Information Extraction on Para-Relational Data

by

Zhe Chen

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2016

To My Parents

ACKNOWLEDGEMENTS

The PhD journey is full of ups and downs, and I would not be able to complete it on my own. I owe many thanks to my advisor, Michael Cafarella. I learned from him to think big and creatively. I am deeply influenced by his research style to build real, innovative systems that have impact, instead of sticking to small papers. I also learned from him to be a good writer and storyteller. English is my second language, and I did not know how to write a paper on my own when I first came to Michigan. Mike was very patient to help me with all kinds of paper re-organizing and re-writing, especially during my first several years. Without Mike's continuous guidance and help throughout my PhD studies, I would not be able to finish this thesis. So Mike — thank you.

I would like to express my thanks to all my committee members. Professor Eytan Adar is an amazing person to work with. He is hands-on and knows everything. I have wanted to be a person like Professor H. V. Jagadish. He gives so much care to all the students and is always so nice to everybody. Professor Qiaozhu Mei is also very nice and was willing to give me advice on being an international student in the US. Professor Barzan Mozafari is very helpful and gave many insightful suggestions.

I am so grateful that I can work with all these fantastic people at the University of Michigan. Thanks to the database group for their valuable discussions and help: Michael Anderson, Dolan Antenucci, Matthew Burgess, Daniel Fabbri, Lujun Fang, Arnab Nandi, Fei Li, Yunyao Li, Bin Liu, Yongjoo Park, Li Qian, Manish Singh, Jing Zhang, and others. Thanks to all the fellow students in the School of Information:

Wei Ai, Xin Rong, Jian Tang, Shiyan Yan, Yue Wang, and many others.

The people at Tableau are my friends and mentors, and they graciously allowed me to publish the research work and put it in the dissertation. Thanks to Jock Mackinlay, Sasha Dadiomov, Richard Wesley, Daniel Cory, Gang Xiao, and many others for all kinds of support and keen insights. By working with them, I learned to use research to impact the real product.

I also want to thank my mentors and friends in my undergraduate studies. The internship in MSRA working with Zaiqing Nie changed my life: I decided to pursue a PhD degree at that time. Zaiqing's research style also influenced my way of critical thinking. My fellow students at Renmin University, Bolin Ding and Min Xie, were willing to help, even before I formally met them.

My roommates and friends in Ann Arbor are my family in the United States. I moved to live with Lei Lei during the darkest time of my PhD studies, and I had not met her before. She welcomed me with such a sweet home. I am grateful that I can spend the last year of my PhD living with Jing Zhang and Shuyi Zhang, and I will not forget the times when we were roommates. I will also not forget the coffee time with Yue Liu when we were on fire discussing the interview questions. Thanks to Cheng Li for all the happiness that we shared and Zhe Zhao for all the hot pot parties.

Lastly, I would like to express my deepest love to my parents and my family. The PhD study is rewarding, but full of all kinds of obstacles. This happy ending would not be possible without their support and comfort along the way. I have never imagined that I would begin my postgraduate journey with a job in a hedge fund company, but I am sure they will all be proud of me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Information Extraction on Para-Relational Data

by

Zhe Chen

Chair: Michael Cafarella

Para-relational data (such as spreadsheets and diagrams) refers to a type of nearly relational data that shares the important qualities of relational data but does not present itself in a relational format. Para-relational data often conveys highly valuable information and is widely used in many different areas. If we can convert para-relational data into the relational format, many existing tools can be leveraged for a variety of interesting applications, such as data analysis with relational query systems and data integration applications.

This dissertation aims to convert para-relational data into a high-quality relational form with little user assistance. We have developed four standalone systems, each addressing a specific type of para-relational data. *Senbazuru* is a prototype spreadsheet database management system that extracts relational information from a large number of spreadsheets. *Anthias* is an extension of the Senbazuru system to convert a broader range of spreadsheets into a relational format. *Lyretail* is an extraction system to detect long-tail dictionary entities on webpages. Finally, *DiagramFlyer* is a web-based search system that obtains a large number of diagrams automatically extracted from web-crawled PDFs. Together, these four systems demonstrate that converting para-relational data into the relational format is possible today, and also suggest directions for future systems.

CHAPTER I

Introduction

1.1 Para-relational Data

In recent decades, a variety of tools, such as SQLServer, MySQL, and DB2, have been developed to manage data in the relational format. Some other tools can query the relational data or integrate many different relational data sources. These tools all require the stored data to be in a strictly relational format. Unfortunately, the massive amount of data available today is often not in a strictly relational format, which makes it difficult to use most of the existing tools to manage the data.

Studies have been conducted to convert specific types of data into a relational form to manage the huge amount of data using the tools built on relational data. The most prominent types of data studied so far have been semi-structured (such as XML) or unstructured data (such as free text).

Semi-structured data often refers to a form of structured data that does not conform with the formal relational format, but does contain tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data.[1] Semi-structured data is often self-describing, so it is easy to obtain the embedded semantics by parsing the embedded structures. For example, Figure 1.1 provides an example of XML code, which shows that the book *Harry Potter* is written by *J.*

[1]`https://en.wikipedia.org/wiki/Semi-structured_data`

```
<book>
<name>Harry Potter</name>
<author>J. K. Rowling</author>
</book>
```

Figure 1.1: An example of XML.

K. Rowling. This meaning is encoded via the metadata in XML. Researchers have proposed a series of tools to manage semi-structured data [1, 45, 26, 28, 11, 17, 55].

Unstructured data often refers to natural language text. A large number of research projects were conducted to extract the structured format of data [52, 18, 15, 51, 113, 79, 20, 3, 101, 10]. Subsequently, tools built on relational data can manage the unstructured data.

This dissertation focuses on a new type of nearly relational data that we call *para-relational data*. Our goal is to demonstrate that it is possible to convert para-relational data into a relational format. Para-relational data refers to a type of data that shares the important qualities of relational data but does not present itself in a relational format. Our definition of the para-relational data has two important qualities:

- **Parseable Relational Units** — Para-relational data can be converted into the relational form, but it does not present itself directly as a relational form. That being said, each cell in the target relational table is positioned as an individual unit, encoded using the para-relational data structure such as a hierarchical structure or other standard structures.

  This quality of the para-relational data is similar to semi-structured data, as it uses tags, markers, or standard structures to encode the semantic units. This quality is different from unstructured data because natural language text does not use embedded structures to encode relational data. We often must parse the grammar of the text to obtain each individual unit or cell of the target relational table from the unstructured data.

- **Hidden Semantic Meanings** — Para-relational data is often encoded in a structure with implicit, embedded semantic meanings. As a result, it is necessary to understand the underlying semantic meanings in order to correctly parse the implicit relational data.

  This quality is similar to the unstructured data, as we must recognize the semantics of the data to correctly identify the hidden relational data. However, the semantic meanings of semi-structured data are explicit. Semi-structured data (such as XML) uses the associated metadata to explicitly encode semantic meanings. For example, Figure 1.1 shows that we can easily tell the name of the book is *Harry Potter* and the author is *J. K. Rowling*, according to the tags.

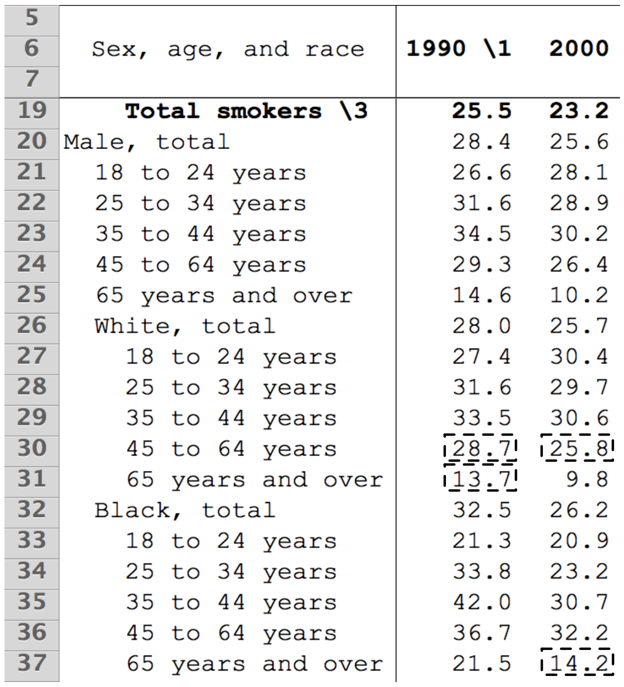
This dissertation concentrates on three typical types of para-relational data: Spreadsheets, dictionaries in webpages, and diagrams. We give three real-life examples, showing each type of para-relational data.

1.1.1 Spreadsheets

Spreadsheets are a critical data management tool that is diverse and widely used: Microsoft estimates the number of worldwide Excel users at more than 400 million, and Forrester Research estimates 50 to 80% of businesses use spreadsheets.[2] Moreover, there is a large amount of data on the web that is, practically speaking, only available via spreadsheets. For example, the United States government published a compilation of thousands of spreadsheets about economic development, transportation, public health, and other important social topics; a spreadsheet was the only data format used.

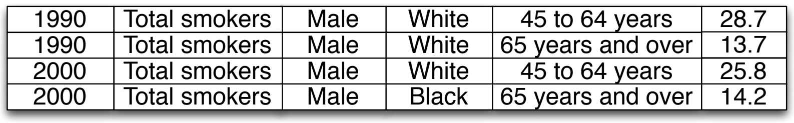
Spreadsheets are a type of para-relational data:

[2]http://www.cutimes.com/2013/07/31/rethinking-spreadsheets-and-performance-management



5			
6	Sex, age, and race	**1990 \1**	**2000**
7			
19	**Total smokers \3**	**25.5**	**23.2**
20	Male, total	28.4	25.6
21	18 to 24 years	26.6	28.1
22	25 to 34 years	31.6	28.9
23	35 to 44 years	34.5	30.2
24	45 to 64 years	29.3	26.4
25	65 years and over	14.6	10.2
26	White, total	28.0	25.7
27	18 to 24 years	27.4	30.4
28	25 to 34 years	31.6	29.7
29	35 to 44 years	33.5	30.6
30	45 to 64 years	28.7	25.8
31	65 years and over	13.7	9.8
32	Black, total	32.5	26.2
33	18 to 24 years	21.3	20.9
34	25 to 34 years	33.8	23.2
35	35 to 44 years	42.0	30.7
36	45 to 64 years	36.7	32.2
37	65 years and over	21.5	14.2

(a) A spreadsheet example



1990	Total smokers	Male	White	45 to 64 years	28.7
1990	Total smokers	Male	White	65 years and over	13.7
2000	Total smokers	Male	White	45 to 64 years	25.8
2000	Total smokers	Male	Black	65 years and over	14.2

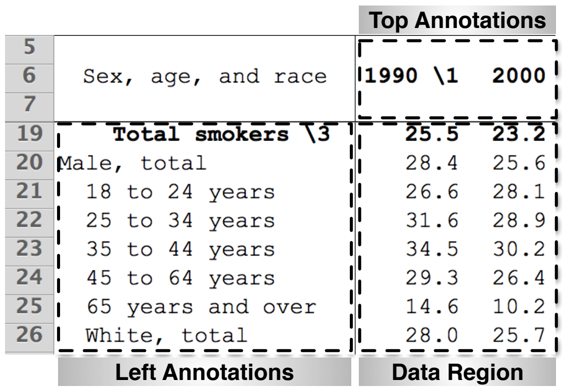
(b) Relational table for the four values in the spreadsheet

Figure 1.2: A spreadsheet about smoking rates, from the *Statistical Abstract of the United States*.

- **Parseable Relational Units** — Spreadsheets often contain relational data but do not explicitly present the relational formatted data. For example, Figure 1.2(a) shows a portion of a spreadsheet downloaded from the Statistical Abstract of the United States.[3] A user can easily tell that the *data* value 28.7 is described by the *annotations* 1990, Male, White, and 45 to 64 years. We call this implicit relationship between annotations and data a ***mapping***. By repeatedly finding such mappings, we can reconstruct the *relational table* seen in Figure 1.2(b). However, this relational table is implicit and the spreadsheet itself does not explicitly show the relational table.

- **Hidden Semantic Meanings** — There are many implicit structures often

[3]`http://www.census.gov/compendia/statab/2012/tables/12s0204.xls`



5	Sex, age, and race	1990 \1	2000
6			
7			
19	**Total smokers \3**	**25.5**	**23.2**
20	Male, total	28.4	25.6
21	18 to 24 years	26.6	28.1
22	25 to 34 years	31.6	28.9
23	35 to 44 years	34.5	30.2
24	45 to 64 years	29.3	26.4
25	65 years and over	14.6	10.2
26	White, total	28.0	25.7

Figure 1.3: The three semantic components of a data frame spreadsheet.

hidden in spreadsheets. First, a *data frame* is a very common data model in spreadsheets. A data frame often consists of three semantic regions: The top and left annotation regions and the data region. For example, Figure 1.3 shows the three components that make up a data frame: Two rectangular annotation regions (left and top) and a single rectangular data region.

