EIUAS: AN ENTERPRISE INFORMATION UPDATE AND ALERT SYSTEM

BY

JINDA HAN

THESIS

Submitted in partial fulfillment of the requirements for degree of Master of Science in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Advisor:

Professor Chengxiang Zhai
Abstract

Many companies today need a system to gather useful online information about products or customers, and then, such companies will improve their products or services consequent to this information and improve the functionality of the company. Thus, receiving a large amount and accurate information becomes the first important step in this process, which is followed by storing and retrieving such amounts of data. The next step in the process involves comparing these with the existing system database to update out-of-date data. Finally, future predictions and automatic information updates on available alert will be the last important step. Currently, there is a little research work on how to build such a system.

In this thesis, I propose a novel Update and Alert System (EIUAS). This system contains many parts, including, among others, a web crawler, data extractor, data verifier, data integration, dynamic database design, data retrieval engine, updatable checker, predict module, and updatable alert. The system would enable a company to receive all of the useful information about products or customers, and that information will be downloaded from public Internet mediums – related forums, customer reviews, blogs, social networks, etc. To evaluate the usefulness of this system as a baseline, we perform a calculation on the updatable percentage. Additionally, qualitative assessment by a domain expert also confirmed the system as very useful and with updatable information.
Acknowledgments

First, I would like to express my great appreciation for my advisor, Professor Chengxiang Zhai, who has been incredibly helpful not only on my thesis work, but also guiding me on how to perform strong research and building a useful system. He guided me through every part of this system from the beginning, and he gave me the freedom to explore all of the methods to build this system, and he motivated me to solve some specific detailed novel research problems. Further, he is very patient, respects my work and trusts my abilities. He generously shared his story and experiences with me. Without his support, this thesis would not be possible.

Also, thanks to Michelle Wellens, the Director of Advancement, and Colin Robertson, Program Specialist, who both are from the Department of Computer Science at the University of Illinois at Urbana-Champaign (UIUC). Thanks for their help on improving and verifying the system. Thanks to Professor Karrie Karahalios, Associate Professor at UIUC, who guided me on using a machine learning technique - logistic regression – for predicting the future via existing data. Thanks to others for supporting the entire project: Professor Rob A. Rutenbar, Bliss Professor and Head of Computer Science at UIUC, and Michael McClintick, Chief Information Officer at University of Illinois Foundation (UIF). Additionally, thanks for the extra help from Patrick Nowlan, Associate Director, Portfolio Information Management at UIF; Richard Sowers, Professor from the Department of Industrial and Enterprise Engineering at UIUC; Tyler
Fitch, Associate Vice President and Planning & Policy Analysis at UIF; and Steve Krause, Network and Systems Administrator at UIF.

Finally, thanks to my parents for their support and encouragement throughout my studies, and especially my Mother, who helps me during difficult times. Thanks to the Computer Science Department at the University of Illinois at Urbana-Champaign and University of Illinois Foundation for providing great resources and providing me with excellent experience and opportunities during both undergraduate and graduate study.
# Table of Contents

Chapter 1 Introduction ......................................................................................... 1

1.1 Motivation and Background ........................................................................ 1

1.2 Related Works .............................................................................................. 2

1.3 Contributions ............................................................................................... 2

Chapter 2 Problem Formulation ......................................................................... 4

2.1 Problem Overview......................................................................................... 4

2.2 Requirements and specifications ................................................................... 4

Chapter 3 System Design .................................................................................... 7

3.1 Web Crawling................................................................................................. 7

3.2 Data Process ................................................................................................ 10

  3.2.1 Database Design ................................................................................... 10

  3.2.2 Data Scoring and Similarity ................................................................ 13

  3.2.3 Data Extraction .................................................................................... 14

  3.2.4 Data Integration ................................................................................... 16

3.3 Verification System Design .......................................................................... 18

  3.3.1 System Automatic Update Checking .................................................... 18

  3.3.2 Human Verify System ......................................................................... 19

3.4 Prediction Module Design .......................................................................... 22

3.5 Update Alert System Design ....................................................................... 24

Chapter 4 Evaluation .......................................................................................... 25

4.1 Dataset .......................................................................................................... 25

4.2 System Usability and Results ....................................................................... 25

Chapter 5 Conclusion .......................................................................................... 27

5.1 Conclusion .................................................................................................... 27

5.2 Future Plan ................................................................................................... 27
References........................................................................................................................................ 29
Appendix A: Database Structure Design....................................................................................... 31
Chapter 1
Introduction

