

WikiDiscovery

Sowmya Hariharan and Avinash Kautham Subramaniam Ravi

Department of Computer and Information Science & Engineering

INTRODUCTION

Wikipedia is one of the largest online repositories used by people all around the world. Generally, used to get abundance of information it is not always that a user going to Wikipedia is satisfied with one page, information is vital and generally going through a few pages gives a more clear idea of topic currently searched. A primary challenge to this is that, user has to manually go through the page and related articles to find out relevant information.

Here we propose WikiDiscovery – which is a topic discovery mechanism over the Wikipedia pages. Given two topics, second topic is discovered from the first with the help of tag propagation algorithm, the various related articles which have similar tags are returned as suggested reading list. So, the user can go through the related articles and gather the needed information.

OBJECTIVES

WikiDiscovery aims to automate entire search process, so it makes browsing through the topics less cumbersome. It is a filtered search criteria.

This mechanism allows a natural transition between the two topics, based on tags therefore giving the user a better understanding to the topics.

With this, a comparative study between the two topics can be done, which can be used to get more information, which can also be contributed again to the page.

Populating a particular page with related tags, could be re-used in the future for other applications as well. More filtering methods can be employed as well.

ALGORITHM

The various topics of wikipedia pages/articles are extracted with a python script, which is stored in JSON format. Annotated-Wiki Extractor is then used to extract actual url and annotations inside the url which are related tags(initially), output is then stored in a separate file. The next step is, Wiki-HTTP-Pruning where the topic is extracted from the URL and stored along with tags. These are to be stored in the form of a graph database, where there exists two kinds of nodes - "TOPIC" and "TAG".

WHY Graph Database ?

If, topic and associated tags are stored in a single node then finding similar nodes will be extremely difficult task, you will have to keep track of list of tags which have been encountered so far and also maintain node associativity. There will be repetition of tags as well. You would need to traverse the entire graph as well to find the needed information. In this case, by maintaining TOPIC and TAG nodes, it limits number of hops to only 2, so querying becomes extremely fast. There is no repetition of TAG nodes, if two topics have same tags, those two TOPIC nodes will be linked to that particular TAG node.

After initial graph database is constructed, tags have to be propagated. So, we manually assigned tags to certain number of nodes and called tag propagation algorithm based on Affinity Propagation. As the tags are propagated, for a new TAG assigned to a TOPIC node, graph has to be updated, there should be a link from the Topic node to TAG node. The next step is to query the nodes. Given two topics, locate first TOPIC node, check the other topic node whichever have common tags, they need to be returned. At the end, list of topic nodes will be presented as a list of URLs to the user.

RESULTS

The input is in the form of topics to be searched, output is a list of URLs that is presented to the user. So, to verify the output, we manually have to click the various links and check. The metrics which we have chosen for performance is Precision and Recall, which are generally used for pattern recognition and information theory.

Precision is the probability that a retrieved document is relevant. Recall is the probability that a relevant document is retrieved in a search.

No of queries	Precision	Recall
5	70%	65%
10	72%	68%

FUTURE WORK

There seems to quite a few areas which can be worked here. So, the first is using user feedback. So have a rating system, which provides user feedback. So as more users give feedback weights can be adjusted and propagated tags can be validated.

Caching locally the retrieved information, usually the same data gets searched again, so if we maintain a local cache, it would avoid same searches again, check local cache first before running the algorithm. This will save time and also avoid one user to increase confidence or rating un-necessarily.

REFERENCES

- [1] Automatic Image Tagging through Information Propagation in a Query Log Based Graph Structure, Teresa Bracamonte and Barbara Poblete http://jcc2011.atalca.cl/actas/ET/et2011_submission_16.pdf
- [2] Introduction to Bayesian Learning, <http://www.dcc.fc.up.pt/~ines/aulas/0809/MIM/aulas/bayes08.pdf>
- [3] Dbpedia – Wikipedia dataset http://downloads.dbpedia.org/preview.php?file=3.8_sl_en_sl_labels_en.nq.bz2
- [4] Neo4j Reference Manual <http://docs.neo4j.org/chunked/1.8.2/reference-documentation.html>
- [5] Precision and Recall http://en.wikipedia.org/wiki/Precision_and_recall

ACKNOWLEDGEMENTS

We would like to thank the following people –
Daisy Zhe Wang, Assistant Professor, Computer and Information Science & Engineering, Course Instructor – CIS6930 Large Scale Adv Data Analysis

Abhiram Jagarlapudi, Grader, CIS 6930 Large Scale Adv Data Analysis

Vikrant Vicky Sagar, Prachi Maheshwari, Kumar Shashank – Collaborators for the Graph database.

Algorithm used to discover topics:

