Pattern extraction from the world wide web

Praveena Mettu

University of Nevada, Las Vegas

Follow this and additional works at: http://digitalscholarship.unlv.edu/thesesdissertations

Part of the Computer Sciences Commons

Repository Citation
Mettu, Praveena, "Pattern extraction from the world wide web" (2010). UNLV Theses, Dissertations, Professional Papers, and Capstones. 741.
http://digitalscholarship.unlv.edu/thesesdissertations/741

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
PATTERN EXTRACTION FROM THE WORLD WIDE WEB

by

Praveena Mettu

Bachelor of Technology, Computer Science and Engineering
Jawaharlal Nehru Technological University, India
May 2008

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2010
THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Praveena Mettu

titled

Pattern Extraction from the World Wide Web

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
School of Computer Science

Kazem Taghva, Committee Chair
Ajoy K. Datta, Committee Member
Laxmi P. Gewali, Committee Member
Muthukumar Venkatesan, Graduate Faculty Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies and Dean of the Graduate College

December 2010
ABSTRACT

Pattern Extraction from the World Wide Web

by

Praveena Mettu

Dr. Kazem Taghva, Examination Committee Chair
Professor, Department of Computer Science
University of Nevada, Las Vegas

The World Wide Web is a source of huge amount of unlabeled information spread across different sources in varied formats. This presents us with both opportunities and challenges in leveraging such large amount of unstructured data to build knowledge bases and to extract relevant information.

As part of this thesis, a semi-supervised logistic regression model called “Dual Iterative Pattern Relation Extraction” proposed by Sergey Brin is selected for further investigation. DIPRE presents a technique which exploits the duality between sets of patterns and relations to grow the target relation starting from a small sample.

This project built in JAVA using "Google AJAX Search API" includes designing, implementing and testing DIPRE approach in extracting various relationships from the web.

Keywords: Pattern Extraction, Machine Learning, DIPRE
TABLE OF CONTENTS

ABSTRACT ...................................................................................................................... iii

LIST OF FIGURES ........................................................................................................ v

ACKNOWLEDGEMENTS ............................................................................................... vi

CHAPTER 1  INTRODUCTION ....................................................................................... 1
  1.1 Aims and Objectives ............................................................................................. 2
  1.2 Thesis Organization ............................................................................................ 3

CHAPTER 2  LITERATURE OVERVIEW ...................................................................... 4
  2.1 Information Extraction Systems .......................................................................... 5
  2.2 DIPRE Algorithm .................................................................................................. 9

CHAPTER 3  IMPLEMENTATION .................................................................................. 12
  3.1 Choosing the search engine ................................................................................ 12
  3.2 Choosing the programming Language ................................................................... 13
  3.3 Overview of Java Program .................................................................................. 14
    3.3.1 User Interface ............................................................................................... 14
    3.3.2 Find Occurances ........................................................................................ 15
    3.3.3 Get the next set of tuples .......................................................................... 18
    3.3.4 Output ......................................................................................................... 18

CHAPTER 4  RESULTS AND EVALUATION .................................................................. 25

CHAPTER 5  CONCLUSION AND FUTURE WORK ...................................................... 34
  5.1 Conclusion ........................................................................................................... 34
  5.2 Future Possibilities ............................................................................................. 35

BIBLIOGRAPHY .......................................................................................................... 38

VITA .............................................................................................................................. 40
# LIST OF Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Pattern Learner</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2</td>
<td>DIPRE Algorithm</td>
<td>10</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Input for parameter1</td>
<td>14</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Input for parameter2</td>
<td>14</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Input for SearchSite</td>
<td>15</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Json operations to fetch results from internet</td>
<td>16</td>
</tr>
<tr>
<td>Figure 7.1</td>
<td>Result Set1</td>
<td>19</td>
</tr>
<tr>
<td>Figure 7.2</td>
<td>Result Set2</td>
<td>20</td>
</tr>
<tr>
<td>Figure 7.3</td>
<td>Result Set3</td>
<td>21</td>
</tr>
<tr>
<td>Figure 7.4</td>
<td>Result Set4</td>
<td>22</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Google Query Restriction</td>
<td>36</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I would like to take this opportunity to thank Dr. Kazem Thagva, my research advisor for his support and guidance. Throughout the project, he provided with a lot of ideas and sound advice.

