Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

Free and Open Source search engine

1School of Computer, Wuhan University, Wuhan 430072, China
2Department of computer science, Huazhong Normal University, Wuhan 430079, spain
(2013)

Abstract

Nutch is an effort to build a Free and Open Source search engine. It uses Lucene for the search and index component. The fetcher (robot) has been written from scratch solely for this project.

Nutch has a highly modular architecture allowing developers to create plug-ins for activities such as media-type parsing, data retrieval, querying and clustering.

Doug Cutting is the lead developer of Nutch.

1.2 What is Lucene?

Lucene is a Free and Open Source search and index API released by the Apache Software Foundation. It is written in Java and is released under the Apache Software License.

Lucene is just the core of a search engine. As such, it does not include things like a web spider or parsers for different document formats. Instead these things need to be added by a developer who uses Lucene.

Lucene does not care about the source of the data, its format, or even its language, as long as you can convert it to text. This means you can use Lucene to index and search data stored in files: web pages on remote web servers, documents stored in local file systems, simple text files, Microsoft Word documents, HTML or PDF files, or any other format from which you can extract textual information.

Lucene has been ported or is in the process of being ported to various programming languages other than Java:

1.3 What License?

Both Nutch and Lucene are Apache projects and carry the Apache license

2 The Design of Nutch

The Nutch search engine consists, very roughly, of three components:

1. The Crawler, which discovers and retrieves web pages
2. The ‘WebDB’, a custom database that stores known URLs and fetched page contents
3. The ‘Indexer’, which dissects pages and builds keyword-based indexes from them

 After the initial creation of an Index, it is usual to perform periodic updates of the index, in order to keep it up-to-date. We will look into the details of index maintenance in the parts following this.

2.2 The Nutch Web Application

Apart from the above three components, it has a Search Web Application. This application is a JSP application that can be configured and deployed in a servlet container.

2.3 The Nutch API

All components listed above use the nutch API. The users can utilize the API via two approaches, which depends on the task at hand.

1. Through the nutch Shell Script for administrative tasks, such as creating and maintaining indexes
2. Through the Search Web Application, in order to perform a search using keywords

The *sequence diagram* below shows how each of these components interact in implementing a Nutch based search application.

3 Implementing a Nutch Search

Implementing our own version of Nutch is fairly easy, provided that you;

1. have a basic understanding of how a web search engine works and
2. are comfortable working in a command line and finally
3. have a fair knowledge of Java and Servlet containers

If you said ‘yes’ to all three questions above, you have a very high probability of having your Nutch implementation up and running by the end of the steps which follows.

3.1 Before We Begin

3.1.1 Download Nutch

Go to http://www.apache.org/dyn/closer.cgi/lucene/nutch/ and select a mirror to download Nutch. The version described in this document is version 0.7. After downloading the archive, extract it to your disk.

NOTE: This document assumes that the archive was extracted to /home/tyrell/nutch-0.7 change this path to reflect your location.

3.1.2 Download and Install a Servlet Container

Apache Tomcat is a popular Open Source servlet container. We will use this to deploy the Nutch search web application in this document. The version referred to in this document is version 5.5. You can download Tomcat from http://tomcat.apache.org/download-55.cgi

3.1.3 To Cygwin or not to Cygwin

To run the Nutch shell scripts and create the indexes, we require a UNIX like environment. If you do not have access to a UNIX environment, you can use Cygwin as an alternative. More details and download information for Cygwin can be found at http://www.cygwin.com/

3.2 Creating the Index

In order for the nutch web application to function, it will require at least one search index. A search index in nutch is represented in the file system as a directory. However, it is much more than that and is similar in functionality to a database. The nutch API will interact with this index making the internal mechanisms transparent to both developers and end-users.

The steps to create and integrate a new index are as follows;

NOTE: The steps below are assumed to be carried out from inside the /home/tyrell/nutch-0.7 directory created when extracting the archive. Change the path according to your local instance.