1.1 Motivation and Background

Many companies and organizations today need a system to gather useful online information in order to maintain their products and customers. Companies and organizations may improve their products or service through this information, which, ultimately, improves company performance. In the age of Big Data, it is possible to gather a great deal of useful information online. Therefore, acquiring a large amount and accurate information becomes the first important step in this process. Then, storing and retrieval of such an amount of data becomes the next important part, which is followed by a comparison of these with the existing system database in order for companies and organizations to update out-of-date data. Finally, future predictions and automatic information update available alerts will be the last important step. The main motivation is that, currently, there is no or little research on how to build this kind of system for other companies to achieve this goal. Significantly, this kind of system is strongly demanded by many companies and organizations.

In this thesis, I leverage a previous project [1] and extend it to develop a more complete system.
1.2 Related Works

There are some related works for some parts of this system. Rui Wang [1] and Fan Fu have completed parts of the web crawling system, which is a good starting point for me on this project. Some papers discuss technique, especially how to mine and extract personalized structured data through social network [2], or Web Pages [3, 4, 5, 6], and to build a data-driven system [7]. [8] and a book [9] talk on switch database from MySQL to MongoDB. Papers [10, 11] offered a reference for how to find the most relevant result on related query, used this method to verify the correct product or a person, and also to find the most ranked the most related review or news articles for a product or a person. Then, paper [12 - 26] offered many methods for how to use machine learning techniques on people, and [27] will include a brief introduction on how to use data visualization tools to show the results.

1.3 Contributions

In this thesis, I propose an Enterprise Information Update and Alert System (EIUAS). This system contains many parts, including, among others, a web crawler, data extractor, data verifier, data integration, dynamic database design, data retrieval engine, updatable checker, predict module, and updatable alert. Finally, receive all of the useful information about the products or customers, and that information will be derived from the public Internet and related forums – customer reviews, blogs, social networks, etc. To evaluate the usefulness of this system as a baseline, we perform a calculation on the updatable percentage, and invited a domain expert to perform a human verification on
the amount of updatable information. The experts confirmed the system as very useful and with a great deal of updatable information.
Chapter 2
Problem Formulation

2.1 Problem Overview
To overcome the limitation of human collection of data when maintaining the database, it was necessary to build such a system to help people produce a large amount of data in a very short time. It became clear that it was crucial to build many sub-systems to achieve the entire goal. In this instance, thinking from the end point is particularly productive. The questions become, if we have a particular amount of data, how do we know the future update information, and how do I use the data to predict future products or future customers? If I need a particular amount of data, how do we know those data are newer than the data in my database? If we know these data are different and newer than the one in my database, how do I verify that the data matched the products or the person? If I know that I have a high percentage accurate, and this is the correct product or person, how do I store the data; and, moreover, how do I add more columns for more features in the database easily in the future? Where and how do I get this amount of data? If I know how to get those data and crawl them through the public Internet, how do I initialize what data I should crawl? I will start on the Web Crawling and determine what kind of data to crawl first.

2.2 Requirements and specifications
In the pursuit of building this system, I made use of following tools and technology:
System Requirements

- This system running work on *Ubuntu 16.04 LTS 64-bit version*

Software Requirements

- *Apache 2* as the server for the system
- *Nginx* as the tools server
- *Python 2.7* for back-end programming
- *PHP 7* with *PHP-MongoDB driver*
- *Node.js* with *Express*
- *AngularJS* mainly for front-end data interaction
- *Foundation* mainly for front-end interaction design
- *MongoDB* for NoSQL database
- *PyLucene* for indexing and searching the documents in Python version

Other Libraries and Frameworks

- *Scikit-learn*: An open source machine learning library for Python
- *NLTK 3.0*: a leading platform for building Python program to work with human language data.
- *Scrapy*: An open source and collaborative framework for extracting the necessary data from websites
- *Beautiful Soup 4*: A Python library designed for quick turnaround projects like screen-scraping
- *NumPy*: A Python library for scientific computing
- **SciPy**: An Open Source library of scientific tools for Python
- **PHPCrawl**: a PHP web crawler library/framework
- **D3.js**: A JavaScript library for producing dynamic and interactive data visualizations in the web browsers.
Chapter 3
System Design

To build Enterprise Information Update and Alert System (EIUAS), we need to build some sub-systems, such as Web Crawling System, Data Extracting and Integration System, etc. The general architecture of EIUAS is as follows:

Figure 3.1: The Enterprise Information Update and Alert System Architecture

3.1 Web Crawling

Web Crawling is the first important part of data collection for fulfilling the needs of a company or organization. The entire public Internet is a great resource, and many companies offer an API for other developers using their certain data, such as Google
provide Google Search API, Google Map API, LinkedIn provide LinkedIn API, ZabaSearch provide ZabaSearch API, Twitter provide Twitter API, etc.