I am also grateful to all the faculty, librarians and secretaries at University of Nevada, Las Vegas.

It is a pleasure to offer my regards to my friends and students at the University for a stimulating and fun-filled environment to learn and grow. Thanks a ton for the camaraderie and entertainment.

Lastly and most importantly, I would like to thank my family for all the love and sacrifices they made for my well-being. To them, I dedicate this thesis.
CHAPTER 1

INTRODUCTION

The World Wide Web, with its masses of unstructured and inconsistently coordinated information is easier to construe by humans than by machines.

Imagine what could be done if we can train systems to analyze text and mine the enormous amount of data created by people on the Internet. It would become the source of unprecedented amount of information easily interpreted by machines, thus maximizing both people and computer resources. It would include the largest and most diverse databases of people, products and academic work constantly growing with thousands of new web pages appearing everyday in the World Wide Web [1].

Information Extraction (IE) can be dated back to the early days of Natural Language Processing (NLP) in the 70’s. IE is a technique to retrieve focused target relations from unstructured machine-readable documents by means of NLP [2]. Most information extraction systems are based on rule-based methods that employ some form of machine learning. They can generate rules based on labeled-data which unfortunately may require labor-intensive hand-coding of search patterns. Data may also need to be massaged specific to the desired target relation which we want to extract. One approach to automating
the information extraction process is a class of techniques called semantic bootstrapping. In general, these techniques use some form of machine learning to recognize semantic patterns in large amounts of unlabelled data. The recognized patterns are then applied iteratively to extract more semantics. Effectively, semantic bootstrapping applies semantic labels to the data based on what it already knows and then labels even more data based on the new labels it applied [3].

1.1 Aims and Objectives

This thesis focuses on extracting entities from World Wide Web which have a relationship between them that is similar to the one existing between the two entities provided. There have been a number of algorithms proposed in recent years aimed at extracting relationships between entities. In this thesis, by implementing one of those algorithms, we intend to evaluate its validity and proficiency.

In his experiment, Sergey Brin focused on extracting (author, title) pairs from web pages. Given the nature of the data, there must have been heavy use of HTML formatting tags to automatically generate patterns consisting of authors and titles from various book-related sites, often in tables or lists. However, in our experiment we want to enlarge the data-set on which DIPRE can be executed and evaluate its effectiveness on a more free-form natural language text in internet [3].
1.2 Thesis Organization

Chapter 2 covers the Literature Review, which discusses a general approach to various research topics in information extraction. This chapter also covers some techniques for extracting factual information from the web and their applications that relate to this thesis.

Chapter 3 covers the approach implemented in this thesis (Java program, Google search API). It analyses the algorithm and design decisions made for the project. Explanation of various parts of the algorithm and how they function together is provided. Also, the technical challenges faced during project implementation and how they were overcome are mentioned.

Chapter 4 lists the results from the execution of the java program built based on DIPRE algorithm when provided with a seed pair. It also discusses the effectiveness and limitations of the algorithm.

Finally, Chapter 5 concludes the entire thesis and suggests how this utility program can be improvised for future work.
CHAPTER 2
LITERATURE OVERVIEW

This chapter presents relevant background knowledge that is closely related to this thesis. This will help readers to easily understand the analysis.

Machine Learning allows computer programs to automatically improve by interpreting the obtained results. This concept can be applied in Natural Language Processing by either annotating examples of mapping we are interested in or by applying a machinery to learn from examples. Since Information Extraction systems have to process large bodies of information, machine learning is an integral part of such algorithms.

Reducing the human workload in developing extraction rules is a huge challenge in Information Extraction. Depending on the level of human involvement most adaptive information extraction systems use different machine learning algorithms. However, many IE systems use very complex pre-defined rules rather than using any machine learning technique at all to accomplish the extraction task [4]. In the first section of this chapter, each type of IE system is reviewed and their advantages and disadvantages are discussed briefly. In the second section, we discuss Sergey Brin’s DIPRE algorithm.
2.1 Information Extraction Systems

Information Extraction Systems usually rely on a set of Seed-Patterns that they use to compare and retrieve relevant information from plain text. The new-patterns retrieved can be used as both extractors and discriminators.