3.2.1 Create a directory of root urls.

The nutch 'crawl' command expects to be given a directory containing files that list all the root level urls to be crawled. So create a 'urls' directory in the nutch directory. Then, to crawl the http://www.virtusa.com site from scratch, you might start with a file named 'virtusa' in the 'urls' directory, and in the file, add just the URL for the Virtusa home page, http://www.virtusa.com. All other pages should be reachable by links from this page.

The 'depth' option to the crawl command will limit how far the crawl goes. Also, the conf/crawl-urlfilter.txt file, described next, will limit what sites to crawl to.

3.2.2 Edit the file conf/crawl-urlfilter.txt

If you are using TRUNK then there is no file called conf/crawl-urlfilter.txt but conf/crawl-urlfilter.txt.template. Just do

```
```

If you already have this file then replace the existing domain name with the name of the domain you wish to crawl. For example, if you wished to limit the crawl to the virtusa.com domain, the line should read:

```
•
```

This will include any url in the domain virtusa.com in the crawl.

3.2.3 Running a Crawl

Once things are configured, running the crawl is done by using the crawl command.

Its options include:

```
•   * -dir dir names the directory to put the crawl in.
•   * -depth depth indicates the link depth from the root page that should
    be crawled.
•   * -delay delay determines the number of seconds between accesses to
    each host.
•   * -threads threads determines the number of threads that will fetch in
    parallel.
```

For example, a typical command might be:

```
•   bin/nutch crawl urls -dir crawl.virtusa -depth 10
```

3.2.4 Output of the crawl

Assuming that the above command is executed with the given parameters, the result will be as follows;

- A new directory will be created named ‘crawl.virtusa’ in the working directory (according to this guide, /home/tyrell/nutch-0.7/crawl.virtusa)
- This new directory will contain the search index for the URLs given in the flat file named ‘urls’ (created in the working directory according to this example)
- The ‘depth’ of the search index will be 10

3.2.5 Errors and Failures

- JAVA_HOME environment variable not set
  - Set the variable to point to your JDK installation
- Missing ‘urls’ flat file
  - Create the file and give the correct path to it in the crawl command
- The indexing domain is not properly defined in the ‘crawl-urlfilter.txt’, as described in the guide

3.3 Configuring the Nutch Web Application

The search web application is included in your downloaded Nutch archive. In order for the nutch search web application to function properly, it needs to know where to find the indexes. We need to map our indexes by editing the ‘nutch-site.xml’ file.

The steps to follow would be;

1. Deploy the Nutch web application as the ROOT context
   - It is not clear why the developers designed the application to run in the root context. However it is possible to modify the application to enable it to be deployed normally.
2. In the web application deployment directory, open the '\WEB-INF\classes\nutch-site.xml' file in a text editor.
3. Change the values of the tags as follows and save the changes.
4. Re-start Tomcat

3.4 Running a Test Search

Now that we have created the indexes and configured the Nutch web application, the only thing left is to give it a test run.

Open a browser and type your Tomcat URL (ex: http://localhost:8080). The following page will greet you if the web application is configured properly.

Now type a keyword (ex: virtusa) and click search. If the implementation works as expected, the following results page will be displayed.

3.5 Maintaining Our Index

Now that all is working, we need to think the long term maintenance of the Index. This is a required activity because the web gets updated frequently. New content will appear on sites while existing content might get modified or deleted altogether.

Nutch provides the administrator with a set of commands to update a given index, however performing them manually will not only be tiresome but also unproductive. Since this task need to be carried out periodically it should ideally be scheduled.

3.5.1 Creating a Maintenance Shell Script

Create a new shell script with the commands given in fig 3.4. The script contains three command segments.

1. Commands, which set the environment (ex:JAVA_HOME).
2. Commands to do the index updating.
3. Commands to optimize the updated index.

 NOTE: Index optimization is necessary to prevent the index from becoming too large, which will eventually result in a ‘too many open files’ exception in Lucene.