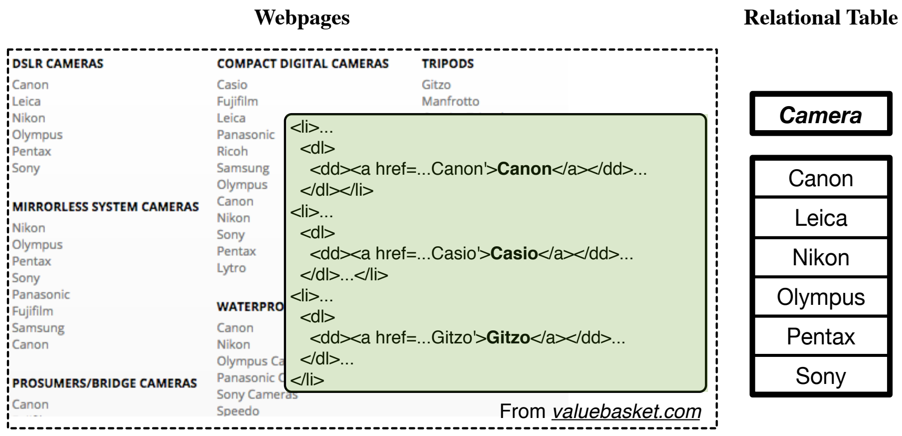
Second, the *hierarchical structure* is also common in spreadsheets. In Figure 1.2 (a), the *data* value 28.7 is implicitly annotated by the *annotations* 1990, Male, White, and 45 to 64 years in a hierarchical fashion. We call this implicit mapping relationship the *hierarchical structure* in spreadsheets.

1.1.2 Dictionaries from Webpages

Webpages often contain a list of instances belonging to the same conceptual class that we call a *dictionary* (also known as gazetteers). For example, a camera brand dictionary contains “Canon”, “Nikon” and so on.

Dictionaries in webpages are para-relational data:

- **Parseable Relational Units** — Dictionaries in webpages can be presented in a relational format but the webpages do not explicitly show it. For example, the left side of Figure 1.4 shows an example webpage with extractable dictionary



```
<li>...
  <dl>
    <dd><a href=...Canon'>Canon</a></dd>...
  </dl></li>
<li>...
  <dl>
    <dd><a href=...Casio'>Casio</a></dd>...
  </dl>...</li>
<li>...
  <dl>
    <dd><a href=...Gitzo'>Gitzo</a></dd>...
  </dl>...
</li>
```

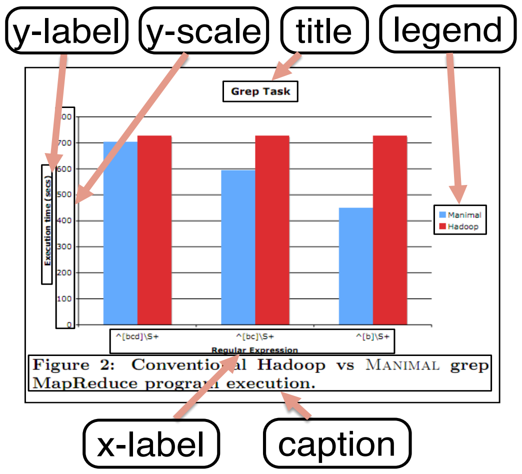
Figure 1.4: A list of camera manufacturers from a webpage (with corresponding HTML source code) can be represented in the relational format.

items for camera manufacturers. However the webpage itself does not explicitly present the relational table shown on the right side of Figure 1.4.

- **Hidden Semantic Meanings** — The webpage may use a variety of encoding formatting styles to present dictionaries, such as an HTML list or table, or even simple hyperlinks. Simply extracting the items according to the encoding format is not sufficient to extract the dictionary item. Figure 1.4 shows that this webpage uses the HTML list to format the dictionary items, but the list contains both in-set items (camera manufacturers such as "Canon" and "Nikon") and out-of-set items (tripod manufacturers such as "Gitzo"). To obtain the relational form of this list, it is important to understand the underlying semantics to distinguish in-set from out-of-set items.

1.1.3 Diagrams

Data-driven diagrams (or statistical graphics) are an important method for communicating complex information. Diagrams, a stylized mixture of graphics and text, offer succinct quantitative summaries of data that motivates the overall document's content. For many technical documents, the diagrams may be readers' only access



(a) A diagram example

x-label	*y-label*	*x-scale*	*y-scale*	*title*	*caption*	*legend*
Regular..	Execution…	^[bcd]\s+…	0 100…	Grep…	Figure 2…	Manimal…

(b) Relational table for the diagram metadata

x-label	*y-label*	*value*	*others*
^[bcd]\S+	Execution..	700	Manimal
^[bcd]\S+	Execution..	721	Hadoop
^[bc]\S+	Execution..	596	Manimal

(c) Relational table for the diagram data points

Figure 1.5: A diagram contains several characteristic regions of text: Title, x-label, y-label, legend, and so on. The diagram can be represented in a relational format.

to the raw data underlying the documents' conclusions. Especially for quantitative disciplines such as finance, public policy, and the sciences, certain diagrams could be more valuable than the surrounding text.

The diagrams are para-relational data:

- **Parseable Relational Units** — Diagrams can be presented in a relational format but the they do not explicitly show it. For example, the two-dimensional diagram as shown in Figure 1.4 (a) often consists of a set of metadata information, including y-label, y-scale, title, legend, x-label, and x-scale. It is possible to represent this diagram with two relational tables, as Figures 1.4 (b) and (c) show: One relational table for the diagram metadata and another one for the data points in the diagram. Unfortunately, the diagram itself does not explicitly present the two relational tables.

- **Hidden Semantic Meanings** — A diagram often contains a set of semantic fields, as shown in Figure 1.4 (a): A diagram contains the fields that can be used to generate a unique diagram image, including x-label, y-label, y-scale, title, and so on. Without understanding each of the key fields in a diagram, it is impossible to automatically construct the resulting relational table shown in Figures 1.4 (b) and (c).

1.2 Design Criteria and Challenges

The goal of this dissertation is to demonstrate that we can convert various types of para-relational data into a relational form. Doing so makes it possible to use many interesting downstream applications on the para-relational data.

- **Data Analysis Applications** — If we can convert para-relational data into a relational form, we can leverage the operations designed on relational data to analyze the para-relational data. For example, we can use the SQL query language to select, project, or aggregate to obtain important information in a huge amount of para-relational data.

- **Data Visualization Applications** — Most current data visualization tools (such as Tableau) can only support relational data. If para-relational data can be converted into a relational form, we can simply use the existing visualization tools to generate diagrams from para-relational data without writing customized code.

- **Data Integration Applications** — It would be easy to integrate the para-relational data with various other relational sources.

  For example, consider a policy expert Fred who wants to see if the strength of the connection between smoking and lung cancer is consistent across all U.S. states. The user does not have the relevant data at hand, so looks for it on the web.

  In one sense, the user is fortunate. Different branches of the government have collected the data relevant to his task and made it available online, likely via two separate downloadable spreadsheets: One for smoking statistics and one for lung cancer statistics. Unfortunately, finding such data via current search engines is quite tedious. In our case, the user would need to issue a text query, and review all the returned documents before finding the relevant spreadsheets. Moreover, because spreadsheets do not have explicit relational schema, the user cannot benefit from society's huge investment in data integration tools that work on relational databases. Instead, the user likely must write custom code to combine the two spreadsheets, which is a tedious process.

  If it is possible to convert spreadsheets into a relational format, users can easily use an existing integration tool to integrate the two spreadsheets.

To convert para-relational data into a relational form, we have the following three design criteria:

- **Machine Learning Algorithms** — Each type of para-relational data would be encoded using a customized semantic structure. A rule-based approach would require the user to write a massive number of brittle rules. In practice, machine learning methods are the best way to handle the diversity of expression in para-relational data. As a result, we must employ machine learning algorithms to learn the extraction algorithms.

- **High-quality Results** — We require high-quality (*i.e.*, high-precision and high-recall) or even perfect extraction results, because the extracted relational form of data would be fed into the downstream applications for further data processing. Even one mistake from the extraction procedure will cascade and hinder downstream applications.

  For example, a single mistake in hierarchy detection task can yield an extracted relation that has many incorrect tuples, so ruins our data integration application. As Figure 1.2 shows, if we fails to recognize that Male annotates White, the system would generate incorrect relational tuples for all the data values from rows 26 to 31.

  Therefore, we require high-quality and even perfect extraction algorithms.

- **Little User Assistance** — Finally, user labeling tasks are expensive for many tasks. For example, it often requires specialized skills or knowledge to label the training data. Therefore, when building the extraction algorithms, it is highly likely that we will not receive much user assistance in providing training data or feedback to obtain perfect extraction results.

Considering the above design criteria, we face the following two challenges to convert the para-relational data into the relational form:

Implicit Semantics Extraction — Para-relational data is often encoded in a structure with implicit embedded semantic meanings, but does not explicitly present its structure. Different types of para-relational data may contain different semantic structures. For example, Figure 1.2 shows the spreadsheet hierarchical structure, and Figure 1.3 shows the spreadsheet data frame structure. Figure 1.4 shows a webpage with dictionary items of *camera manufacturers* that are mixed with entities belonging to *tripod manufacturers*. Figure 1.5 shows that diagrams often contain a set of semantic fields, including x-label, y-label, y-scale, title, and so on. To construct

System	**Para-relational Data**
Senbazuru	spreadsheets
Anthias	spreadsheets
Lyretail	dictionaries
DiagramFlyer	diagrams

Table 1.1: Each of the four systems addresses a specific type of para-relational data.

the relational form of the data, it is critical to correctly understand the underlying semantics of the hidden semantic structures. However, it is not straightforward how to automatically obtain these structures.

Little User Assistance — We employ a variety of machine learning techniques to obtain the implicit hidden semantics for different types of para-relational data. However, there are two key problems: First, many machine learning algorithms require a sufficient amount of training data for the training procedure to obtain a satisfying prediction performance. But it is always hard and costly to obtain a lot of labeled data. Second, it is almost impossible for any machine learning algorithm to obtain perfect prediction results. However, in many cases, especially when there are downstream applications, the quality of the prediction results is critical. The user always must browse the results one after another for validation. That said, it requires a huge amount of user effort to obtain perfect extraction results.

In this dissertation, we explore a variety of machine learning techniques to extract the inherent structure or semantic meaning of the para-relational data (*i.e.*, spreadsheets, dictionaries in webpages, and diagrams), therefore making it possible to automatically, or with little user effort, convert the data to a relational format. We now discuss our approaches in detail.

1.3 Approaches and Systems

In this dissertation, we aim to convert para-relational data into a high-quality relational form with little user assistance. We developed four standalone systems that contribute to the conversion from para-relational data to the relational format. Each system addresses a specific type of para-relational data, as shown in Table 1.1. *Senbazuru* is a prototype spreadsheet database-management system that extracts relational information from a large number of spreadsheets. *Anthias* suggests an extension of the Senbazuru system to convert a broader range of spreadsheets into a relational format. *Lyretail* is an extraction system that detects long-tail dictionary entities on webpages. *DiagramFlyer* is a web-based search system that obtains a large number of diagrams automatically extracted from web-crawled PDFs. These four systems demonstrate that converting para-relational data into the relational format is possible today, and also suggest directions for future systems.

1.3.1 Senbazuru: Extraction on Spreadsheet Structures

We developed Senbazuru, a prototype spreadsheet database management system (SSDBMS). Senbazuru extracts the structural information from a large number of spreadsheets, making it possible to convert spreadsheets into a relational form with little user effort.

Our technical contributions mainly lie in two parts:

Implicit Semantics Extraction — We have identified two typical implicit semantic structures of spreadsheets: The data frame model and the hierarchical structure.

The data frame is a very common data model in spreadsheets. A data frame often consists of three semantic regions, as shown in Figure 1.3. We propose a conditional random field based approach to identify the semantic regions in a spreadsheet by assigning one of four labels (*i.e.*, header, data, footnote or title) to each row of a spreadsheet. In this way, we could automatically recognize the the three semantic regions of a data frame in a spreadsheet.

Hierarchical structure is an important semantic structure in spreadsheets. We propose a new two-phase *semiautomatic* approach based on an undirected graphical model to accurately extract spreadsheet annotation-to-data mappings. It receives spreadsheets as input and computes a hierarchical structure with user interaction.

Little User Assistance — To obtain perfect spreadsheet hierarchical structure, we propose a new two-phase *semiautomatic* approach based on an undirected graphical model to accurately extract the hierarchical structure with little user effort. First, the *automatic extractor* receives spreadsheets as input and computes a mapping without user interaction. Based on an undirected graphical model, it exploits single-spreadsheet graphical style hints (such as the font and typographic alignment) and correlated extraction decisions in one spreadsheet or across spreadsheets. Second, our system offers an *interactive repair* phase, in which a user repeatedly reviews and corrects the automatic extractor's output until no errors remain. Our interactive repair is more than simply asking a user to fix every single extraction error. It exploits the correlations among different extraction decisions to make more effective use of each user repair operation. A user's single repair can be silently and probabilistically applied to multiple possible errors, allowing us to *amortize* the user's effort over many likely extractor mistakes.

Figure 1.6 shows an example of the user interface for applying repairs. The left side of the diagram indicates the initial hierarchy obtained by the automatic extractor for Figure 1.2. The dashed arrow shows that a user performs a repair by clicking and dragging White so that it becomes a child of Male, indicating that Male annotates White. This one repair operation triggers multiple error fixes, including setting Male to also annotate Black.