![Pattern Learner Diagram](image)

**Figure 1: Pattern Learner [5]**

Extractors can be further used as seed-patterns to extract new patterns. Not every extracted pattern may be relevant for our study. By adding those to the Discriminators list (negative list), we improve the overall accuracy of the algorithm. These discriminators prevent the algorithm from issues of semantic drift, where erroneous new patterns cause the system to run off track.

Many different IE approaches have been developed in the recent past for various IE tasks depending on the goal of study and the nature of
information source. However, no matter which data sources are utilized in relation extraction, all Information Extraction Systems have to meet three requirements:

1) A source which comprises of entities of our interest hidden in plain text. A group of such databanks will serve as data sources from which semantic relations will be extracted

2) A semantic or linguistic source in which the context for relations between entities is provided

3) Algorithms for automatic execution of processing operations.

How well a relation extractor performs is determined mainly by the contextual information sources and algorithms [6]. A heavy rule-based extractor with a good context source need not always contain “pattern learning algorithms”. However, this involves humungous human effort. Pattern Learning algorithms generate their rules from examples thus reducing human effort. The early IE systems were primarily rule-based. Skillful engineers embed the extraction rules in these systems using their knowledge in the domain. Unfortunately, natural languages are extremely flexible and it is impossible to hand code a set of rules to extract accurate information from a variety of web pages with varying structures. Hence, more intelligent and scalable approaches were sought after.

Machine Learning techniques can help train IE systems to discover new extraction rules automatically with experience. This is much faster
than the rule-based approach and has been used more often in recent IE system developments. Depending upon the amount of human intervention required, Machine Learning (ML) approaches can be roughly divided into 3 categories [4].

Supervised IE systems: These methods use a set of manually labeled data to construct extraction rules. This approach is relatively more automated compared to a rule-based approach, however needs human supervision. AutoSlog is a relationship extraction approach based on this method [7]. It needs texts and tagged noun phrases to generate patterns. For example, let us consider this sentence:

“A pedestrian was attacked by a dog”

Manually, “pedestrian” is tagged as a victim and “dog” as the perpetrator of a murder. AutoSlog will propose “<X> was attacked” as a pattern where <X> is a victim. It can use this pattern to find more victims. It will also propose “was attacked by <Y>” as a pattern where <Y> is the perpetrator. It can find more perpetrators by matching new sentences with learned pattern. The laborious job of manually annotating training documents makes this approach unfavorable in many practical scenarios.

Non-Supervised IE systems: These methods use highly automated algorithms hence; do not require any training documents or any human supervision. However, as a drawback they tend to suffer from inaccuracies in results. AutoFeed [8], Yangarber [9] and KnowItAll [10] are
examples of this method. Although these methods are still “young”, they suggest futuristic possibilities of building automated information extraction systems that can be used in many practical applications like search engines, knowledge-based AI systems, etc [10].

Semi-Supervised IE systems: These methods are somewhere in-between supervised and unsupervised IE systems. The advantage of a semi-supervised method over supervised method is that, given a small set of annotated examples, these methods can generate their own training data through “active learning”. These are also the most practical methods used in IE systems. In the process of active learning, a small amount of labeled training set is fed to the algorithm. It applies these rules to classify unlabeled data. The most trusted newly classified sets along with their labels are added to the training set. The algorithm re-learns from this new training set. These rules are again applied to classify unlabeled data. This procedure is applied repeatedly until unlabeled data set is exhausted or the trust-worthiness of the results drops below a threshold.

DIPRE (Dual Iterative Pattern Relation Extraction) [1] and Snowball [11] are two examples of Semi-Supervised IE systems which are used in extracting patterns from unstructured text. Traditional IE systems tend to extract all semantic information from each document. However, DIPRE and Snowball help create knowledge base from the web by reading only relevant content of the documents. They can extract pairs of related
entities, like authors and their books, from the web using a small group of seed pairs. They are especially effective in extracting commonly occurring relationships like (authors, books), (acronyms, meanings), etc.

The relation extraction approach described in DIPRE [1] is the core algorithm involved in a lot of other semi-supervised algorithms as well. Hence, it is of utmost importance for any IE system architect to understand this algorithm. In the next section, we shall discuss DIPRE [1] in detail.