First, I need to secure enough real-time data from the public Internet. Previous researchers, Fan Fu and Rui Wang [1] used Java with codec lib to gathering data from the web, for example, Google Search API, LinkedIn API, ZabaSearch API, etc. I used this resource as a starting point for designing the crawling system. Unfortunately, however, most of the code that required library and API were both expired. Thus, it became necessary for me to reapply the API and rebuild the crawler.

I began to discover the productivity of a Java crawler, which is worse than other programming, for instance Python, PHP, JavaScript, among others. I tried many methods, like using JavaScript, to build a node.js tiny server to achieve a special requirement, such as dynamic address, random time generator crawling, or JavaScript for building Chrome plug-ins to script web information. Also, I used PHPCrawl¹ library to fetch dynamic web pages. Although JavaScript and PHP offer a good crawler library, it is, nevertheless, not sufficient, given their shortcomings. After seriously consideration, I decided to try Python libraries, which are Scrapy² and Beautiful Soup³. Scrapy is an entire web crawling framework. Moreover, Beautiful Soup is a library that focuses only on data parsing. In other words, Scrapy has more functions than Beautiful Soup in web crawling, but Beautiful Soup mostly focuses on parsing the data. Finally, I decided to use both as my new crawling system. First, I would scrap all of the pages using Scrapy and store them in HTML (for images, I would save the link only). Then, I would use

² Scrapy: “An open source and collaborative framework for extracting the data you need from websites.”
³ Beautiful Soup: “A Python library designed for quick turnaround projects like screen-scraping.”
BeautifulSoup4 to conduct a post-processing, where I extract only the special useful information and then store it in the database.

After the initial setup, I started to build the pre-processing, which was based on certain key words necessary to search on the web page. For example, first, you need to obtain the list of products or customer information, and then you need to search a product or a customer on different web pages. For the most part, searching products will start from websites, such as Amazon, eBay, Google, etc. Moreover, searching people usually starts with websites like LinkedIn, Google, Whitepages, Twitter, etc. For a person, the attributes you might need to include are “first name”, “last name”, “state”, etc. To achieve a more precise search, you might add a “middle name”, “position”, “company”, and “university”, among other words.

Then, I would provide a loop on each product or person on the list, and I would use the entities’ combined query to conduct a search on each website search engine. Then, after the query, we would obtain rich information and links through the results pages. I constrained a limited number of result pages to store as a “seed page.” Next, I would also save one more layer page as the target page. Another two important key features are random time and dynamic IP address. For the break interval between each page, it is necessary to use a random time. For example, I set the time from 0.1 to 2.0 seconds. Next, it becomes necessary to use a dynamic IP address for the local server, since some websites will block access once the limit time-visit amount is reached. I added this feature though Scrap two components, RandomProx ⁴ and RotateUserAgentMiddleware. For the host proxy list, it is not difficult to find a number of free hosts on the Internet.

---

⁴ Random proxy middleware for Scrapy: https://github.com/aivarsk/scrapy-proxies.


### 3.2 Data Process

In order to use the web page data that I collected, we need to extract, verify, integrate and store the data into the database. Before data extraction, we need to design the database schema.

#### 3.2.1 Database Design

Imagine that we crawled gigabytes of data from the public Internet. The question becomes, how to extract and integrate this data and save it to the database? To address these questions, I will first address database design.

At the start, I designed the system relational database in MySQL, which includes certain columns, such as “ID”, “first name”, “last name”, “headline”, “location”, “industry”, […], “position 1”, “company 1”, […], “position 3”, […], “university 1”, “degree 1”, “major 1”, “year 1”, […], “year 3”, “twitter”, “linkedin”, […], “news 1”, “date 1”, “source 1”, […] “source 2”. Given this data increment, I found that it is especially difficult for me to add extra columns. It takes a great deal of time for each modification, such as adding a column for “company 4”, “university 4”, etc. Moreover, it also wastes a lot space for the row, which does not include a value for the columns. It maintains has a “null” value.