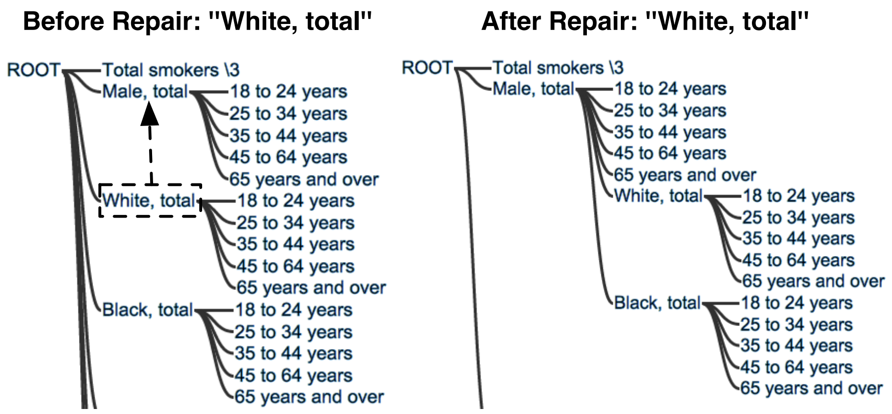


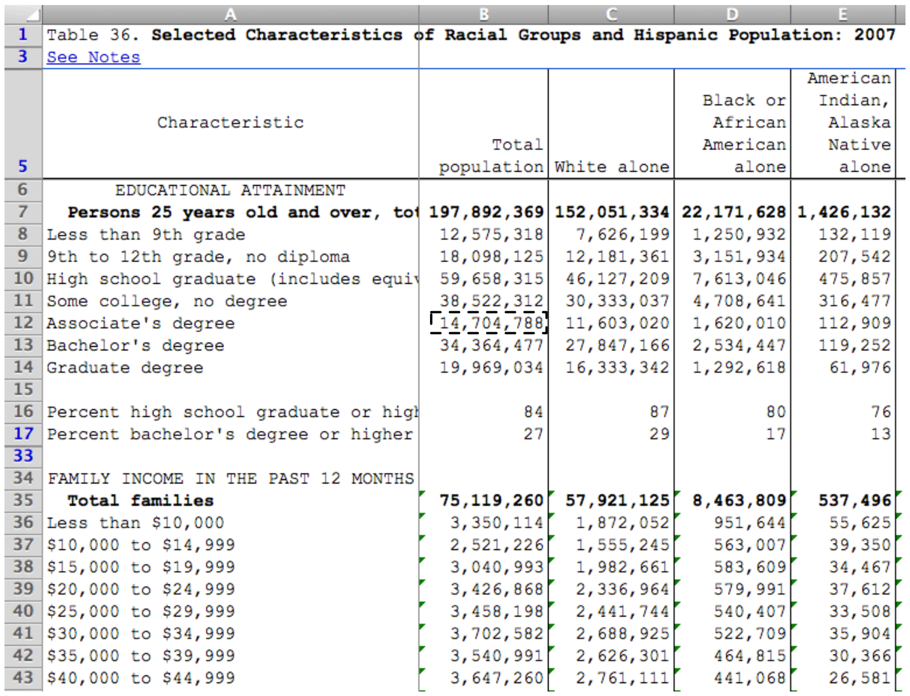
Figure 1.6: Our user interface for repairing mappings.

1.3.2 Anthias: Extraction on Spreadsheet Properties

We propose the Anthias system, which is an extension of Senbazuru, to convert a broader range of spreadsheets (in addition to the data frame spreadsheet mentioned in Section 1.3.1) into a relational format. Anthias enhances Senbazuru by considering a variety of spreadsheet properties. We use the *spreadsheet properties* to refer to a series of transformation programs that contribute toward the spreadsheet-to- relational table transformation framework. For example, Figure 1.7 shows a portion of a spreadsheet downloaded from the Statistical Abstract of the United States.[4] The spreadsheet shows a *sheet table* that consists of the header region (row 5) and data region (rows 6-43), but it is not a relational table. Figure 1.8 shows the ideal relational tables that are equivalent to this sheet table, but generating them requires a series of transformation programs:

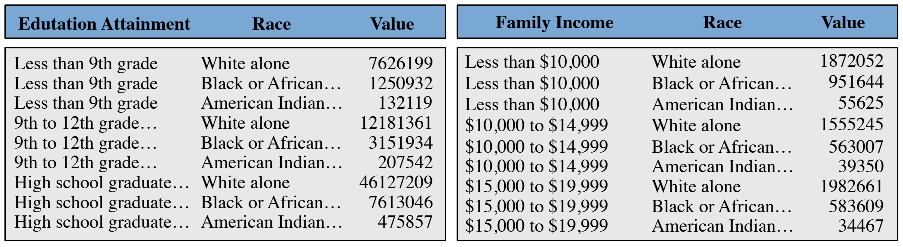
- Transform *aggregation rows* — Data values in rows 16-17 are aggregated values defined on rows 7-14. We *remove* the aggregated values in the resulting table.
- Transform *aggregation columns* — Data values in column B are aggregated values defined on columns C-E. We *remove* the aggregated values in the resulting table.

[4]`http://www.census.gov/compendia/statab/2010/tables/10s0036.xls`



	A	B	C	D	E
1	Table 36. **Selected Characteristics of Racial Groups and Hispanic Population: 2007**				
3	See Notes				
5	Characteristic	Total population	White alone	Black or African American alone	American Indian, Alaska Native alone
6	EDUCATIONAL ATTAINMENT				
7	**Persons 25 years old and over, tot**	**197,892,369**	**152,051,334**	**22,171,628**	**1,426,132**
8	Less than 9th grade	12,575,318	7,626,199	1,250,932	132,119
9	9th to 12th grade, no diploma	18,098,125	12,181,361	3,151,934	207,542
10	High school graduate (includes equiv	59,658,315	46,127,209	7,613,046	475,857
11	Some college, no degree	38,522,312	30,333,037	4,708,641	316,477
12	Associate's degree	14,704,788	11,603,020	1,620,010	112,909
13	Bachelor's degree	34,364,477	27,847,166	2,534,447	119,252
14	Graduate degree	19,969,034	16,333,342	1,292,618	61,976
15					
16	Percent high school graduate or high	84	87	80	76
17	Percent bachelor's degree or higher	27	29	17	13
33					
34	FAMILY INCOME IN THE PAST 12 MONTHS				
35	**Total families**	**75,119,260**	**57,921,125**	**8,463,809**	**537,496**
36	Less than $10,000	3,350,114	1,872,052	951,644	55,625
37	$10,000 to $14,999	2,521,226	1,555,245	563,007	39,350
38	$15,000 to $19,999	3,040,993	1,982,661	583,609	34,467
39	$20,000 to $24,999	3,426,868	2,336,964	579,991	37,612
40	$25,000 to $29,999	3,458,198	2,441,744	540,407	33,508
41	$30,000 to $34,999	3,702,582	2,688,925	522,709	35,904
42	$35,000 to $39,999	3,540,991	2,626,301	464,815	30,366
43	$40,000 to $44,999	3,647,260	2,761,111	441,068	26,581

Figure 1.7: A spreadsheet about population statistics, from the Statistical Abstract of the United States.



Edutation Attainment	Race	Value
Less than 9th grade	White alone	7626199
Less than 9th grade	Black or African…	1250932
Less than 9th grade	American Indian…	132119
9th to 12th grade…	White alone	12181361
9th to 12th grade…	Black or African…	3151934
9th to 12th grade…	American Indian…	207542
High school graduate…	White alone	46127209
High school graduate…	Black or African…	7613046
High school graduate…	American Indian…	475857

Family Income	Race	Value
Less than $10,000	White alone	1872052
Less than $10,000	Black or African…	951644
Less than $10,000	American Indian…	55625
$10,000 to $14,999	White alone	1555245
$10,000 to $14,999	Black or African…	563007
$10,000 to $14,999	American Indian…	39350
$15,000 to $19,999	White alone	1982661
$15,000 to $19,999	Black or African…	583609
$15,000 to $19,999	American Indian…	34467

Figure 1.8: The ideal relational tables for the spreadsheet example shown in Figure 1.7.

- Transform *crosstab* — The headers of columns C-E (*i.e.*, “White Alone”, “Black or…” and so on) form the horizontal dimension *Race*. We *pivot* this horizontal dimension into a new column *Race* in the resulting relational table.

- Transform *split tables* — Rows 6-17 show “Education Attainment” and rows 34-43 show “Family Income”. We *split* the two parts as two tables.

Each of the transformation programs above describes a piece of the process on a specific characteristic of a sheet table that yields a result closer to a relational table. We use a spreadsheet property (such as aggregation rows, aggregation columns, crosstab, and split tables) to represent such a transformation program that contributes to the spreadsheet-to-relational-table transformation.

Our technical contributions mainly lie in two parts:

Implicit Semantics Extraction — Spreadsheet properties are a critical concept to to represent the spreadsheet-to-relational-table transformation. We study the task of spreadsheet property detection, which decides if a spreadsheet contains a specific spreadsheet property (*e.g.*, whether a spreadsheet contains "aggregation rows"). To the best of our knowledge, we are the first to propose the spreadsheet property detection problem, which is the first step toward building the spreadsheet-to-relational-table pipeline.

Little User Assistance — Our main technical contribution is a novel rule-assisted active-learning framework to construct high-quality spreadsheet property detectors with little user labeling effort. We developed a hybrid approach that integrates crude user-provided rules with an active learning approach to reduce user labeling effort. In addition to the labeled instance suggested by the active learning approach, we bring in crude rules from users to generate additional labeled data for the property detectors. We produce labeled instances with the agreed decision from both the current trained classifier and the user-provided rules. This bagging-like technique makes it possible for our framework to tolerate *bad* rules. Our hope is that this hybrid approach can generate additional high-quality labeled data especially in the initial stage to quickly warm up the classifiers.

1.3.3 Lyretail: Extraction on Dictionaries from Webpages

We develop the Lyretail extraction system. Using only a few user-given seeds, Lyretail automatically produces a high-quality page-specific dictionary for each input webpage. The page-specific dictionaries (PSDs) from many webpages can be further aggregated as a high-quality comprehensive dictionary to answer the user's request.

Our contribution mainly lies in two parts:

Implicit Semantics Extraction — We develop the Lyretail extraction system, which builds a unique extraction model that automatically produces a high-quality page-specific dictionary for each input webpage given a few seed examples. By aggregating the PSDs of many webpages retrieved from the web, Lyretail can also compute a high-quality comprehensive dictionary as its answer to the user's request.

Little User Assistance — We are able to obtain high-quality extraction results while reducing user assistance as much as possible in two ways: First, Lyretail automatically generates training data for building high-quality page-specific extractors using a distant supervision based algorithm. Second, we propose a co-training framework that incorporates sequential features to build high-quality page-specific extractors. We leverage the output from the extractor built on web lists to train a semi-supervised CRF as the resulting page-specific extractor. As a result, we can build high-quality page-specific extractors on each webpage that often can distinguish infrequently-observed true extractions from incorrect ones. These results are then aggregated into high-quality long-tail dictionaries.

1.3.4 DiagramFlyer: Extraction on Diagrams from PDFs

We present DiagramFlyer, a working search engine that provides search services for a corpus of 319k diagrams extracted from a web crawl of PDFs online.

Our contributions are mainly lie in **Implicit Semantics Extraction**. For each data-centric diagram discovered in a large number of documents, the DiagramFlyer extractor recovers as much of the underlying diagram production process as possible (such as captions and x-axis and y-axis labels). We also provide a software architecture and set of algorithms for implementing DiagramFlyer's query tools that perform diagram relevance ranking, similar item finding via lexicons, and snippet generation.

Regarding **Little User Assistance**, we attempt to construct high-quality diagram metadata extractors by asking for a sufficient amount of training data from the users.

1.4 Scientific Contributions

In this dissertation, we aim to convert para-relational data into a high-quality relational form with little user assistance. Our scientific contributions mainly lie in the following three areas (we will discuss the related work in the three areas in detail in Chapter 2):

- **Information extraction** — Previous research on information extraction can be divided into either domain-dependent or domain-independent extraction. Domain-dependent extraction often requires a lot of user effort but can obtain high-quality extraction results, while domain-independent extraction only requires very little user assistance but may sacrifice the extraction performance.

  This dissertation developed information extraction techniques on para-relational data to obtain high-quality or even perfect extraction results while reducing user assistance as much as possible.

- **Machine learning** — It is very hard for any machine learning algorithms to obtain the fully accurate prediction result. In addition, previous classification and joint-inference algorithms often require a sufficient amount of user effort on training data, while active learning approaches can often save a sufficient amount of training data but suffer from the cold-start problem.

  In this dissertation, in addition to applying a variety of existing machine learning algorithms in the information extraction applications, we make two attempts to innovate around machine learning algorithms: We attempt to obtain fully accurate extraction results by interacting with users while reducing the required user effort as much as possible; and we attempt to further reduce the user effort required by active learning by incorporating simple, crude user-provided rules, thereby alleviating the cold-start problem of active learning techniques.

- **Data management** — Previous data management researches mainly focused on relational or semi-structured data sources, and the researches have dealt with many application problems such as query and integration.

  This dissertation focuses on a new type of data that is different from previous work on data management which we call para-relational data. Our contribution is mainly from an application perspective. We convert para-relational data into a high-quality relational form based on our novel techniques on information extraction and machine learning. This makes it possible to use existing tools on relational data to manage the para-relational data.

1.5 Outline of the Dissertation

The next chapter provides a brief background description of three areas of research relevant to this dissertation: Information extraction, machine learning, and data management. These are all huge areas of work, so we focus on areas of the literature where they have intersected to some degree. Chapters 3-6 cover the four projects on para-relational data extraction in detail: Senbazuru, Anthias, Lyretail, and DiagramFlyer. We conclude and discuss future work in Chapter 7. All together, these chapters present a coherent set of research contributions and a vision for future work on para-relational data extraction.

I was the lead researcher for all the projects presented in this dissertation, and almost all the projects have appeared, or will appear, in various venues. We presented the Senbazuru project in Chapter 3 as a system demonstration in VLDB 2013 [34]. We showed the system pipeline of Senbazuru in SSW 2013 [31] and presented the hierarchy extraction algorithm in KDD 2014 [32]. The Anthias project in Chapter

4 is under submission for the WWW conference in 2016 [36]. The Lyretail project in Chapter 5 will appear in the WSDM conference in 2016 [35]. We presented the DiagramFlyer project in Chapter 6 in Frontiers of Engineering in 2012 [30] and also as a system demonstration at the 2015 WWW conference [33].