2.2 DIPRE Algorithm

DIPRE algorithm by Brin [1] is one of the earliest examples of discovering binary relationships using bootstrap learning approach. This is a technique where the duality between sets of relations is exploited to grow the target relations starting from a small sample.

Brin demonstrated this principle by finding authors and titles in the WWW. Let’s say we look for “Mario Puzo” and “The Godfather” in the internet. There could be a few repeated patterns when you observe the citations of this book, for example “The Godfather was written by Mario Puzo”. By looking for these “repeated patterns” in the internet, there is a good chance of uncovering other authors and titles. For example, the pattern/relation “was written by” could lead to various citations like “Catch-22 was written by Joseph Heller”, “Gone with the wind was written by Margaret Mitchell”, etc. These new authors and titles can again be used to find new patterns.
Steps in DIPRE:

1. Start with a small set of trusted pairs (authors, books)
2. Find occurrences of these pairs in the WWW or in a defined set of Websites/WebPages
3. Identify generalized patterns from the citations fetched in step 2.
4. Use these identified patterns to find more pairs in the web.
5. Repeat steps 2 to 4 until no new patterns can be learnt.
Pattern Generation (Step 3) is the most critical step in this algorithm. The accuracy of results and performance can get hugely impacted depending on how the implementation of this step is designed. Many complex variations of DIPRE algorithm have been proposed in the recent past; each with different ideas of pattern generation (Step 3). However, in this project we will focus on the well-known DIPRE algorithm originally suggested by Brin.
CHAPTER 3
IMPLEMENTATION

There were two major decisions that had to be made before beginning the development of the application.

1. Search Engine
2. Programming Language

3.1 Choosing the search engine

There are various APIs available to retrieve information from the internet. Famous ones are listed below:

Google AJAX Search API [12]: This is a JavaScript Library that returns JSON encoded results for easy processing. It has a limit of 64 results per query.

Bing API [13]: Also known as the Live Search API, this API was built by Microsoft on standards that are based on SOAP, XML and WSDL technologies. It provides an XML web service through a SOAP API. Historically, this API has been more tested using C# projects. Hence, is more suitable to the .NET platform.

Yahoo BOSS API [14]: BOSS (Build Your Own Search Service) is Yahoo!’s open search web services platform. For each query, this API can return up to 100 results in XML or JSON format. It also has the ability to restrict search in a pre-defined set of websites.
NewYork Times Article Search API [15]: This is another interesting API which searches NYTimes articles to retrieve headlines, abstracts, lead paragraphs and links to associated multimedia. This API also returns results in JSON format. Before using any of these APIs, we need to adhere to their Terms of Service. There is not much of a difference between these APIs. Almost all of them retrieve results in JSON format which is I believe is an easy-to-use form of representing data similar to XML. We chose Google AJAX Search API for this implementation simply because it is the current market leader and has been researched extensively.

3.2 Choosing the Programming Language

JSON (JavaScript Object Notation) [16] is a lightweight data-interchange format. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.

We considered coding the algorithm either in C#, java or Perl. Eventually, we chose java as it is comparatively much faster than the others.
3.3 Overview of Java Program

In this section, we explain the UI and various sections of the algorithm.

3.3.1 User Interface

As per DIPRE, we intend to take two parameters as input from user. In Brin’s example, the study was focused on Books & Authors. However, in our experiment, we intend to apply DIPRE algorithm on any related parameters.

After providing the parameters, user can also specify any site where the search has to be restricted to. By default, we intend to search in
www.wikipedia.org because that’s where we found most of our results. However, user is free to change it to any other specific site. If the SearchSite is not specified, the search will be similar to a standard “Google Search” where it looks for these two related parameters all over the internet.

![Figure 5: Input for SearchSite](image)

### 3.3.2 Find Occurrences

Using these details, we generate the string that needs to be passed to the Google AJAX Search API. The following is done to build the search string.

1. Replace all spaces with + in parameters.
2. If SearchSite is empty, replace it with www.google.com
3. `SearchString = "%22 " + parameter1 + "%22" + "+" + "%22 " + parameter2 + "%22" + " site:" + searchsite`

Then the URL is built using this searchstring which can be passed to the search API.
The API result set is in the form of a JSON object. Within this object, there are eight occurrences inside an array. The program loops on the array and each result string is subdivided into “prefix”, “middle term” and “suffix” sections by looking up parameter1 and parameter2 in the string.

```java
URLConnection connection = url.openConnection();
connection.setRequestProperty("Referer", HTTP_REFERER);

// Get the JSON response
String line;
StringBuilder builder = new StringBuilder();
BufferedReader reader = new BufferedReader(
    new InputStreamReader(connection.getInputStream()));
while ((line = reader.readLine()) != null) {
    builder.append(line);
}

String response = builder.toString();
JSONObject json = new JSONObject(response);
JSONArray jarray = json.getJSONObject("responseData")
    .getJSONArray("results");

if (count == 0) {
    System.out.println();
    System.out.println(" Results:");
    System.out.println();
}

for (int i = 0; i < jarray.length(); i++) {
    System.out.print((i+count+1) + ": ");
    JSONObject job = jarray.getJSONObject(i);
    /*
     System.out.println(job.getString("titleNoFormatting"));
     System.out.println(job.getString("url"));
     System.out.println(job.getString("content"));
    */
    substring1 = job.getString("content");
```

Figure 6: json operations to fetch results from internet
Before working with json objects, json library has to be downloaded from json.org and added to the library list of the project. The above code describes how the results are fetched from the search API. The connection is opened for the specific URL which was built using the parameters passed by user. The inputstream is read line by line and appended to StringBuilder. This is then converted into a string “response”. The data in “response” is put into a JSON object “json”. The array list of results in json object can be looped on. As per Google AJAX Search API documentation, Title, TitleNoFormat, Url and Content tags can be used to get specific information from each result in Json object. By fetching the content into a string, it can be subdivided into various sections by looking for the parameters in the string. Once these sub-strings are stored, we can get the next set of eight results from the search API by passing the following string.

"http://ajax.googleapis.com/ajax/services/search/web?start=8&rsz=large&v=1.0&q=" + query

Parameter “start” tells the API where to start returning the results from. For example, if “start” is set as 8, then results 9-16 are passed back to the program. Parameter result size “rsz” can be set as small or large. Small gives 4 results and large gives 8 results at a time. The query is repeated by passing 8, 16, 24, 32, 40, 48, 56 as “start” parameters. Eventually, we get up to 64 results.
3.3.3 Get the next set of tuples

Most of the times, we get bad results because all the results need not always be relevant to the experiment. Hence, of the 64 results only a few maybe relevant for our analysis. The valid results are stored. Parsing through the valid results, most repeated “middle term” is fetched. Also, the longest common substring is fetched from “prefix” and “suffix”.

We prepare a query string by using these substrings.

Search query = Longest_prefix + "*+" + most_repeated_middle_term + "*+" + Longest_suffix + " site:” + searchsite

By passing this query to the search API, we can again get up to 64 results of new tuples. However, very few of them will be valid tuples. The occurrences of these valid tuples can again be searched in the internet to get valid relations. Such iterations can continue until the search API stops returning valid results. There has to be a filtering algorithm to determine valid tuples. They can also be manually checked before using to get next set of relations. This is a common problem in a lot of semi-supervised learning algorithms.

3.3.4 Output

The results are displayed on the console using a simple “System.out.println” statement. Let’s say I search for “Lebron James” and “NBA” in “Wikipedia.org”. Following is the result retrieved.
Querying for : %22 lebron+james%22+%22 nba%22 site:wikipedia.org
Figure 7.1: Result Set1
Figure 7.2: Result Set 2
Figure 7.3: Result Set3
Repeated middle term: BECAME THE YOUNGEST
number of times repeated: 3
Longest common url:
Longest common prefix: ON DECEMBER 17 2007
Longest common suffix: PLAYER TO SCORE ON FEBRUARY 22 2008

LEBRON JAMES GRABBED HIS 2500TH REBOUND AS A CAVALIER

Finding other parameters related to term: ON DECEMBER 17 2007

+BECAME THE YOUNGEST+ PLAYER TO SCORE ON FEBRUARY 22 2008 LEBRON JAMES GRABBED HIS 2500TH REBOUND AS A CAVALIER


Results: NIL

Explanation of output:

1. LEBRON JAMES-A-NBA-B-2-C-


HTTPSPORTSYAHOOCOM-E-

NEWSSLUGAPLEBRONSBOOKAMPPROVAPAMPTYPE LGNS

LIVINGSTON BILL JULY 22 2009 QUOTNEW-F- BOOK TELLS OF A

The substrings are separated using identifiers –A-, -B-, -C-, -D-, -E- and -F-

Parameter 1: LEBRON JAMES

Parameter 2: NBA

Order of parameters in the result : 2 (1 implies parameter1 came ahead of parameter2 in the result and 2 implies the opposite)

Link to webpage where the content was found :

http://en.wikipedia.org/wiki/LeBron_James
Middle Term : YAHOO NEWS HTTPSPORTSYAHOOCOM
Prefix :
NEWSSLUGAPEBRONSBOKAMPPROVAPAMPTYPE GNS LIVINGSTON
BILL JULY 22 2009 QUOTNEW
Suffix : BOOK TELLS OF A

And the most common repeated term was “BECAME THE YOUNGEST” and obviously there were no results found for further iterations. This happens often with DIPRE because the principle used in determining the next tuples is not very robust.
CHAPTER 4

RESULTS AND EVALUATION

In his experiment, Brin used DIPRE [1] to build a database of {Authors, Titles}. He started off with an initial sample of 5 books. Eventually, after numerous iterations and a bit of manual intervention to remove bogus data retrieved, he claimed to have built a repository of around 15,000 unique books. He suggested that the same principles can be applied to other domains like movies, music, restaurants, etc. Brin also mentioned that a sophisticated version of DIPRE may also be able to extract people directories, product catalogs, and more.

We wanted to experimentally test and evaluate some of these claims about the abilities of Brin’s DIPRE algorithm. As part of the experiment, we developed our own implementation of DIPRE in a java program which uses Google AJAX Search API as the web crawling tool.

This program developed as part of this project can be used to get relations between any two parameters. However, in this experiment we decided to focus on music and build a repository of {singers, songs}; similar to Brin’s experiment which built a repository of {Authors, Titles}.

We started the experiment with only one set of related parameters.

Parameter1: Bryan Adams

Parameter2: song

SearchSite: Wikipedia.org
The query ""%22 " + BRYAN+ADAMS + "%22" + "+ " + "%22 " + SONG + "%22" + " site:" + WIKIPEDIA.ORG” resulted in a list of songs by Bryan Adams.

The longest prefix was “is a”, most repeated middle term was “written by” and there was no longest suffix.

When the query “IS A +*WRITTEN BY*+ site:http://en.wikipedia.org/wiki/” was passed to the search API, it resulted in more singers and their songs.

Iteration 1

Parameters:

Parameter1: Bryan Adams

Parameter2: song

SearchSite: Wikipedia.org

Analysis of Search Results:

Repeated middle term: WRITTEN BY

Longest common url: http://en.wikipedia.org/wiki/

Longest common prefix: IS A

Longest common suffix:

Result List of songs by Bryan Adams:

Everything I do I do it for you

Mysterious ways

Waking Up the Neighbours

Summer of 69
Heaven
Christmas Time
Run To You

Relevant parameters with similar relation:

Search Query to get more tuples: IS A + *WRITTEN BY* + site:http://en.wikipedia.org/wiki/


Iteration 2

(2a)

Parameters:

Parameter1: COCHRANE
Parameter2: SONG
SearchSite: http://en.wikipedia.org/wiki/

Analysis of Search Results:

No identifiable Repeated middle term.

Result List of songs by COCHRANE:

Life is a highway

(2b)

Parameters:

Parameter1: KNIGHT AND MIKE CHAPMAN

Parameter2: SONG

SearchSite: http://en.wikipedia.org/wiki/

Analysis of Search Results:

No identifiable Repeated middle term.

Result List of songs by KNIGHT AND MIKE CHAPMAN:

Silent Wings

(2c)

Parameters:

Parameter1: DYLAN

Parameter2: SONG

SearchSite: http://en.wikipedia.org/wiki/

Analysis of Search Results:

No identifiable Repeated middle term.