To solve this problem for an existing system with a large amount of data becomes a massive construction. After careful consideration and surveying, I learned of a NoSQL type data storing method to solve this problem. NoSQL is another storage method instead of tabular relations for SQL databases, which offers more flexible relational database tables. It offers the possibility to easily expand any “field” (“column” in SQL); moreover, it easily addresses more complicated “joined tables”. Therefore, the finalized
database I used MongoDB, and the initialized data, which was extracted from the HTML page, was then saved to MongoDB. A partial example database structure design for 

**Customer** is as follows:

Collection: “customers”

```javascript
{
   "_id" : ObjectId("1"),
   "companyID" : "10001",
   "name" : {
      "first_name" : "John",
      "middle_name" : "M",
      "last_name" : "Doe"
   },
   "headline" : "CEO",
   "location" : {
      "current" : {
         "20110101" : "Mountain View, CA",
         "url" : "http://www.abc.com/1234567890"
      },
      "past" : {
         "20080101" : "Chicago, IL",
         "url" : "http://www.abc.com/1234567890"
      }
   },
   "industry" : "Internet",
   "gender" : "male",
   "photo" : {
      "20160101" : "http://www.abc.com/1234567890.png",
      "20150102" : "http://www.def.com/1234567890.png"
   }
   "spouse" : {
      "current" : {
         "id" : ObjectId("666"),
         "first_name" : "Jane",
         "middle_name" : "F",
         "last_name" : "Doe"
      },
      "past" : {
      }
   }
   "email" : {
      "current" : {
         "20110101" : "jmd1@gmail.com",
         "url" : "http://www.abc.com/1234567890"
      },
      "past" : {
         "20060101" : "jmd3@gmail.com",
         "url" : "http://www.ghi.com/1234567890"
      }
   }
   "address" : {
      "current" : {
```
"20110101": "1234 Happy St, Champaign, IL",
"url": "http://www.abc.com/1234567890"
},

"phone": {
    "current": {
        "20110101": "1234567890",
        "url": "http://www.abc.com/1234567890"
    }
},

"web_link": {
    "company": {
        "current": {
            "20110101": "http://www.abcedf1.com",
            "url": "http://www.abc.com/1234567890"
        }
    },
    "blog": {
        "current": {
            "20110101": "http://www.abcedf1.com",
            "url": "http://www.abc.com/1234567890"
        }
    },
    "personal": {
        "current": {
            "20110101": "http://www.abcedf1.com",
            "url": "http://www.abc.com/1234567890"
        }
    },
    "linkedin": "http://www.linkedin.com/1234567890",
    "facebook": "http://www.facebook.com/1234567890",
    "twitter": "http://www.twitter.com/1234567890"
},

"summary": {
    "current": {
        "20110101": "I am a CEO of XYZ company, I like playing piano.",
        "url": "http://www.abc.com/1234567890"
    }
},

"work": {
    "current": {
        "position": "CEO",
        "company": "XYZ",
        "date": {
            "start_date": "20100101",
            "period_years": "6.5"
        },
        "location": "1234 Happy St, Mountain View, CA",
        "Description": "My company is great",
        "20160101": "http://www.abc.com/1234567890"
    }
},

"education": {
    "past": {
        "university": "MIT",
        "degree": "MS",
        "major": "Computer Science",
        "20110101": "http://www.abc.com/1234567890"
    }
}
3.2.2 Data Scoring and Similarity

Before building the classifier for each website and extracting data from HTML files, I needed to integrate the data in relation to the correct person. To verify and select the relevant web pages from the HTML pools, I indexed the documents and performed a search of previous work in [1]. The difference in this instance is that I used PyLucene instead, which is the Python version of Lucene. I transformed the raw HTML pages into text documents after removing the HTML tags. Then, I used PyLucene to build the index for each HTML page.