CHAPTER II

Research Background

There are three large areas of research that are relevant to para-relational data extraction. We discuss the area of information extraction in Section 2.1, machine learning in Section 2.2, and relational data management in Section 2.3.

2.1 Information Extraction

Information extraction often refers to the automatic extraction of structured information, such as entities, relationships between entities, and attributes describing entities from unstructured data sources [95].

The early work in information extraction (IE) was inspired by the Message Understanding Conferences (MUCs), initiated by DARPA (Defense Advanced Research Projects Agency) in the early 1990s. At that time, MUC was focusing on Information Extraction (IE) tasks in which structured information of company activities and defense-related activities was extracted from the unstructured text. For the last two decades, the IE task has attracted much research. In general IE research can be divided into two parts: *Domain-dependent extraction* is sensitive to topic-specific data, rules, or schemas and *domain-independent extraction* often avoids extraction rules or training data that is tailored to a specific topic.

We discuss the two areas of work in detail in the next two sections.

2.1.1 Domain-dependent Extraction

Domain-dependent extraction often refers to the series of IE tasks on topic-specific data, rules, or schemas. One of the most well-known domain-dependent extraction tasks is the Named Entity Recognition and Classification (NERC) task. The goal of the NERC task is to recognize information units such as names, including person, organization and location names, and numeric expressions including time, date, money, and percentage expressions [82]. One of the first research papers in the field was presented by Lisa Rau in 1991. Rau [89] proposed an algorithm to extract and recognize company names that relies on heuristics and the handcrafted rule.

There are a variety of domain-dependent extraction tasks. Fleischman [52] developed a method for extracting the subcategorization of location names, including city and state. The extraction of *person* is quite common and used at least once in an original way Bodenreider and Zweigenbaum [18] proposed several criteria to identify proper names in biomedical terminologies. Bick [15] developed a named entity recognizer that targets six primary name types (*human, organization, place, event, title/semantic product*, and *brand/object*). In addition to the IE tasks on common types, there is research on marginal types for specific needs, such as *scientist* [51], and *email address* and *phone number* [113, 79].

To summarize, the domain-dependent extraction can often produce very high-quality extraction results but requires a lot of domain knowledge or heuristics to process the extraction.

2.1.2 Domain-independent Extraction

Domain-independent extraction does not target for a specific tasks and often does not require extraction rules or training data tailored to a specific topic. This area of research has attached much attention especially in the last two decades.

The first step in automating IE moved from knowledge-based IE systems to train-able systems. AutoSlog-TS [91], in the 1990s, attempted to automatically generate extraction patterns using annotated text and a set of heuristic rules. DIPRE [20] and Snowball [3] further reduced manual labor needed for relation-specific text extraction by only requiring a small set of valid seed tuples for the target relation provided by the users to start the extraction process. YAGO [101] used Wikipedia information to produce a large number of ontology objects consisting of is-a and 13 other fixed binary relations while retaining high quality. For example, the object *Paris* is in the *FamilyNameOf* relation along with *Priscilla Paris*, and in the *Means* relation along with *Paris, France.*

Later projects focused on conducting the extraction process on a large scale of webpages. OpenIE [10] employed the distant-supervision method to obtain millions of relational tuples from a large corpus by making a single data-driven pass without requiring any user input. The WebTable project [22] also conducted a web-scale IE system to obtain a huge number of web HTML tables. This huge amount of resources was further used to explore searching tables, attribute synonym finding and many other tasks.

To summarize, the domain-independent extraction often requires very little user effort but can produce satisfactory extraction results at a large scale. The domain-independent extraction saves user effort but may sacrifice performance.

2.2 Machine Learning

Our extraction tasks use a variety of machine learning algorithms. Machine learning algorithms are one of the oldest computer applications. There were a large number of important algorithmic and theoretic developments over the past century [98]. We mainly focus on the classification tasks and briefly introduce some basic and joint-inference classification models, and the active learning technique in the next three sections.

2.2.1 Basic Classification Algorithms

The classification tasks (binomial or multinomial) are one of the most frequently studied problems in machine learning. The classification problem studies the task of classifying the elements of a given set into two or more groups based on the classification rules. The classification rule can be manually designed but is often learned through training data.

The most prominent classification algorithms include logistic regression, naive bayes, support vector machines (SVM), decision trees, and random forests. We now briefly introduce logistic regression in detail.

Logistic regression — Logistic regression is used to predict the odds of being a case based on the values of the independent variables (predictors).

Consider the binary classification with $y = 0$ or 1. Each example is represented by a feature vector x. Logistic regression defines the probability distribution on the feature vector x as follows:

$$p(y|x) = \frac{1}{1 + \exp(-y\theta^T x)} \tag{2.1}$$

Logistic regression can also be easily generalized to multiple classes. Let there be K classes. Each class has its own parameter θ_k. The probability distribution is defined via the softmax function:

$$p(y = k|x) = \frac{\exp(\theta_k^T x)}{\sum_{i=1}^{K} \exp(\theta_k^T x)} \tag{2.2}$$

To summarize, almost all these simple classification algorithms assume that the entities are independent of each other, and require a sufficient amount of training data to estimate the unknown parameters defined in the models.

2.2.2 Joint Inference Algorithms

In recent decades, joint-inference models are especially popular in the natural language processing area by considering the correlation among entities [80]. The joint-inference models achieve better performance on well-defined subproblems such as part-of-speech tagging, phrase chunking, syntactic parsing, named-entity recognition, and semantic-role labeling. The popular joint inference models include conditional random fields [75, 102] and other types of graphical models [71].

Conditional Random Fields (CRFs) — Conditional random fields (CRFs) are a probabilistic framework for labeling and segmenting structured data, such as sequences, trees and lattices. It is a discriminative model and defines a conditional probability distribution over label sequences given a particular observation sequence.

Consider a linear-chain conditional random field [75, 102]. Let $\mathbf{y}, \mathbf{x}$ be random vectors, $\theta = \theta_k$ be the parameter vectors and $f_k(y, y', \mathbf{x_t})$ be a set of feature functions. Then a linear-chain conditional random field defines the probability distribution as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\Pi_{t=1}^{T} \exp\{\sum_{k=1}^{K} \theta_k f_k(y, y', \mathbf{x_t})\} \tag{2.3}$$

where Z is a normalization factor.

In addition to the linear-chain CRFs, there are other forms of CRFs that can be easily generalized according to the probability distribution shown in Equation 2.3, such as the skip-chain CRFs and general CRFs.

Graphical model — A graphical model G [71] describes a joint distribution over a set of n random variables $\mathbf{x} = \{x_1, ..., x_n\}$, where each variable x_i takes a label l_i from a set of labels L. The model captures properties of each variable and dependencies among variables in the graph by defining *potential functions* on cliques of correlated variables. The probability distribution is defined as:

$$P(x_1, ..., x_n) = \frac{1}{Z} \exp(\sum_{C \subseteq cliques(G)} \theta(C, \mathbf{x})) \tag{2.4}$$

A common method to define the potentials is as a dot function between the *weight* parameters and a *feature* vector [85]. A node potential captures the features that correspond to a single variable. The node potential is usually defined on a variable x_i as $\theta(x_i) = \mathbf{w_1}^T \mathbf{f}(x_i, l_i)$, where $\mathbf{f}(x_i, l_i)$ is a feature vector and $\mathbf{w_1}$ is the associated weight parameters. Similarly, the edge potential is usually defined on pairwise variables x_i and x_j to describe their correlation as $\theta(x_i, x_j) = \mathbf{w_2}^T \mathbf{f}(x_i, l_i, x_j, l_j)$. Users generally provide domain knowledge via the feature vectors $\mathbf{f}$, while the parameters $\mathbf{w} = \{\mathbf{w_1}, \mathbf{w_2}\}$ are *trained* from labeled data. In the *training* stage, the feature vector is derived from a set of labeled data to obtain the optimal value for the *weight* parameters $\mathbf{w}$. In the *inference* stage, the optimal labeling can be obtained by finding the maximum joint probability. As our model is conditionally trained, it belongs to the class of general graph conditional random fields [75].

To summarize, these algorithms assume the entities are correlated in some predefined patterns. However, similar to the basic classification algorithms, all the algorithms require a sufficient amount of training data to estimate the unknown parameters defined in the models.

2.2.3 Active Learning

Active learning is a subfield of machine learning. This area of research attracted much attention in the last two decades. The key idea of active learning is that we can save the amount of data needed to be labeled if the learning algorithm is allowed to choose the data in a smart way [97]. An active learner may ask queries in the form of unlabeled instances to be labeled by an oracle (such as a human annotator).

There are two common active learning [97] approaches. The first is *uncertainty sampling*, and it is the most popular active learning strategy. An uncertainty sampling strategy often chooses to label instances closest to the decision boundary and refines the decision boundaries by heavily exploiting the current knowledge space. The uncertainty sampling approach in [94] simply selects the instance with the predicted probability closest to 0.5. The second is a variation of the Query by Committee (QBC) [103] technique. It is a selection framework that takes into account the disagreement of multiple committee classifiers. The QBC strategy is more complicated than uncertainty sampling, as it requires careful design of approaches to create a set of models (*i.e.*, committee members) and a metric to measure disagreement among the committee members.

Active learning methods can save a sufficient amount of training data but often suffer from the cold-start problem [117]. In the beginning stages, the classifier lacks training data to approach the ideal decision boundary and suggest effective instances to label.

2.3 Data Management

There has been a large variety of work on data management, but most of it focuses on managing relational data or semi-structured data sources. Researchers also studied the problem of model management, which is a generic approach to solving problems of data programmability in which precisely engineered mappings are required [13]. However the research on model management mainly focused on the mappings between relational data and semi-structured data [12, 13, 67]. We briefly discuss relational and semi-structured data management in the next two sections.

2.3.1 Relational Data Management

Relational data management has been an active research area even before E.F. Codd proposed the relational model of data in 1970 [38]. The oldest working system that is a recognizably modern database may be IBM's IMS database, released in

1968 [100].

The various software systems used to maintain relational databases are known as relational database management systems (RDBMS). A variety of commercial relational databases systems were developed in many different institutions, including IBM DB2, Oracle RDBMS and Microsoft SQLServer. In addition to the data management system, SQL (*i.e.*, Structured Query Language) was developed as the standard language for querying and maintaining the database. SQL was one of the first commercial languages for Codd's relational model [38].

There are many active research areas on database applications. One of the most pervasive challenges is data integration. The goal is to make it easier to query across multiple autonomous, heterogeneous data sources [58]. Data integration is crucial especially in large enterprises that own a multitude of databases. Most of the time, the data sets are independently produced by multiple researchers.

In summary, there are a variety of tools available that can easily manage relational data, including querying and integrating relational data, and many other applications.

2.3.2 Semi-structured Data Management

Another prominent line of work on data management is on semi-structured data sources, Most of this work focuses on data sources such as Extensible Markup Language (XML).

XML is a markup language that defines a set of rules for encoding documents in a format that is both user- and machine-readable. XML is an application profile of SGML, which was used by early digital-media publishers in the late 1980s, even prior to the rise of the web. The first (XML 1.0) was initially defined in 1998. It has undergone minor revisions since then. Nowadays, XML is widely used everywhere in the world.

The research on XML data mainly focuses on query tree-structured data. A few XML query languages were proposed: Lorel [1], XML-QL [45], XML-GL [26], Quilt [28], XPath [11] and XQuery [17]. Of all the existing XML query languages, XQuery is being standardized as the major XML query language.

In summary, many different tools have been proposed to query or manage semi-structured data.

CHAPTER III

Senbazuru: Extraction on Spreadsheet Structure

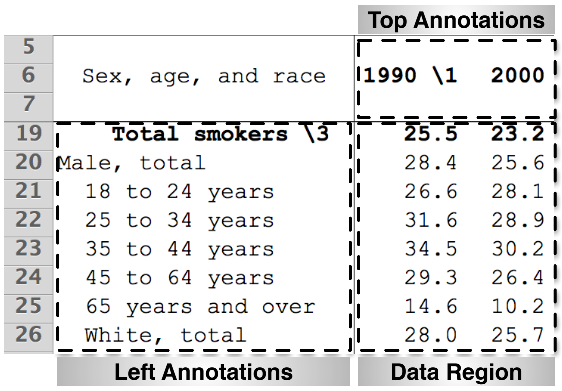
3.1 Problem Overview

One notable form of web statistical data is spreadsheets. Spreadsheets are an extremely popular data management tool, allowing users to complete a range of data tasks commonly associated with relational systems: projection, sorting, aggregation, and simple ETL (Extract, Transform and Load) jobs. Moreover, spreadsheets are often contain data that are roughly relational, but the schema is often designed for human consumption, thus entirely *implicit*.

In this chapter, we consider the transformation from spreadsheets to the relational form on a specific type of spreadsheet which we call the *data frame* spreadsheet. For example, the spreadsheet in Figure 3.2 shows a data frame spreadsheet about the smoking rate downloaded from the government's Statistical Abstract of the United States.[1]. Each row clearly represents a different configuration of the smoking rate; for example, 13.7 in the value region is the rate for people with constraints Male, White, 65 years and over in the annotation region, and it yields an annotating *relational tuple* at the bottom. But there are two main problems here.

First, the spreadsheet only implicitly indicates which cells carry *values* versus *annotations*. Often a spreadsheet is a mix of annotations, values, and other elements

[1] `http://www.census.gov/compendia/statab/2012/tables/12s0204.xls`



5			
6	Sex, age, and race	1990 \1	2000
7			
19	**Total smokers \3**	**25.5**	**23.2**
20	Male, total	28.4	25.6
21	18 to 24 years	26.6	28.1
22	25 to 34 years	31.6	28.9
23	35 to 44 years	34.5	30.2
24	45 to 64 years	29.3	26.4
25	65 years and over	14.6	10.2
26	White, total	28.0	25.7

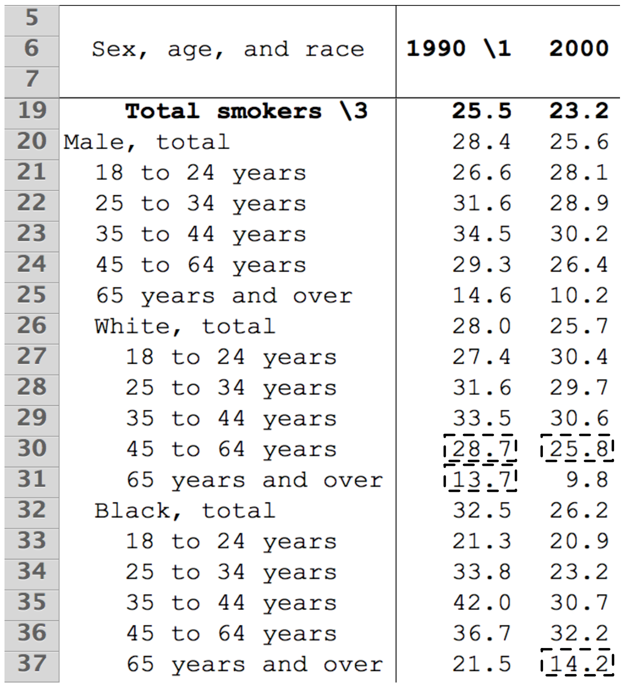
Figure 3.1: The three semantic components of a data frame spreadsheet.

such as titles and footnotes. These elements are not easily distinguished from each other. As shown in Figure 3.1, we call it a ***data frame*** structure, which consists of two rectangular annotation regions (left and top) and a single rectangular data region.

Second, the spreadsheet does not explicitly indicate which *annotations* describe which *values*. If the leftmost column is processed naïvely, rows 25, 31, and 37 will yield three tuples that have different smoking rates for 65 years and older. All three extracted tuples are incorrect, as none will contain any mention of the annotation Male. We call this implicit mapping relationship the ***hierarchical structure*** in spreadsheets.

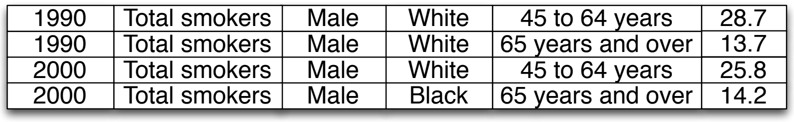
In summary, Figure 3.2 shows a clean, high-quality spreadsheet, but extracting relational data from it requires us to: (1) *data frame extraction* — detect *annotations* and *values*, (2) *hierarchy extraction* — identify the hierarchical structure of left and top attributes, and (3) *relation construction* — generate a *relational table* for each value in the spreadsheet.

In this chapter, we present Senbazuru, a prototype spreadsheet database management system that is able to extract relational information from a large number of spreadsheets. We introduce the data model, data sources, and We present the system pipeline and our approaches to convert data frame spreadsheets into a rela-



	Sex, age, and race	1990 \1	2000
5			
6	Sex, age, and race	1990 \1	2000
7			
19	**Total smokers \3**	**25.5**	**23.2**
20	Male, total	28.4	25.6
21	18 to 24 years	26.6	28.1
22	25 to 34 years	31.6	28.9
23	35 to 44 years	34.5	30.2
24	45 to 64 years	29.3	26.4
25	65 years and over	14.6	10.2
26	White, total	28.0	25.7
27	18 to 24 years	27.4	30.4
28	25 to 34 years	31.6	29.7
29	35 to 44 years	33.5	30.6
30	45 to 64 years	28.7	25.8
31	65 years and over	13.7	9.8
32	Black, total	32.5	26.2
33	18 to 24 years	21.3	20.9
34	25 to 34 years	33.8	23.2
35	35 to 44 years	42.0	30.7
36	45 to 64 years	36.7	32.2
37	65 years and over	21.5	14.2

(a) A spreadsheet example



1990	Total smokers	Male	White	45 to 64 years	28.7
1990	Total smokers	Male	White	65 years and over	13.7
2000	Total smokers	Male	White	45 to 64 years	25.8
2000	Total smokers	Male	Black	65 years and over	14.2

(b) Relational table for the four values in the spreadsheet

Figure 3.2: A spreadsheet about smoking rates, from the Statistical Abstract of the United States.

tional form in Section 3.3. We conduct the extensive experiments on extracting the data frame and the hierarchical structure in Section 3.4. We demonstrate the system Senbazuru in Section 3.5. Finally we discuss related work in Section 3.6 and conclude in Section 3.7.

3.2 Preliminary

In this section, we will introduce the spreadsheet terminology and the data sources that we use throughout this chapter. We also collect the statistics on web spreadsheets in order to better design the Senbazuru system.

3.2.1 Terminology

In its most generic incarnation, a spreadsheet is simply an $M \times N$ grid of cells, in which each cell can contain a string, a number, or nothing. In practice, most spreadsheets, especially the high-quality ones that carry data that we want to extract, have substantially more structures. We make two assumptions about the spreadsheets we will process without seriously compromising our approach's generality.

Data Frames – First, we focus on a prototypical form of spreadsheet that we call a *data frame*. Figure 3.1 shows the three components that make up a data frame: Two rectangular *annotation regions* (*left* and *top*) and a single rectangular *data region*. For each data in the *data region*, there is usually at least one *annotation* in the *top* and *left* annotation regions. For example, in Figure 3.1, the data 14.6 has annotations 65 years and over, Male and 1990.

Hierarchies – Second, we focus on *hierarchical* spreadsheets. We assume a spreadsheet is *hierarchical* if the annotations in the *top* or *left* annotation region exhibit a hierarchical tree structure of at least two layers. Each annotation region has a notional tree that characterizes how each annotation describes the *data region*. For example, in Figure 3.2, all the data at row 31 have a *direct* annotation of 65 years and older and *indirect* annotations of White, Male and Total smokers in the *left* hierarchy. Note that the annotation hierarchies are *not* ontologies. For example, Male is not a super-category of its children.

Relational Tuples — Given the annotation hierarchies for a spreadsheet, we can recover the equivalent relational tuples. For each data in the data region, we generate a relational tuple that consists of the following: (1) The data itself; and (2) Its direct annotation plus all of its indirect annotations in the *left* and *top*. For example, in Figure 3.2, the data 28.7 has a direct annotation of 45 to 64 years and indirect annotations of White, Male and Total smokers in *left*. Similarly, we obtain the direct and indirect annotations in the *top* hierarchy and then generate the relational tuple as shown in Figure 3.2 (b).

Relational Table — Combining multiple relational tuples into a single relational table is sometimes straightforward, but may depend on data-specific details to align each value of a tuple properly to a consistent attribute. Elmeleegy *et al.* [50] proposed a relevant method to perform this alignment in the context of relational table extraction from the web.

3.2.2 Data Sources

We have obtained two spreadsheet corpora:

- ***SAUS*** – The 2010 Statistical Abstract of the United States (SAUS) consists of 1,369 spreadsheet files totaling 70MB. We downloaded the dataset from the U.S. Census Bureau. It covers a variety of topics of general public interest, such as state-level finances, educational attainment, levels of public health, and so on. The data come from different sources inside the government, but to the human eye appears uniformly high in quality of design and content.
- ***WEB*** – Our web dataset (WEB) consists of 410,554 Microsoft Excel files from 51,252 distinct Internet domains. They total 101 GB. We found the spreadsheets by looking for Excel-style file endings among the roughly 10 billion URLs in the ClueWeb09 web crawl [37]. The data come from many different sources and to a human appears to have a wide range in quality.

3.2.3 Web Spreadsheet Statistics

To better design the Senbazuru system, we answer the following critical questions about the general properties of the web spreadsheets and the popularity of the data frame spreadsheets on the web:

1. Where are those web spreadsheets from? The web spreadsheets cover a huge range of topics and show wide variance in cleanliness and quality. Most of

Domain	**# files**	**% total**	**data frame**	**h-top**	**h-left**
www.bts.gov	12435	3.03%	99%	30%	40%
www.census.gov	7862	1.91%	94%	72%	70%
www.stat.co.jp	6633	1.62%	x	x	x
www.bankofengland.co.uk	5520	1.34%	98%	77%	35%
www.ers.usda.gov	4328	1.05%	95%	77%	70%
www.agr.gc.ca	4186	1.02%	87%	77%	81%
www.wto.org	3863	0.94%	96%	61%	77%
www.doh.wa.gov	3579	0.87%	81%	53%	64%
www.nsf.gov	2770	0.67%	96%	53%	76%
nces.ed.gov	2177	0.53%	98%	55%	92%
average	**5335**	**1.30%**	**93.78%**	**61.67%**	**67.33%**

Figure 3.3: The top 10 domains in our web spreadsheet corpus. h-top and h-left are percentages of spreadsheets with a hierarchical *top* or *left* region.

the spreadsheets are statistical data, with a heavy emphasis on government, finance, transportation, etc. We are also interested in the distribution of the spreadsheets from different Internet domains. Figure 3.3 shows the top 10 Internet domains that host the largest number of spreadsheets in the WEB corpus. Nine of the top 10 domains are sites run by the U.S., Japanese, UK, or Canadian governments. Figure 3.4 shows the distribution of spreadsheets among hosting domains. We rank the domains according to the size of their hosting spreadsheets in descending order. The plot indicates that the spreadsheets follow a strongly skewed distribution, with a large number of spreadsheets from relatively few domains and with a large number of domains hosting relatively few spreadsheets.

2. How many of the web spreadsheets consist of data frame structures? To better understand the structure of the WEB spreadsheets, we randomly chose 200 samples and asked a human expert to mark their structures. We found 50.5% of the spreadsheets consist of data frame components and 32.5% have hierarchical top or left annotations. The other 49.5% non-data frame spreadsheets belong to the following categories: 22.0% are Relation spreadsheets that can be converted to the relational model almost trivially (we can simply translate each spreadsheet column

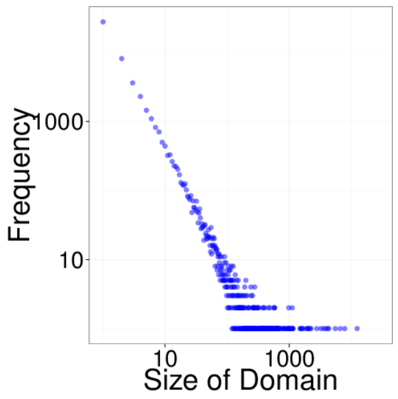


Figure 3.4: The distribution of web spreadsheets.

into a relational table column and translate each spreadsheet row into a relational tuple); 10.5% are Form spreadsheets that are not for data storage and are designed to be filled by a human; 3.5% are Diagram spreadsheets for visualization purposes, and they are often data-intensive without any schema information; and 3% are List spreadsheets that consist of non-numeric tuples. The 10.5% Other spreadsheets are schedules, syllabi, scorecards, or other files whose purpose is unclear. Although there are a variety of categories of spreadsheets on the web, in this paper, we only focus on data frame spreadsheets.

3. How many of the web spreadsheets are hierarchical? Are those hierarchical spreadsheets spread uniformly across the web? As just mentioned, 32.5% of the 200 sample web spreadsheets have hierarchical *top* or *left* annotations in a data frame. To better understand how the hierarchical spreadsheets are distributed in different domains, we randomly selected 100 spreadsheets from each of the top 10 domains, yielding 900 spreadsheets in total.[2] Figure 3.3 shows the fraction of spreadsheets with data frames or hierarchical annotations in the top 10 domains. The ratios are much higher than the fractions we obtained from the general web sample. We also randomly selected 100 spreadsheets from domains hosting fewer than 10 spreadsheets. We found 19% with data frame structures, 4% of which have hierarchi-

[2]www.stat.co.jp is excluded because it is in Japanese.

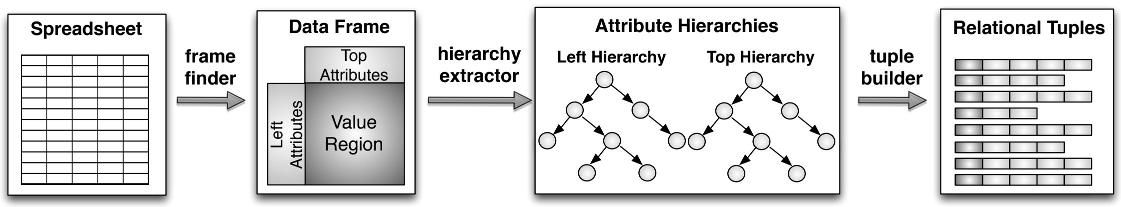


Figure 3.5: The system pipeline for Senbazuru to process a single spreadsheet.

cal *top* annotations and 6% of which have hierarchical *left* annotations. These results suggest that the number of hierarchical spreadsheets differs greatly by domain and may be linked to the domain's popularity or degree of professionalism. Computing the exact distribution of hierarchical spreadsheets among domains would be useful but requires a huge amount of labeled data; we will explore this question in future work. Even without computing that distribution, we have found a huge number of hierarchical spreadsheets: 32.5% of all spreadsheets on the web and more than 60% in popular domains. Therefore, to extract relational data from spreadsheets, we believe our system must process hierarchical-style metadata.