Result List of songs by DYLAN:

I shall be released
All along the watch tower

(2d)

Parameters:

Parameter1: EVANS

Parameter2: SONG

SearchSite: http://en.wikipedia.org/wiki/

Analysis of Search Results:

Repeated middle term: FEATURED

Longest common url: http://en.wikipedia.org/wiki/

Longest common prefix:

Longest common suffix:

Result List of songs by EVANS:

Trapped in the closet

This is home

Relevant parameters with similar relation:

Search Query to get more tuples: Song + *FEATURED* + site:http://en.wikipedia.org/wiki/


Parameters:

Parameter1: musician Tom Johnston
Parameter2: SONG

SearchSite: http://en.wikipedia.org/wiki/

Analysis of Search Results:

No identifiable Repeated middle term.

Result List of songs by MUSICIAN TOM JOHNSTON:

Nil

Parameters:

Parameter1: ADAMS AND JIM VALLANCE
Parameter2: SONG

SearchSite: http://en.wikipedia.org/wiki/

Analysis of Search Results:

Repeated middle term: WRITTEN BY BRYAN
Longest common url: http://en.wikipedia.org/wiki/
Longest common prefix: IS A
Longest common suffix:

Result List of songs by ADAMS & JIM VALENCE:

Summer of 69
Reckless

Relevant parameters with similar relation:
Search Query to get more tuples: Song +*WRITTEN BY BRYAN*+ 
site:http://en.wikipedia.org/wiki/

Tuple 3d: Parameter1: SONG, Parameter2: BRYAN-MICHAEL COX, 
searchsite: http://en.wikipedia.org/wiki/

Tuple 3e: Parameter1: SONG, Parameter2: BRIAN WILSON, searchsite: 
http://en.wikipedia.org/wiki/

Tuple 3f: Parameter1: SONG, Parameter2: BRIAN MOLKO, searchsite: 
http://en.wikipedia.org/wiki/

Tuple 3g: Parameter1: SONG, Parameter2: BRIAN MAY, searchsite: 
http://en.wikipedia.org/wiki/

Tuple 3h: Parameter1: SONG, Parameter2: BRYAN FERRY, searchsite: 
http://en.wikipedia.org/wiki/

Similarly, iteration 3 is also performed with these 8 tuples to get 
more songs and singers. Eventually after 4 iterations following 43 results 
were retrieved:

1. Everything I do I do it for you by Bryan Adams
2. Mysterious ways by Bryan Adams
3. Waking Up the Neighbours by Bryan Adams
4. Summer of 69 by Bryan Adams
5. Heaven by Bryan Adams
6. Christmas Time by Bryan Adams
7. Run To You by Bryan Adams
8. Reckless by Bryan Adams
9. Life is a highway by Cochrane
10. Silent Wings by Knight and Mike
11. I shall be released By Bob Dylan
12. All along the watch tower By Bob Dylan
13. Trapped in the closet by Evans
14. This is home by Evans
15. Freedom by Akon
16. Trouble Nobody by Akon
17. The Sweet Escape by Akon
18. I Wanna Love You by Akon
19. We Don’t Care by Akon
20. Pot of Gold by Akon
21. Hypnotized by Akon
22. I Can’t Wait by Sleepy Brown
23. Morning After Dark by Timbaland
24. Scream by Timbaland
25. Release by Timbaland
26. Good Vibrations by Brian Wilson
27. The Warmth of The Sun by Brian Wilson
28. Sleeping with Ghosts by Brian Molko
29. Battle for the Sun by Brian Molko
30. Pure Morning by Brian Molko
31. Without Im Nothing by Brian Molko
32. Tear The Signs Down by Brian Molko
33. Made in Heaven by Brian May
34. A Hard Rain by Brian Ferry
35. All Along the Watchtower by Brian Ferry
36. Avalon by Brian Ferry
37. Another Time Another Place by Brian Ferry
38. Positively 4th Street by Brian Ferry
39. You Won’t See Me by Brian Ferry
40. She’s Leaving Home by Brian Ferry
41. Street Life by Brian Ferry
42. I Put a Spell On You by Brian Ferry
43. Jealous Guy by Brian Ferry

Usually, the results tend to wander off after 4 iterations. Robust filtering techniques are required to make sure the results stay relevant to the experiment. Manual interventions are also mandated to make sure we get the right results.