For scoring, the use of similarity would be more feasible for my data source since each document has the products of person information details along with many entities. Consequently, a similarity matching would provide results that are more relevant. I proposed BM25L [10], a simple yet executive extension of BM25, in order to overcome the problem of BM25, which tends to over-penalize very long documents. With the same computation cost, this is more effective and robust than BM25. The formulae are as follows:

5 Website (e.g. Product: Amazon, eBay, etc. Person: LinkedIn, Facebook, etc.).
6 PyLucene is a Python extension for accessing Java Lucene. http://lucene.apache.org/pylucene/
\[
\sum_{q \in Q \cap D} \frac{(k_3 + 1)c(q, Q)}{k_3 + c(q, Q)} f(q, D) \cdot \log \frac{N + 1}{df(q) + 0.5}
\]

\[
f'(q, D) = \begin{cases} 
\frac{(k_1 + 1) \cdot [c'(q, D) + \delta]}{k_1 + [c'(q, D) + \delta]} & \text{if } c'(q, D) > 0 \\
0 & \text{otherwise}
\end{cases}
\]

\[
c'(q, D) = \frac{c(q, D)}{1 - b + b |D| \text{avdl}}
\]

I applied Rocchio feedback [11] to adjust the result with best/worst results top words. But, in order to avoid over adjusting, I set the adjustment weight according to retrieval ranking. And, I also set the limit to a positive adjustment (smaller than the original query) and the negative adjustment (smaller than the positive constant). This feedback approach is usually robust and effective, but it would provide a better result in combination with explicit relevance feedback, which requires server recording. Finally, the system returns list was processed for each query and ranked by similarity scores.

### 3.2.3 Data Extraction

Based on the database structure, it is evident how to store the data, and further expansion is not difficult. For the next steps, I needed to extract and integrate the unstructured data into a structured database. Before extracting the data, it is necessary to build classifiers before saving the data and trying to ensure that the data is stored under the same name. This is particularly important for future steps, as I build the text classifiers, as shown in the following partial sample database structure design for the classifier of Names:

Collection: “name_classifier”
Once a classifier is established, it becomes possible to process the HTML files. For different websites, the design structure will be quite different. Nevertheless, it is not difficult to learn of common parts through the related web design [7].

Thus, it was necessary for me to build a classifier for different web designs, for example on “name”: 
3.2.4 Data Integration

Upon establishing the most relevant web pages for each product or person query, and given the prepared datasets, the next step involves integrating the data into the database and the display in the front-end web page. Below is an example of several relevant web pages, which are integrated into one single web page.
Figure 3.3: The Example Demo Web Page or a Person.  

Note: the content in Figure 3.3 was artificially generated.
3.3 Verification System Design

There are two main parts for verifying the system. The first is that the system automatically compares and verifies, and the second is the human verify system interface.

3.3.1 System Automatic Update Checking

For this part, first, I need determine if this product or this person is the correct one that I wish to find. Thus, I will consider some important entities as key to verify with the company’s existing database. For example, to check a person, I will consider the entities from the crawled database, such as “first_name”, “middle_name”, “last_name”, “headline”, and whole list for “location”, “email”, “address”, “email”, “phone”, “web_link”, “work”, and “education”.

The following is sample Pseudo code for partial update system algorithm:

```plaintext
for each first_name and last_name matched case (set entity E),
    for each entity in entities list
        //case 1, if they have accurate match
        if (phone or email or address or web_link) matched with same entity in company’s database:
            if the matched Eméd data’s date is older than current Ecurr’s date:
                mark as updatable
                mark block green
            else if matched Eméd data’s date is current Ecurr’s date and other E’s date not empty:
                if other E’s date is older
                    mark as old data
                    do nothing
                else if other E’s date is same with Ecurr’s date
                    mark as possible updatable
                    mark block as yellow
```

---

8 web_link: Linkedin, Twitter, blog, Facebook, etc.
else if (matched $E_{med}$ data’s date is empty or current $E_{curr}$’s date is empty)
mark as possible updatable
mark block as yellow

else
mark as error

// case 2, check the possibility that they match ...
if a person has either updatable or possible updatable mark
add to human verify list
else
return

And finally, we arrive at a list of matched and possible updatable products or customers, then next step is a human double verify system.

### 3.3.2 Human Verify System

The human verify system is designed as in the following sample.