3.3 Spreadsheet Structure Extraction

We now describe our spreadsheet extraction pipeline. The goal of the extraction pipeline is to create a relational model of the data embedded in data frame spreadsheets: it takes in a data frame spreadsheet and emits a relational table.

In the following sections, we first describe our spreadsheet extraction pipeline, and then we discuss the two critical components that are our core contributions. They are the data frame extraction and the hierarchy extraction.

Algorithm 1 TupleBuilder

```
Input: The left hierarchy $H_l$, the top hierarchy $H_t$,
the set of values in the value region $V = \{v\}$
Output: The relational tuples $T = \{t\}$
1: Initiate $T$
2: for each $v \in V$ do
3:     Initiate $t$
4:     Get annotating attributes for $v$ from $H_l$ as $\{a_l\}$
5:     Get annotating attributes for $v$ from $H_t$ as $\{a_t\}$
6:     $t \leftarrow v \cup \{a_l\} \cup \{a_t\}$
7:     $T \leftarrow T \cup t$
8: end for
```

3.3.1 System Pipeline Overview

We developed the spreadsheet management system Senbazuru. The extraction pipeline consists of three components, as shown in Figure 3.5. They are the frame finder, the hierarchy extractor, and the tuple builder. The frame finder identifies the data frames, locating *attribute* regions and *value* regions. The hierarchy extractor recovers the hierarchical metadata from spreadsheets, and the tuple builder generates a relational tuple for each value in the value region.[3].

The tuple builder is straightforward, as long as the previous steps are accurate. We generate a relational tuple for each value in the value region, annotating each one with relevant annotations from the annotation hierarchies. For example, Figure 3.2 shows the full six-field tuple we want to recover for the highlighted value 13.7. The tuple builder is also algorithmically straightforward. It processes the extracted annotation hierarchies and the value region to generate a series of relational tuples. As described in Algorithm 1, for each value v, we find the its annotating annotations along the path to the root in the annotation hierarchies for both *left* and *top* attribute regions. The tuple builder relies entirely on the frame finder and hierarchy extractor for correctness.

We discuss the frame finder and the hierarchy extractor in the next two sections.

[3]It should be able to work on both *flat* and *hierarchical* spreadsheets because we treat *flat* spreadsheets as a special case of *hierarchical* ones

3.3.2 Data Frame Extraction

The frame finder identifies the value region and the *top* and *left* attribute regions. It receives a raw spreadsheet as input and emits geometric descriptors of the data frame's three rectangular regions. We define the problem as follows:

Definition III.1. (Frame Finder) Let a spreadsheet be a grid of cells $\mathbf{c} = \{c_{ij}\}$, where i represents the row index and j represents the column index. The frame finder assigns each cell $c_{ij} \in \mathbf{c}$ with a label $l_{ij} \in \mathrm{L} = \{\mathsf{top}, \mathsf{left}, \mathsf{value}, \mathsf{other}\}$, where top represents *top annotations*, left represents *left annotations*, value represents *values*, and other represents everything else.

To simplify the problem, we assume that the structure of the spreadsheets has the following property: there may be multiple data frames in a spreadsheet, but they only stack in the vertical dimension.[4] This assumption allows us to treat data frame-finding as a problem of row labeling. Therefore, we start with the row labeler task, which assigns each row in a spreadsheet to one of the following four categories: title, header, data, or footnote. The label title represents a spreadsheet title, header represents a row that contains *top* annotations only, data represents a row that contains *left* annotations or *values*, and footnote is information that annotates the main contents. As in Figure 3.2, rows 5-7 are labeled header and rows 19-37 are labeled data. A formal definition is as follows:

Definition III.2. (Row Labeler) Let $\mathbf{r} = \{r_1, r_2, ..., r_N\}$ be a set of variables representing the non-empty rows in a spreadsheet. The row labeler assigns each $r_i \in \mathbf{r}$ with a label $l_i \in L = \{\mathsf{title}, \mathsf{header}, \mathsf{data}, \mathsf{footnote}\}$.

We observe the following two types of signals that the row labeler should use to automatically assign semantic labels to each non-empty row: (1) the properties of

[4]In fact, we found less than 2% of the 900 spreadsheets in the top 10 most popular HTTP domains violate the assumption.

each non-empty row indicate its semantic label, such as its fonts and keywords; and (2) the labels assigned to adjacent rows are highly related. For example, if we know the current row is a header row, it is highly probable that the next row is a header or data row. Therefore, we employ an approach based on a linear-chain, conditional random field (CRF) [75] to exploit these two types of signals. Pinto *et al.* [86] used linear-chain CRFs to obtain labels for textual tables in government statistical text reports. We also use the linear-chain CRFs to obtain the semantic labels for each row of a spreadsheet, and our training and inference procedure is the same. However, with the access to spreadsheet APIs, we are able to build the CRFs with a richer set of features, such as the alignment and indentation information that is hard to obtain from plain text report. Our extraction features fall into two main categories: *layout features* test visual characteristics of a row, and *textual features* test the contents of the row. Each of the features is a binary function, taking in a given row in a spreadsheet as the input and emitting a 0/1 Boolean value as the output. The features attempt to test whether the properties of a row are an indication of a certain category in {title, header, data, footnote}. The features are listed in Table 3.1.

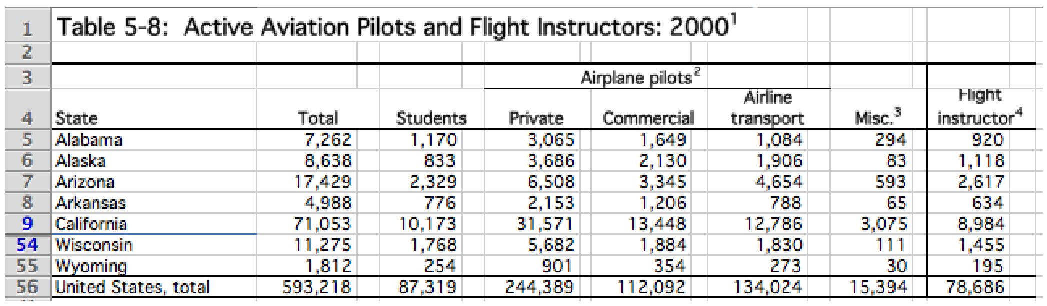
Once we have labels for each row in a spreadsheet, we can construct the correct data frame regions. The vertical extent of a value region is described by the set of rows marked data, and its horizontal extent is determined by finding regions of numeric values. The *top* attribute region is delimited by all header rows, and the *left* attribute region is everything to the left of the value region.

3.3.3 Hierarchy Extraction

The hierarchy extractor recovers the annotation hierarchies. This step receives a data frame with *top* and *left* regions as input and emits hierarchies as output: one for *left* and one for *top*. These trees describe the hierarchical annotation relationship among annotations in the *top* and *left* regions. For example, in Figure 3.2, row 31

Layout Features	
1	Has a bold font cell
2	Has a cell reaching the left bound
3	Has a cell reaching the right bound
4	Has a cell with indentations
5	Has a center-aligned cell
6	Has a left-aligned cell
7	Has a merged cell
8	Has only one column
Textual Features	
1	Contains colon
2	Contains punctuations
3	Has a cell with with a word count > 40
4	Numeric cells within year range ratio > 0.6
5	Row is blank
6	With all words in lowercases
7	With all words capitalized
8	With all words starting with capitals
9	With numeric cells ratio > 0.6
10	With words starting with "table"

Table 3.1: Extraction features for the frame finder.



1	Table 5-8: Active Aviation Pilots and Flight Instructors: 2000[1]								
2									
3				Airplane pilots[2]					
4	State	Total	Students	Private	Commercial	Airline transport	Misc.[3]	Flight instructor[4]
5	Alabama	7,262	1,170	3,065	1,649	1,084	294	920
6	Alaska	8,638	833	3,686	2,130	1,906	83	1,118
7	Arizona	17,429	2,329	6,508	3,345	4,654	593	2,617
8	Arkansas	4,988	776	2,153	1,206	788	65	634
9	California	71,053	10,173	31,571	13,448	12,786	3,075	8,984
54	Wisconsin	11,275	1,768	5,682	1,884	1,830	111	1,455
55	Wyoming	1,812	254	901	354	273	30	195
56	United States, total	593,218	87,319	244,389	112,092	134,024	15,394	78,686

Figure 3.6: An example of the hierarchical top annotations in spreadsheets.

is annotated by annotations at rows 26, 20, and 19. An example of a *top* hierarchy can be found in Figure 3.6, where the *attribute* Airplane pilots annotates the *attribute* Airline transport. Now we formally describe the problem of recovering the annotation hierarchy for a single region as follows:

3.3.3.1 Approach Overview

In this section, we describe the hierarchy extraction task in detail. We have proposed a new two-phase *semiautomatic* approach based on an undirected graphical model to extracting spreadsheet annotation-to-data mappings accurately and with little user effort.

First, the *automatic extractor* receives spreadsheets as input and computes a mapping without user interaction. Based on an undirected graphical model, it exploits single-spreadsheet graphical style hints, such as the font and typographic alignment, that are obvious to a human observer. It also identifies and exploits correlated extraction decisions; these correlated decisions can appear within one spreadsheet or between two unrelated spreadsheets. Our resulting automatic extractor obtains accuracy that beats a baseline approach by up to 91% on a large workload of spreadsheets.

Second, our system offers an *interactive repair* phase, in which a user repeatedly reviews and corrects the automatic extractor's output until no errors remain. We expect a user will review the automatic extractor's output. But our interactive repair is more than simply asking a user to fix every single extraction error. We again exploit the correlations among different extraction decisions to make more effective use of each user repair operation. A user's single repair can be silently and probabilistically applied to multiple possible errors, allowing us to *amortize* the user's effort over many likely extractor mistakes. Building a model that can perform this amortization, and managing the inadvertent errors that such an approach might introduce (a problem we call *backtracking*), is one of this paper's core contributions.

Figure 3.7 shows an example of the user interface for applying repairs. The left side of the diagram indicates the initial hierarchy obtained by the automatic extractor for Figure 3.2. The dashed arrow shows that a user performs a repair by clicking and dragging White so that it becomes a child of Male, indicating that Male annotates White. This one repair operation triggers multiple error fixes, including setting Male

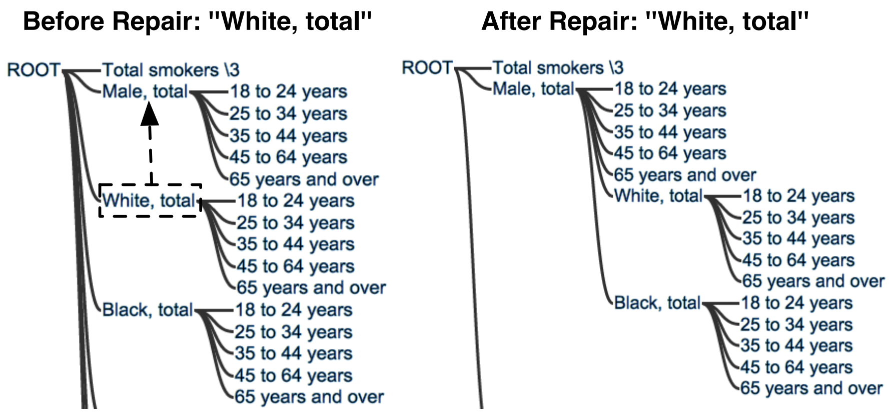


Figure 3.7: Our user interface for repairing mappings.

to also annotate Black. By making our system part of the user's natural review-and-repair loop, we can reduce the number of manual repairs by up to 71% when compared to our already-effective automatic extractor.

3.3.3.2 Problem Definition

We now formally describe the hierarchy extraction task. The task of hierarchy extraction is to detect all of the *ParentChild* pairs $P = \{ParentChild(a_i, a_j)\}$ in an annotation region A. One way to model this problem is to create a Boolean variable x to represent a *ParentChild* pair candidate (a_p, a_c) for every annotation pair $a_p, a_c \in A$. Each variable x takes a label $l \in L = \{true, false\}$, and x holds true if a_p is the parent of a_c. For example, Figure 3.8 shows a portion of the created variables for Figure 3.2's *left* metadata. Each oval node corresponds to a single boolean *ParentChild* decision. For example, setting the node (18 to 24 years, Male) to true indicates that 18 to 24 years is the hierarchy *parent* of Male.

The spreadsheet hierarchy extraction task, thus, amounts to recovering all the *ParentChild* pairs for its annotation regions. For example in Figure 3.2, the solution for mappings in *left* is a set of all its *ParentChild* pairs {*(row-19, row-20), ..., (row-32, row-37)*}.

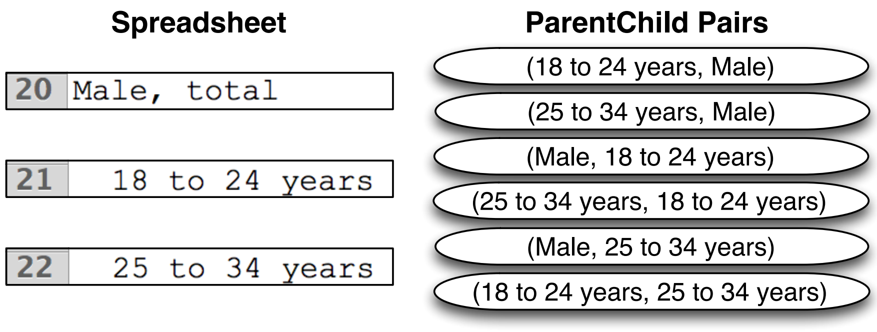


Figure 3.8: A sample of *ParentChild* variables.