For example, let’s say after analyzing the results we get a most commonly repeated middle term but no longest_common_prefix or a longest_common_suffix. In such cases, since we are looking specifically for songs in this particular example, we can add “song” as a prefix or a
suffix in the search query and try to induce results related the experiment. Brin did similar adjustments to the algorithm and there have been many researchers who proposed similar enhancements to DIPRE.
5.1 Conclusion

DIPRE has a simple generalization principle to control its expansion process based on longest matching prefix/suffix and most often repeated middle term. This does not always pick high quality tuples for next set of iterations. This leads to bad results and high recalls. Sometimes, there cannot be any iteration possible at all as the prefix, middle term and suffix found after the very first query can be erratic.

The primary difference between this experiment and the one did by Brin was that Brin’s focus was on extracting author&amp;books from the internet. This experiment applies the same principles on a broader set and looks to retrieve relations between any two parameters.

During my experiments with various parameters, usually after a couple of iterations, there were a lot of patterns generated by many seemingly valid results. However, these patterns tend to generate bad results after a while making the system unstable. Unless we develop good filtering algorithms, it is difficult to scale up DIPRE algorithm to build practical knowledge databases.

Infact, research mentions that this instability is a common problem in many bootstrapping algorithms [3].

However, to Brin’s credit, it has to be acknowledged that up to the point patterns started delivering divergent results; the data retrieved had
reasonable accuracy. His research has allowed others to come up with many models as enhancements to DIPRE; paving path for future research work in trying to make semi-supervised models better day by day.

5.2 Future Possibilities

There are a lot of practical applications that can be built based on machine learning. Examples could be to build knowledge bases, comparison-shopping [17], medical applications like counting red blood cells, recognition of cell tissues through microscopes to the detection of tumours in magnetic resonance scans and the inspection of bones and joints in X-ray images.

Recommendations for future enhancements to the current JAVA program.

1. Implement Snowball algorithm: This is an algorithm similar to DIPRE. While using DIPRE, system usually becomes unstable after a few iterations due to incorrect relations in results causing a snowball effect leading to bad patterns which introduce more bad relations. Snowball algorithm re-evaluates the patterns after each iteration and only the ones with highest confidence are kept for next iteration. Hence, Snowball gives more precise results. Unlabeled text as data: Feed unlabeled text to the algorithm instead of crawling web pages in the World Wide Web.
2. Using the algorithm, do a comparative study of results from various web search engines like Google AJAX Search API, Bing API and Yahoo BOSS API. Find a way to remove restrictions on usage of Google Search API. We can get only 64 results from a search query using Google AJAX Search API. Google sometimes suspects the queries from our application as automated queries from robots/spiders and stops responding.

![Google Query Restriction](image)

Figure 8: Google Query Restriction

Computer Science Department at UNLV should make a formal request to Google Inc. to remove these limitations for a certain “Google AJAX Search API Key”. This way, we can pass this key in the search queries from our applications and Google would know that UNLV Computer
Science Department is sending the query for educational purpose. A few Universities have done the same and Google Inc. has good-heartedly provided them with greater access.


[7]. Ellen Riloff. “Automatically constructing a dictionary for information extraction Tasks”, in proceeding of the eleventh national conference on artificial intelligence


[12]. Google AJAX Search API.  
http://code.google.com/apis/ajaxsearch/documentation/


[14]. Yahoo BOSS API.  
http://developer.yahoo.com/search/boss/boss_guide/

[15]. NewYork Times Article Search API.  
http://developer.nytimes.com/docs/article_search_api/

[16] JSON. http://www.json.org/

VITA

Graduate College
University of Nevada, Las Vegas

Praveena Mettu

Degrees:
  Bachelor of Technology in Computer Science and Engineering, 2008
  Jawaharlal Nehru Technological University, India

Thesis Title: Pattern Extraction from the World Wide Web

Thesis Examination Committee:
  Chair Person, Dr. Kazem Taghva, Ph.D.
  Committee Member, Dr. Ajoy K. Datta, Ph.D.
  Committee Member, Dr. Laxmi P. Gewali, Ph.D
  Graduate College Representative, Dr. Muthukumar Venkatesan, Ph.D