![Figure 3.4: Example Demo for the Human Verify System (Part 1)](image)
From the Figure 3.4 and Figure 3.5, we can recognize easily that there are different data. The upper row comes from the web, and the lower rows come from the company’s database. First, I will address the virtual person, John Doe, who currently is the CEO of XYZ Company, but the company’s database records him as a Software Engineer at Google. After system verify, it is evident that Joe was formerly a Software Engineer at Google, but this information is out of date, so it is marked as a green label, which means, that there is a very high possible that Joe moved from his position as a Software Engineer at Google to the CEO of XYZ company. This verification continued by checking his phone number; it did not change. This means, then, that it should be the correct person. Further, a user can monitor his LinkedIn or look for information that is more detailed by clicking on the “view” button. Next, let us check the second virtual person, Jane Doe, who holds the same job function, according to the headline, but actually holds a different position title because she was promoted to the level of a Program Manager 2 in the same company. But, she changed her phone number.
The question becomes, what do humans need to do? If the verifier thinks that this is the correct person, given an analysis of simple or detailed information, then the person needs to check on each entity by comparing it against the web and the company’s data. If the verifier decides that the company’s data is out of date and wants to replace it via web data, then the verifier needs to click on the button. Consequently, the web data will directly replace that entity on the cloned company’s database. If the verifier decides that the web data is out of date, the verifier needs to click on the button. At which time, the web data will be directly replaced by the company’s data. If the verifier thinks that this is a wrong person match, the verifier should click on the “WrongMatch?” button, and the system will automatically remove this pair through the matched result. If the verifier thinks that all the data of a person or goods from the Internet are older than the company’s system, then the verifier should click on the “Useless?” button, which means that this person will be marked and not appear in any matching situation until the information is updated.

After a human monitor manually verifies all of the data, the verifier may click on the “Report” page. There is some information on the report page, such as the total web database number, company database number, as well as a display of updatable products or customers. If the user wishes to find the result of the verified data, the user needs to click on the “Generate a report” button, and then the system will automatically generate a csv form with the user’s login name, and all of the entities will be changed.
### 3.4 Prediction Module Design

After a great deal of useful and verified data is gathered, it becomes possible to employ some machine learning techniques in order to predict some features, such as the selling trend of a product, the continuity of customer spending on a company’s product, or who will be the most likely potential customer on the list, etc. In this case, I use the potential customer prediction as an example.

First, a question that might come to our attention is why a potential customer is so important. To a company, a high amount of potential customers can help the company to maintain its ranking and standing of excellence; it may support employees with a stronger company community. Compared with a human-labeled potential customer list, the potential customer prediction will save a lot of time and materials for human resources, and it will increase contact efficiency. Also, it might find the customer who will be easily missed by human collection alone.

The next question is how to solve this problem? To begin, I will use my existing system and crawl some customer information. Simultaneously, I can easily find previous public customer history for 2013 [29] and 2014 [30].

![Image: Categories Separated by Financial Expenditure](image)

Figure 3.6: Categories Separated by Financial Expenditure [29, 30]

The company separates consumers into four categories: “PLATINUM” refers to people who spend $25,000 or more, “GOLD” refers to people who spend between
$10,000 - $24,999, and so on. To build such a system, I took some entities as parameters, such as:

- Basic information: “headline”, “location”, “industry”, etc.
- Working: “position”, “company”, “period”
- Education: “intuition”, “degree”, “majors”, “years”
- Extract from Headline, Position (such as student, professor, CEO/Chief Executive Officer/president, CTO/…., etc.)
- Label: spend [“yes”, “no”], correct(unknown)

Datasets: more than 4000 identity customer information sets, which were built above.

For the purposes of prediction, I will use logistic regression, which allows me to predict a categorical outcome using categorical and numeric data. I will use 70% of data as training, 10% testing, and 20% for development. The regression was completed at a significance level of $p = 0.01$, and the complete model, including variable coefficients and individual p-values, are as follows in the adjacent table:

Table 3.1: A Simple Logistic Regression Result

<table>
<thead>
<tr>
<th>Variable</th>
<th>Coefficient</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>headline</td>
<td>0.2684</td>
<td>0</td>
</tr>
<tr>
<td>location</td>
<td>0.9812</td>
<td>0</td>
</tr>
<tr>
<td>industry</td>
<td>-0.319</td>
<td>0.1002</td>
</tr>
<tr>
<td>position</td>
<td>0.9491</td>
<td>0</td>
</tr>
<tr>
<td>company</td>
<td>0.2156</td>
<td>0</td>
</tr>
<tr>
<td>Period</td>
<td>-0.8714</td>
<td>0</td>
</tr>
<tr>
<td>Degree_year</td>
<td>1.147</td>
<td>0.0021</td>
</tr>
<tr>
<td>Binary (Intuition, Stanford Uni.)</td>
<td>3.7862</td>
<td>0</td>
</tr>
<tr>
<td>Binary (Intuition, Uni. Of I.)</td>
<td>0.7652</td>
<td>0</td>
</tr>
<tr>
<td>Binary (degree, Bach. of Sci.)</td>
<td>2.006</td>
<td>0.0032</td>
</tr>
<tr>
<td>Binary (degree, Mast. Of Sci.)</td>
<td>1.046</td>
<td>0</td>
</tr>
<tr>
<td>Binary (majors, art and design)</td>
<td>-0.385</td>
<td>0.0007</td>
</tr>
<tr>
<td>Binary (majors, computer science)</td>
<td>4.095</td>
<td>0</td>
</tr>
</tbody>
</table>
3.5 Update Alert System Design