3.3.3.3 Our Observations

We now formally describe our problem and observations. The task of hierarchy extraction is to detect all of the *ParentChild* pairs $P = \{ParentChild(a_i, a_j)\}$ in an annotation region A. One way to model this problem is to create a Boolean variable x to represent a *ParentChild* pair candidate (a_p, a_c) for every annotation pair $a_p, a_c \in A$. Each variable x takes a label $l \in L = \{true, false\}$, and x holds true if a_p is the parent of a_c. For example, Figure 3.8 shows a portion of the created variables for Figure 3.2's *left* metadata. Each oval node corresponds to a single boolean *ParentChild* decision. For example, setting the node (18 to 24 years, Male) to true indicates that 18 to 24 years is the hierarchy *parent* of Male.

However, simply enumerating all pairs in a region A can yield thousands of variables. In practice, it is possible to greatly reduce the set of *ParentChild* candidates with a few heuristics. [5] Failing to create a node for a true *ParentChild* relationship clearly means that we will predict the child's parent incorrectly. A wrong *ParentChild* prediction is bad, but not catastrophic: the user can still describe the correct relationship during interactive repair. Failing to create a node for a true *ParentChild* relationship is bad, but not catastrophic: the user can still describe the correct rela-

[5]We prioritize *ParentChild* candidates in which the typographic styles of the two nodes differ. We also prioritize pairs that are geometrically close to each other in the spreadsheet. Testing on our 200 testbed spreadsheets for SAUS and WEB, our heuristics only incorrectly filtered out just 0.01% and 0.13% of correct pairs, respectively.

tionship during interactive repair.

***ParentChild* Pair Properties** — A true *ParentChild* variable may be indicated by the surrounding style and layout information. For example, a variable that describes annotations which are physically close is likelier to be true than a variable that describes annotations that are physically distant. We formulated 32 features for evaluating a *ParentChild* variable.

For *left* attributes, given a *ParentChild* pair candidate (a_i, a_j), we employ a set of features to characterize its properties, thus determining whether it is a true *ParentChild* pair. The testing features include *unary features* and *binary features*, as shown in Table 3.2. The unary features apply on each of the child and parent attributes, and the binary features apply on the attribute pair.

For *top* attributes, given a *ParentChild* pair candidate (a_i, a_j), we utilize a set of layout features to characterize the properties of the attribute pair, thus determining whether it is a true *ParentChild* pair. The features we used are shown in Table 3.3.

Correlating ParentChild Decisions — *ParentChild* decisions can be correlated; knowing the assignment of one *ParentChild* variable sheds light on some others. We found the following four types of correlations.

Correlation (i) — Stylistic Affinity. When two *ParentChild* variables in the same spreadsheet have identical visual style for parents and for children, it is likely that the two variables should be decided together. For example in Figure 3.9 (a), the two *ParentChild* variables ((White, College) and (Male, 18 to 24 years)) should be decided together because the parents (White and Male) share the same typographic style, as do the children (College and 18 to 24 years). We say that two variables have stylistic affinity when the parents and children share a range of visual qualities: alignment, indentation, capitalization, typeface, type size, type style (*i.e.*, bold or italicized), and use of certain special strings (*i.e.*, a colon, a number, or the word "total"). Note that stylistic affinity only makes sense when testing *ParentChild* pairs

	Unary Extraction Features
1	Attribute has underline
2	Attribute contains keywords like “total”
3	Attribute contains colon
4	Attribute is bold
5	Attribute is center aligned
6	Attribute is italic
7	Attribute is numeric
8	Attribute letters are all capitalized
9	Is the first attribute
10	Is the last attribute
	Binary Extraction Features
1	Attribute pair is adjacent
2	Attribute pair’s indentation is equal
3	Attribute pair’s style is adjacent in the region
4	Child’s font size is smaller than parent’s
5	Child’s indentation is greater than parent’s
6	Child’s row index is greater than parent’s
7	Child’s style is the same as the first attribute
8	Has blank cells in the middle
9	Has middle cell with indentation between the pair’s
10	Has middle cell with indentation larger than the pair’s
11	Has middle cell with indentation less than the pair’s
12	Has middle cell with style different from the pair’s
13	Has middle cell containing keywords like “total”
14	Parent is the root

Table 3.2: Extraction features for the hierarchy extractor on left attributes.

within a single spreadsheet; different spreadsheets may have different or contradictory ways of visually indicating the *ParentChild* relationship.

Correlation (ii) — Metadata Affinity. If we have a metadata resource available, we can use it to find additional correlations between *ParentChild* variables both *within and between* spreadsheets. For example in Figure 3.9 (b), the two *ParentChild* candidates, (White, Female) and (Black, Male), should be decided together because the parents (White and Black) belong to the same semantic category *race*; similarly the children (Female and Male) belong to *gender*.

Fortunately, we are able to synthesize a *domain-specific metadata resource* from a corpus of spreadsheets. Our central observation is that any useful category of

Layout Extraction Features	
1	Child has no cell right above
2	Child is at the uppermost header row
3	Has a cell in the middle
4	Parent cell covers child's column
5	Parent is on the left of child
6	Parent is on the right of child
7	Parent is right above child
8	Parent is the root

Table 3.3: Extraction features for the hierarchy extractor on top attributes.

annotations — whether a general-purpose one like gender or a hyper-specific one such as chemicalPrecursor — will likely appear in many datasets. Further, annotations drawn from the same category (such as Male and Female) often appear as siblings in an extracted annotation hierarchy. We measure whether two annotations belong to the same category by testing how strongly the annotations appear as siblings in a large number of extracted hierarchies. We perform the test as follows:

1. Extract all annotation hierarchies from a corpus of spreadsheets using a simple classifier or a version of our automatic extractor that does not use metadata information. For each parent annotation, we create a *sibling set* that contains all of its child annotations.

2. Count the number of sibling sets where an annotation a is observed. Divide by the number of sibling sets to obtain $p(a)$, the probability that a randomly chosen sibling set contains a.

3. Count the number of sibling sets where the annotation pair a_i and a_j co-occur together. Divide by the number of sibling sets to obtain $p(a_i, a_j)$, the probability that a randomly chosen sibling set contains both a_i and a_j.

We can then measure the extent to which two annotations a_i and a_j are observed as siblings (and thus are likely to be in the same category) by computing the pointwise mutual information (PMI): $PMI(a_i, a_j) = log\frac{p(a_i,a_j)}{p(a_i)p(a_j)}$.

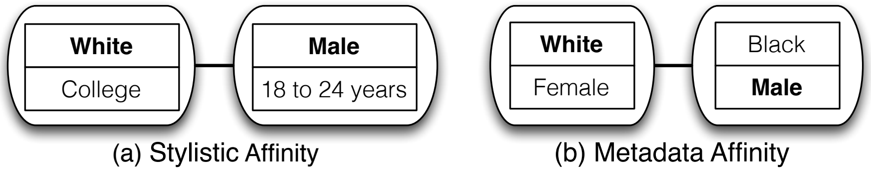


Figure 3.9: An example of stylistic affinity shown in (a) and metadata affinity shown in (b).

Let $x_1 = (a_{p1}, a_{c1})$ and $x_2 = (a_{p2}, a_{c2})$ be two variables in the CRF. The two variables x_1 and x_2 have **metadata affinity** if and only if $PMI(a_{p1}, a_{p2}) > \delta$ and $PMI(a_{c1}, a_{c2}) > \delta$, where δ is a predefined threshold.

Correlation (iii) — Adjacent Dependency. If we consider the *ParentChild* pairs of a *single spreadsheet* as a sequence, adjacent variables often follow a transition pattern of the labels.

Correlation (iv) — Aggregate Design. There are two further constraints that reflect typical overall spreadsheet design and ensure that the resulting variable assignment yields a legal hierarchy (*i.e.*, a tree).

The first is the *orientation constraint*. Spreadsheets tend to have an “upward” or “downward” orientation; that is, parents do not appear above their children in some cases and below their children in other cases. For example in Figure 3.8, the pairs (Male, 18 to 24 years) and (25 to 34 years, 18 to 24 years) cannot both be *true*.

The second is the *one-parent constraint*. We enforce our assumption that *ParentChild* relationships genuinely form a tree; one annotation can only have one parent. Put another way, for all of the variables sharing the same child, only one of them is *true* and the rest are *false*. For example, in Figure 3.8, (Male, 25 to 34 years) and (18 to 24 years, 25 to 34 years) could not both be true.

User Repair Interaction — The interactive repair phase allows the user to check and fix any *ParentChild* decision mistakes made by the system. The goal of interactive repair is to save user effort by using each explicit user-given repair to fix

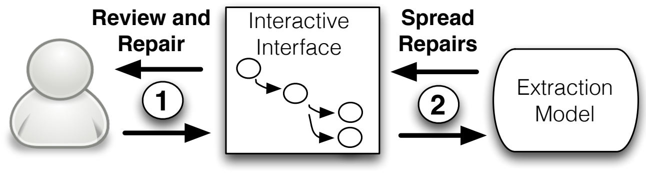


Figure 3.10: Interaction cycle for interactive repair.

not just the error in question, but also additional extraction errors that the user never directly inspects. In this section, we describe the interactive repair workflow in more detail, plus how to modify the graphical model to support the repair process. Finally, we describe the training and inference methods.

During interactive repair, we assume a user always fixes extraction errors correctly. We do not focus on the problem of noisy user-labeled data, and there is crowdsourcing literature on how to ensure trustworthy answers [44].

We now discuss our model workflow for interactive repair. The system starts by presenting to the user the initial extraction results computed by the automatic extraction and then enters the interactive repair interaction loops (shown in Figure 4.5). For each loop, the system takes two steps:

1. Review and Repair — A user is able to repair an error in the extracted hierarchy by dragging and releasing an annotation node on the interface. One ***user repair*** action changes an annotation's parent from one to another. For example in Figure 3.7, a user changes the parent annotation of White from Root to Male.

A *user repair* operation has two implications. First, the variable x that represents the new *correct ParentChild* relationship is set to *true*. In the case of Figure 3.7, the variable (Male, White) is *true*. Second, all the other variables that represent *ParentChild* relationships sharing the same child with x are set to *false*. In the case of Figure 3.7, variables (Root, White) and (Total smoker, White) are *false*.

As a result, a user's repair to an extraction error yields a set of label assignments to some *ParentChild* variables.

2. Spread Repairs — The system now aims to save user effort by repairing other similar extraction errors. Of course, the system has already given its best extraction estimate in the automatic extraction phase, so it does not know where any latent extraction errors are. But we have already used different kinds of affinity to connect two *ParentChild* decisions that are highly likely to share the same label.

It is appealing to spread each user-repaired label on a variable to other variables that are identified by affinity correlations (i) and (ii). But simply propagating assignments might introduce errors where none previously exist, which we call the ***backtracking*** problem. We want to leverage the graphical model to integrate probability information with the node, edge, and global correlations to prevent backtracking.

3.3.3.4 A Graphical Model Based Approach

Now we describe how we encode the *ParentChild* pair properties, correlating *ParentChild* decisions, user repair interaction into the graphical model as described in Section 2.2.2.

Node Potentials – Each variable x in the graphical model represents a *ParentChild* decision, which takes a label $l \in L = \{true, false\}$. We define the **node potential** $\theta(x, l)$ on each variable x assigned the label l. The node potentials depend on Boolean feature functions $\{f_k(x, l)\}$ (The 32 features mentioned in Section 3.3.3.3) and trained weights $\{w_k\}$ associated with the feature functions:

$$\theta(x, l) = \sum_k w_k f_k(x, l) \tag{3.1}$$

Edge Potentials – The correlations (i) (ii) and (iii) mentioned in Section 3.3.3.3 are encoded in the graphical model as pairwise *edge potentials*. The **edge potential** $\theta(x, l, x', l')$ is defined on two variables x and x' in the graphical model on their assignments l and l' if the variables x and x' are found to be correlated in one of the three ways mentioned above. We define,

$$\theta(x, l, x', l') = [\![l = l']\!] \sum_{e} w_e f_e(x, x') \tag{3.2}$$

where $[\![l = l']\!]$ takes the value 1 when $l = l'$ and 0 otherwise; $\{w_e\}$ are the associated weights. The edge features $\{f_e(x, x')\}$ test which type of correlation x and x' belong to and whether x and x' have the same child/parent.

Global Potentials – Finally we encode the correlation (iv) mentioned in Section 3.3.3.3 as *global potentials*. Let $x = (a_p, a_c)$ and $x' = (a'_p, a'_c)$ be two arbitrary variables in the graphical model with the assigned labels l and l', and $R(a)$ represents the row number of an annotation a. We now define two **global potentials**: $\phi_a(\mathbf{x}, \mathbf{l})$ to encode the *orientation constraint* and $\phi_b(\mathbf{x}, \mathbf{l})$ to encode the *one-parent constraint*:

$$\begin{aligned} \phi_a(\mathbf{x}, \mathbf{l}) = [\![\exists x, x' \in \mathbf{x} \ s.t. \ l = true, l' = true, \\ R(a_p) > R(a_c), R(a'_p) < R(a'_c)]\!]^{0}_{-\infty} \end{aligned} \tag{3.3}$$

$$\phi_b(\mathbf{x}, \mathbf{l}) = [\![\forall c, \sum_{p} [\![l = true]\!]^{0}_{1} = 1]\!]^{-\infty}_{0} \tag{3.4}$$

where $[\![C]\!]^{value2}_{value1}$ takes the *value 1* when condition C is true and *value 2* otherwise.