3.5.2 Scheduling Index Updates

The above shell script can be scheduled to be run periodically using a ‘cron’ job.

REFERENCES

[1] J. Campbell, "Speaker recognition: a tutorial," *Proc. IEEE*, vol. 85, pp. 1437–1462, Sept. 1997.

[2] D. A. Reynolds, T. Quatieri, and R. Dunn, "Speaker verification using adapted Gaussian mixture models," *Digital Signal Processing*, vol. 10, no. 1–3, pp. 19–41, 2000.

[3] D. A. Reynolds, "Comparison of background normalization methods for text-independent speaker verification," in *Proc. Eurospeech*, 1997.

[4] D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Trans. Speech Audio Processing*, vol. 3, no. 1, pp. 72–83, 1995.

[5] J. L. Gauvain and C.-H. Lee, "Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains," *IEEE Trans. Speech Audio Processing*, vol. 2, pp. 291–298, Apr. 1994.

[6] E. Bocchieri, "Vector quantization for the efficient computation of continuous density likelihoods," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1993, pp. 692–695.

[7] K. M. Knill, M. J. F. Gales, and S. J. Young, "Use of Gaussian selection in large vocabulary continuous speech recognition using HMMs," in *Proc. Int. Conf. Spoken Language Processing*, 1996.

[8] D. B. Paul, "An investigation of Gaussian shortlists," in *Proc. Automatic Speech Recognition and Understanding Workshop*, 1999.

[9] T. Watanabe, K. Shinoda, K. Takagi, and K.-I. Iso, "High speed speech recognition using tree-structured probability density function," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1995.

[10] J. Simonin, L. Delphin-Poulat, and G. Damnati, "Gaussian density tree structure in a multi-Gaussian HMM-based speech recognition system,"

[11] T. J. Hanzen and A. K. Halberstadt, "Using aggregation to improve the performance of mixture Gaussian acoustic models," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1998.

[12] M. Padmanabhan, L. R. ahl, and D. Nahamoo, "Partitioning the feature space of a classifier with linear hyperplanes," *IEEE Trans. Speech Audio Processing*, vol. 7, no. 3, pp. 282–288, 1999.
in *Proc. Int. Conf. Spoken Language Processing*, 1998.

[13] R. Auckenthaler and J. Mason, "Gaussian selection applied to text-independent speaker verification," in *Proc. A Speaker Odyssey—Speaker Recognition Workshop*, 2001.

[14] J. McLaughlin, D. Reynolds, and T. Gleason, "A study of computation speed-ups of the GMM-UBM speaker recognition system," in *Proc. Eurospeech*, 1999.

[15] S. van Vuuren and H. Hermansky, "On the importance of components of the modulation spectrum of speaker verification," in *Proc. Int. Conf. Spoken Language Processing*, 1998.

[16] B. L. Pellom and J. H. L. Hansen, "An efficient scoring algorithm for Gaussian mixture model based speaker identification," *IEEE Signal Processing Lett.*, vol. 5, no. 11, pp. 281–284, 1998.

[17] J. Oglesby and J. S. Mason, "Optimization of neural models for speaker identification," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 1990, pp. 261–264.

[18] Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, "Global optimization of a neural network—hidden Markov model hybrid," *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp. 252–259, 1992.

[19] H. Bourlard and C. J. Wellekins, "Links between Markov models and multilayer perceptrons," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, pp. 1167–1178, Dec. 1990.

[20] J. Navrátil, U. V. Chaudhari, and G. N. Ramaswamy, "Speaker verification using target and background dependent linear transforms and multi-system fusion," in *Proc. Eurospeech*, 2001.

[21] L. P. Heck, Y. Konig, M. K. Sonmez, and M. Weintraub, "Robustness to telephone handset distortion in speaker recognition by discriminative feature design," *Speech Commun.*, vol. 31, pp. 181–192, 2000.

[22] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. R. Statist. Soc.*, vol. 39, pp. 1–38, 1977.