Based on the prediction module, I will have the priority ranked list for each person. I will determine, for instance, who is most likely to spend more money in the future. Then, I let the system automatically search the Internet for the products or customers with higher scores for a certain time interval. For example, news for every 12 hours and information for every week (since they are not updating personal information that frequently).

As regards each person query to search the web, if a product or a person does not have enough information, the query will run through 3.1 to 3.3 again. On the contrary, if the person has enough information, then the system will target related news, Twitter, etc. After, the system finds the HTML downloaded and all of the articles related to this person, which begins immediately following the most recent previous article. For example, if the database is the same as the following table, then the system will only save the data after 2016-1-1. A partial example database structure design for News is as follows:

```
"news" : {
    "_news_id" : {
        "date": "20160101",
        "url" : "http://www.hhh.com/123567",
        "content" : "This person ranked number 1 rich in his state"
    }
}
```

If the situation is that the news does not have a date, the system will use Python NLTK library to match the similarity of the article with existing articles. If the article is entirely different, then I will add it to the database. Additionally, the user can easily modify the keywords and classifiers further.
Chapter 4
Evaluation

4.1 Dataset
In general, it is necessary to evaluate the usefulness of the Enterprise Information Update and Alert System. First, I used the automatic system and generated an updatable list, which are 973 customers. This list was derived from a total number of 4879 extracted customers. It is highly possible that this information is updatable.

Table 4.1: A Database Simple Statistics

<table>
<thead>
<tr>
<th></th>
<th>Total number of Customer in CompanyDB is: 11598 and Web Crawling DB is 4879</th>
</tr>
</thead>
<tbody>
<tr>
<td>People can update:</td>
<td>973</td>
</tr>
<tr>
<td>percentage</td>
<td>8.38937748%</td>
</tr>
<tr>
<td>People finally updated:</td>
<td>0 and percentage 0%</td>
</tr>
</tbody>
</table>

For the NoSQL Database, it is very easy to expand additional functions, such as adding a mark on which the entity of a person or a product has been update. It has greater flexibility than the SQL database, which often requires more tables in order for the function to work.

4.2 System Usability and Results
Experiments indicate that our system can effectively retrieve relevant HTML pages and extract 973 updatable customers, accounting for 8.39% of the total 11598 customers. This figure is a high enough percentage, and it outperforms a simple baseline approach. To further adjust the usability of the system, months ago, I invited a domain expert to
perform a human verification on the amount of updatable information. The expert finally confirmed that the system indeed is very useful and also that a great deal of customer information may be updated.
Chapter 5
Conclusion

5.1 Conclusion

In this thesis, I proposed Enterprise Information Update and Alert System (EIUAS) for helping a company or organization to improve their current database from web public data. The EIUAS system starts from a very basic line, which is crawling the web information, then scoring for the similarity of documents, extract, and integration of the unstructured data into an easier expandable non-relational database. Then, I built a verification system for both computer and human participation, and it generates the finalized data and exports it as a CSV file. Moreover, to take full advantage of the data, I also performed a very simple logistic regression based on previous customers spending. Also, I created some features from the results to build a news information alert system, which may have customizable alert information for the control over the keywords by users.

5.2 Future Plan

Since the logistic regression implemented in this system is quite simple, in the future, I may further explore more complicated prediction algorithms from the currently dataset. Moreover, I build only a simple fuzzy search engine for the system, but I would like to design it to be more like a smart entity search engine. For example, if the user inputs a query, such as “display all the persons currently in Bay Area”, the system will
automatically present the result showing persons currently living in the Bay Area. Those kind of intelligent search engines seem to “understanding” the query from human users, which would be amazing.
References


[27] X, Zhang. Mining Lego Data Sets to Support Lego Design. Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2015.


