i=1}^{n} q_i^2}} \tag{4}$$

where p_i and q_i are dimensions of 2 given vectors

Implementation

The implementation can be divided into the following sections :

1. Indexing of WSDL files
2. Keyword extraction
3. Query processor and joiner
4. Basic User-Friendly Frontend

Challenges faced

We faced the following challenges during the immplementation of the concept :

1. UDDI repositories are obsolete. Major UDDI registry vendors like Microsoft, IBM, etc. are no longer providing this service
2. Multiple langages in repositories that we gathered
3. Detecting Bad Meta-Files (Corrupt or unavailable)
4. Most of the servers do not allow cross-domain scripting which is essential for automating the process of collecting web service meta-files
5. Working with limited resources like slow WiFi in Hostel

Our Work till 21st September, 2014

1. We have been able to find few public repositories but they contain only a few hundred meta-files
2. We have written customised scripts for gathering the destination and meta-files of all those web services
3. We have also written scripts to parse the various URLs related to meta-files
4. We have made a basic frontend for the engine in the form of a web service although currently it offers only a simple unit conversion as the service

Tools and Packages Used

1. JavaScript - To parse all links
2. Python - To automate downloading of parsed links (Meta-Files) using multiple threads
3. Shell Scripts - Useful tools like sed, grep, tr, etc to parse the documents
4. Web Browser's Developer Tool - For effective debugging
5. LaTeX

What we have Learned ??

- Python, JavaScript, Mechanize, Multi-Threading
- Using tools like Git and Latex
- Documentation is important
- How to conduct research, read and understand paper
- Team Collaboration

References I

[1] C. Platzer and S. Dustdar, "A vector space search engine for web services," *Third European Conference on Web Services*, 2005.