[23] K. Shinoda and C. H. Lee, "A structural Bayes approach to speaker adaptation," *IEEE Trans. Speech Audio Processing*, vol. 9, no. 3, pp. 276–287, 2001.

[24] K. Fukunaga, *Introduction to Statistical Pattern Recognition.* New York: Academic, 1990.

[25] J. C. Junqua, *Robust Speech Recogntion in Embedded Systems and PC*

[26] U. V. Chaudhari, J. Navrátil, S. H. Maes, and R. A. Gopinath, "Transformation enhanced multi-grained modeling for text-independent speaker recognition," in *Proc. Int. Conf. Spoken Language Processing*, 2000.

[27] Q. Lin, E.-E. Jan, C. W. Che, D.-S. Yuk, and J. Flanagan, "Selective use of the speech spectrum and a VQGMM method for speaker identification," in *Proc. Int. Conf. Spoken Language Processing*, 1996.

[28] S. Raudys, *Statistical and Neural Classifiers: An Integrated Approach to Design.* New York: Springer, 2001.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, pp. 318–364.

[30] [Online] Available: http://www.nist.gov/speech/tests/spk/index.htm.

[31] J. Pelecanos and S. Sridharan, "Feature warping for robust speaker verification," in *Proc. A Speaker Odyssey—Speaker Recognition Workshop*, 2001.

[32] B. Xiang, U. V. Chaudhari, J. Navrátil, N. Ramaswamy, and R. A. Gopinath, "Short-time Gaussianization for robust speaker verification," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, 2002.

[33] G. R. Doddington, M. A. Przybocki, A. F. Martin, and D. A. Reynolds, "The NIST speaker recognition evaluation—overview, methodology, systems, results, perspective," *Speech Communication*, vol. 31, pp. 225–254, 2000.

Bing Xiang (M'03) was born in 1973 in China. He received the B.S. degree in radio and electronics and M.E. degree in signal and information processing from Peking University in 1995 and 1998, respectively. In January, 2003, he received the Ph.D. degree in electrical engineering from Cornell University, Ithaca, NY.

From 1995 to 1998, he worked on speaker recognition and auditory modeling in National Laboratory on Machine Perception, Peking University. Then he entered Cornell University and worked on speaker recognition and speech recognition in DISCOVER Lab as a Research Assistant. He also worked in the Human Language Technology Department of IBM Thomas J. Watson Research Center as a summer intern in both 2000 and 2001. He was a selected remote member of the SuperSID Group in the 2002 Johns Hopkins CLSP summer workshop in which he worked on speaker verification with high-lelvel information. In January, 2003, he joined the Speech and Language Processing Department of BBN Technologies where he is presently a Senior Staff Consultant-Technology. His research interests include large vocabulary speech recognition, speaker recognition, speech synthesis, keyword spotting, neural networks and statistical pattern recognition.

Toby Berger (S'60–M'66–SM'74–F'78) was born in New York, NY, on September 4, 1940. He received the B.E. degree in electrical engineering from Yale University, New Haven, CT in 1962, and the M.S. and Ph.D. degrees in applied mathematics from Harvard University, Cambridge, MA in 1964 and 1966, respectively.

From 1962 to 1968 he was a Senior Scientist at Raytheon Company, Wayland, MA, specializing in communication theory, information theory, and coherent signal processing. In 1968 he joined the faculty of Cornell University, Ithaca, NY where he is presently the Irwin and Joan Jacobs Professor of Engineering. His research interests include information theory, random fields, communication networks, wireless communications, video compression, voice and signature compression and verification, neuroinformation theory, quantum information theory, and coherent signal processing. He is the author/co-author of Rate Distortion Theory: A Mathematical Basis for Data Compression, Digital Compression for Multimedia: Principles and Standards, and Information Measures for Discrete Random Fields.

Dr. Berger has served as editor-in-chief of the IEEE TRANSACTIONS ON INFORMATION THEORY and as president of the IEEE Information Theory