Appendix A

Database Structure Design

The complete MongoDB (NoSQL) database structure design for a customer:

```
{
    {
        "_id" : ObjectId("1"),
        "companyID" : "10001",
        "name" : {
            "first_name" : "John",
            "middle_name" : "M",
            "last_name" : "Doe"
        },
        "headline" : "CEO",
        "location" : {
            "current" : {
                "20110101" : "Mountain View, CA",
                "url" : "http://www.abc.com/1234567890"
            },
            "past" : {
                "20080101" : "Chicago, IL",
                "url" : "http://www.abc.com/1234567890"
            }
        },
        "industry" : "Internet",
        "gender" : "male",
        "photo" : {
            "20160101" : "http://www.abc.com/1234567890.png",
            "20150102" : "http://www.def.com/1234567890.png"
        }
    },
    "spouse" : {
        "current" : {
            "id" : ObjectId("666"),
            "first_name" : "Jane",
            "middle_name" : "F",
            "last_name" : "Doe"
        },
        "past" : {
            }
    }
}
```
"email": {
    "current": {
        "20110101": "jmd1@gmail.com",
        "url": "http://www.abc.com/1234567890"
    },
    "current2": {
        "20090101": "jmd2@gmail.com",
        "url": "http://www.def.com/1234567890"
    },
    "past": {
        "20060101": "jmd3@gmail.com",
        "url": "http://www.ghi.com/1234567890"
    }
},
"address": {
    "current": {
        "20110101": "1234 Happy St, Champaign, IL",
        "url": "http://www.abc.com/1234567890"
    },
    "past": {
        "20080101": "5678 Good Drive, Urbana, IL",
        "url": "http://www.abc.com/1234567890"
    }
},
"phone": {
    "current": {
        "20110101": "1234567890",
        "url": "http://www.abc.com/1234567890"
    }
},
"web_link": {
    "company": {
        "current": {
            "20110101": "http://www.abcedf1.com",
            "url": "http://www.abc.com/1234567890"
        },
        "past": {
            "20080101": "http://www.abcedfo.com",
            "url": "http://www.abc.com/1234567890"
        }
    },
    "blog": {
        "current": {
            "20110101": "http://www.abcedf1.com",
            "url": "http://www.abc.com/1234567890"
        }
    }
}
"personal" : {
  "current" : {
    "20110101" : "http://www.abcedf1.com",
    "url" : "http://www.abc.com/1234567890"
  },
  "past" :{
    "20080101" : "http://www.abcedfo.com",
    "url" : "http://www.abc.com/1234567890"
  }
},
"linkedin" : "http://www.linkedin.com/1234567890",
"facebook" : "http://www.facebook.com/1234567890",
"twitter" : "http://www.twitter.com/1234567890"
},
"summary" : {
  "current" : {
    "20110101" : "I am a CEO of XYZ company, I like playing piano.",
    "url" : "http://www.abc.com/1234567890"
  },
  "past" :{
    "20080101" : "Looking for fulltime job on software development",
    "url" : "http://www.abc.com/1234567890"
  }
},
"work" : {
  "current" : {
    "position" : "CEO",
    "company" : "XYZ",
    "date" : {
      "start_date" : "20100101",
      "period_years" : "6.5"
    },
    "location" : "1234 Happy St, Mountain View, CA",
    "Description" : "My company is great",
    "20160101" : "http://www.abc.com/1234567890"
  },
  "past" :{
    "position" : "Software Engineer",
    "company" : "ABC",
    "20080101" : "http://www.abcedfo.com",
    "url" : "http://www.abc.com/1234567890"
  }
}
"date": {
    "start_date": "20090101",
    "end_date": "20100101",
    "period_years": "1"
},
"location": "1234 Happy St, Mountain View, CA",
"Description": "This company is great",
"20160101": "http://www.abc.com/1234567890"
},
"education": {
    "past": {
        "university": "MIT",
        "degree": "MS",
        "major": "Computer Science",
        "year": {
            "start_year": "2005",
            "end_year": "2007"
        },
        "20160101": "http://www.abc.com/1234567890"
    },
    "past2": {
        "university": "UIUC",
        "degree": "BS",
        "major": "Computer Science",
        "year": {
            "start_date": "2001",
            "end_date": "2005"
        },
        "20160101": "http://www.abc.com/1234567890"
    }
},
"news": {
    "_news_id": {
        "date": "20160101",
        "url": "http://www.hhh.com/123567",
        "content": "This person ranked number 1 rich in his state"
    }
}
}