Repair Potentials – Here, we describe how to encode the user repair interaction to the graphical model. Algorithm 2 shows the **SpreadRepair** function that is invoked after each user repair operation (described in step 2 of the previous section). First, when a new repair arrives, we translate this new repair and all the previous repairs to the assignments on a set of variables $\mathbf{x_r} = \{x_{r_1}, ..., x_{r_n}\}$ with labels $\mathbf{l_r} = \{l_{r_1}, ..., l_{r_n}\}$. Second, we generate a new graphical model G' by adding the *repair potentials* to the original automatic extraction graphical model G. The repair potentials capture the pairwise correlation between variables, and we describe the repair potentials later. Finally, we condition on the known variables $\mathbf{x_r}$ and infer labels for the variables of G'. The inferred labels are returned as the updated answer.

Note that by adding repair potentials only to nodes that we also condition on, we add information to the inference process without increasing any inference-time computational complexity. The conditioning process essentially removes the observed nodes and their edges prior to the inference [71].

There is nothing in principle that prevents our system from ***backtracking***, unless we can find heuristics to propagate the assignments fully correctly, which is often hard especially on real-world datasets. However, our mechanism is designed to prevent it. First, we only probabilistically propagate known variable assignments to others, via the repair potentials. Second, this probabilistic repair information is combined with all our previous information sources: the node potentials, edge potentials and global potentials. The hope is that adding high quality new information to the automatic extraction graphical model (instead of treating spreading repairs as a non-probabilistic post-processing stage) will yield better outcomes overall.

We now discuss how to generate the repair potentials. The **repair potential** $\varphi(x, l, x_r, l_r)$ describes the likelihood that the repaired node's label should be spread to a similar *ParentChild* node. A repair potential exists between an observed variable $x_r \in \mathbf{x_r}$ and a variable $x \in \mathbf{x}$ if x_r and x exhibit either stylistic affinity or metadata affinity. In other words, repair potentials do not introduce any novel edges to the graphical model: the edges of repair potentials are a subset of the edges derived from correlations (i) and (ii). The repair potentials are defined as:

$$\begin{aligned} \varphi(x, l, x_r, l_r) = [\![Stylistic(x, x_r)]\!] f_s(x, l, x_r, l_r) \\ + [\![Metadata(x, x_r)]\!] f_m(x, l, x_r, l_r) \end{aligned} \tag{3.5}$$

$[\![C]\!]$ takes the value 1 when condition C is true; otherwise 0. $Stylistic(x, x_r)$ and

$Metadata(x, x_r)$ test whether x and x_r have stylistic or metadata affinity. The two feature functions f_s and f_m weigh the strength of influence from observed variables to unobserved ones. They characterize how similar the unobserved variables are to the observed ones. To be precise, we define $f_s(x, l, x_r, l_r) = logP_s(x = l \mid x_r = l_r)$. where $P_s(x = l \mid x_r = l_r)$ represents the probability of a variable x taking the label l once we observe a variable x_r with the label l_r. This probability can be derived from training data. For example, in the training data, among 1000 *stylistic affinity* edges detected, 900 of them connect two variables with the same assignment. We then set $P_s(x = true|x_r = true) = 0.9$ and $P_s(x = false|x_r = true) = 0.1$. The f_m potentials are defined in the same way.

Summary — We can now formally define the spreadsheet annotation hierarchy extraction framework, which supports both automatic extraction and interactive repair.

Let G be a graphical model that has a set of variables $\mathbf{x} = \{x_1, ..., x_n\}$ where each $x_i \in \mathbf{x}$ represents a *ParentChild* candidate in an *annotation region* and takes a label l_i from L ={*true, false*}. Let $\mathbf{l_r}$ be the set of repair-induced labels on variables $\mathbf{x_r}$. We define node potentials (Equation 1), edge potentials (Equation 2), global potentials (Equation 3 and 4), and repair potentials (Equation 5) in G. The joint distribution of the graphical model G is:

$$P(\mathbf{l} \mid \mathbf{l_r}, \mathbf{x}) = \frac{1}{Z(\mathbf{w})} exp(\sum_x \theta(x, l) + \sum_x \sum_{x'} \theta(x, l, x', l') \\ + \sum_{k \in \{a,b\}} \phi_k(\mathbf{x}, \mathbf{l}) + \sum_x \sum_{x_r \in \mathbf{x_r}} \varphi(x, l, x_r, l_r))$$

3.3.3.5 Training and Inference

In this section, we discuss how to train model parameters and infer assignments to variables in the graphical model.

Algorithm 2 SpreadRepair

```
Input:  All user repairs $R$, and automatic extraction model $G$
Output:  New assignments $\mathbf{l}$ to all variables of $G$.
1: From user repairs $R$, create repair-induced variables $\mathbf{x_r}$ with labels $\mathbf{l_r}$
2: Build new model $G'$ by adding to $G$ the new repair potentials based on $\mathbf{x_r}$. $G'$ has the
   same set of nodes (variables) as $G$.
3: Condition on $\mathbf{x_r}$ and infer assignments $\mathbf{l}$ to $G'$ (and thus, $G$)
```

In the graphical model, we only have unknown parameters for *node* and *edge potentials*. Assuming that no user repairs are involved, we can write the joint probability as,

$$\frac{1}{Z(\mathbf{w})} exp(\sum_x \theta(x,l) + \sum_x \sum_{x'} \theta(x,l,x',l') + \sum_{k\in\{a,b\}} \phi_k(\mathbf{x},\mathbf{l}))$$

Let $\mathbf{w} = \{w\}$ be the set of parameters for *node* and *edge potentials*. Given training data $D = \{\mathbf{x}, \mathbf{l}\}$ that describes hand-labeled correct hierarchies of the training spreadsheets, we estimate $\mathbf{w}$ for node and edge potential functions, $\theta(x,l)$ and $\theta(x,l,x',l')$. A common choice of regularization to avoid overfitting is to add a penalty on weight vectors, based on the Euclidean norm of $\mathbf{w}$ and on a *regularization parameter* $\frac{1}{2\sigma^2}$. The goal is to maximize the regularized log likelihood:

$$\max_{\mathbf{w}} \sum_x \theta(x,l) + \sum_x \sum_{x'} \theta(x,l,x',l') - logZ(\mathbf{w}) - \sum_i \frac{w_i^2}{2\sigma^2} + C$$

where C is a constant. This is a standard form for parameter estimation, and known techniques, such as conjugate gradient and L-BFGS, can be used to find the optimal parameters for this formula. Previous work [71, 75] discusses this optimization problem and its solution in more detail.

The graphical model described poses a serious computational challenge. Inference is NP-hard if no assumptions are made about the structure of the graph [40], yet our application requires that we infer labels after each user repair to redisplay the updated hierarchy. In order to infer variables in interactive time, we first simplify the graphical model.

Model Simplification — The potential stumbling blocks to efficient inference are the edge and global potentials. (The repair potentials do not complicate the inference because the conditioning algorithm [71] erases observed variables along with all the repair potential edges.) The edge potentials alone can yield more than a million edges on a graph with 37,386 nodes derived from just 100 randomly-chosen WEB spreadsheets (see Table 3.7 for details).

We considered two methods for conducting inference in a limited amount of time: running the tree-reweighted belief propagation algorithm [72] on the full graph, or running an exact inference method on a simplified tree-structured model. Our experiments show that when running on a model derived from 100 random SAUS spreadsheets and repeating this process 10 times, tree-reweighted belief propagation is 48 times slower and 5.4% worse on F1 than the tree-structured model. Thus, at inference time we convert our graphical model into a tree-structured model.

It is not easy to find the tree-structured graphical model that yields the highest-quality results. Exhaustively enumerating all the possible trees in a graph with more than a million edges and 37,000 nodes is impractical. We simply randomly sample edges from each type of pairwise correlation (stylistic, metadata, and adjacency), rejecting any edge that would induce a cycle. We terminate when all nodes are connected. We add all possible metadata edges before adding any stylistic edges, and add all stylistic edges before adding any adjacency edges. We found experimentally that this ordering helped slightly, though different orderings do not change F1 very much: testing on 100 random spreadsheets of SAUS, different orderings changed F1 from 0.8808 to 0.8867 and from 0.8237 to 0.8363 when testing on WEB.

Inference — We can now present our method for approximating the graphical model's optimal assignment. First, we build the model with *node* potentials, tree-structured *edge potentials*, and all the *repair potentials* if there exist any. Given a set

Algorithm 3 EnforcedTreeInference

Input: The variables $\mathbf{x} = \{x\}$ and the annotations $A = \{a_1, ..., a_N\}$ in an annotation region.

Output: The *ParentChild* pairs $P = \{(a_p, a_c)\}$ in the annotation hierarchy and its confidence $confidence$.

```
 1: $P \leftarrow \{\}$, $confidence \leftarrow 0$
 2: for each $a_c \in A$ do
 3:     $maxprob \leftarrow 0$, $a_{p_0} \leftarrow root$
 4:     for each $a_p \in A$ do
 5:         Find $x \in \mathbf{x}$ for the ParentChild pair $(a_p, a_c)$
 6:         Obtain the probability $cprob$ that $x = true$
 7:         if $cprob > maxprob$ then
 8:             $maxprob \leftarrow cprob$, $a_{p_0} \leftarrow a_p$
 9:         end if
10:     end for
11:     $P \leftarrow P \cup \{(a_{p_0}, a_c)\}$
12:     $confidence \leftarrow confidence + log(maxprob)$
13: end for
```

of observed variables $\mathbf{x_r}$ with labels $\mathbf{l_r}$ translated from users' repairs (we assume $\mathbf{x_r}$ is empty if no repairs are observed), the *conditioning* algorithm yields a forest-structured model.

Second, we run a standard inference algorithm on this new model to obtain the assignment to all the variables. Because the model is now a forest-structured, a variety of existing algorithms, such as belief propagation, can perform exact inference on such a structure.

Finally, we treat the *global potentials* as a post-processing stage to ensure that the inferred variable assignment yields legal hierarchical trees for the input annotation regions. The goal of *global potentials* is to handle the *orientation* and *one-parent* constraints. Thus, we first enumerate all of the *ParentChild* candidates of each orientation, "upward" or "downward," and compute two separate annotation hierarchies with **EnforcedTreeInference**, seen in Algorithm 3. For all the *ParentChild* candidates with a given annotation as the child, the algorithm selects the one with the maximal probability (derived from the graphical model), thereby handling the *one-parent constraint*. We obtain two possible hierarchies, one "upward" and one "downward," each with computed *confidence*. We select the one with the higher confidence to handle the *orientation constraint*. Therefore, our algorithm yields legal annotation hierarchies.

3.4 Experiments

We can now quantify the performance of the system Senbazuruby evaluating its individual components. In particular, we present the performance of the frame finder and the hierarchy extractor. We do not directly evaluate the tuple builder because it entirely relies on the correctness of the hierarchy extractor, and it will yield the ideal results as long as it receives accurate hierarchies. We use the two spreadsheet corpora as mentioned in Section 3.2.2.

3.4.1 Data Frame Extraction

To evaluate the performance of the frame finder described in Section 3.3.2, we randomly sampled 100 data frame spreadsheets from each dataset. The 100 SAUS spreadsheets contained 6,878 non-empty rows, while the 100 WEB random spreadsheets contained 29,491 non-empty rows. A human expert labeled each row correctly. We randomly split the data into equal-sized training and testing sets, then evaluated frame finder's accuracy. We performed 10 random splits and averaged the results.

Table 3.4 shows good precision and recall for both SAUS and WEB. Perhaps not surprisingly, the SAUS set is slightly "easier" to process than the comparatively heterogeneous WEB corpus. It does relatively poorly at classifying header and title labels in WEB, which can be genuinely difficult even for a human. But as we will see, imperfect frame finder results have only a minor impact on downstream accuracy.

	SAUS		WEB	
	Precision	Recall	Precision	Recall
TITLE	0.983	0.979	0.768	0.735
HEADER	0.960	0.957	0.778	0.714
DATA	0.996	0.999	0.989	0.995
FOOTNOTE	0.970	0.978	0.858	0.821

Table 3.4: Precision and recall of the frame finder extractor.

Dataset		**Hierarchy Levels**			**# Left Annotations**		
		Min	Mean	Max	Min	Mean	Max
SAUS	R200	2	3.8	8	4	37.8	224
	health	2	3.6	6	12	34.5	76
	fin.	3	3.7	6	6	32.4	81
	trans.	3	4.0	8	5	36.1	73
WEB	R200	2	3.4	10	2	59.3	669
	bts	2	2.6	4	4	10.7	26
	nsf	2	4.0	7	9	83.9	331
	usda	2	3.2	4	5	34.5	56

Table 3.5: Basic statistics of our eight test sets.

3.4.2 Hierarchy Extraction

We now evaluate the performance of automatic extraction and interactive repair, and the quality of our *metadata resource*.

3.4.2.1 Experimental Setup

Our experiments are based on two spreadsheet corpora[6]: SAUS and WEB. From each of the two datasets, we randomly selected 200 *hierarchical spreadsheets*. We call these test sets SAUS **R200** and WEB **R200**. We constructed them by randomly sampling from SAUS or WEB and retaining only the hierarchical ones (*i.e.*, ones that have either hierarchical *left* or *top* annotations). In addition, we constructed a series of topic-specific test sets. For SAUS, we used government-provided category labels to identify spreadsheets for each of three topic areas: **health**, **finance**, and **transportation**; we chose 10 random hierarchical spreadsheets from each topic. For

[6]Downloadable:www.eecs.umich.edu/db/sheets/datasets.html

WEB, we used URL domain names as a rough proxy for the category label, choosing 10 random hierarchical spreadsheets from each of **bts.gov**, **usda.gov**, and **nsf.gov**. We asked a human expert to manually examine the above spreadsheets and create ground truth hierarchies. Details about the test sets are shown in Table 3.5.

We used the Python xlrd library to access data and formatting details of spreadsheet files. Our graphical model was implemented with UGM [106].

3.4.2.2 Automatic Extraction

In this section, we evaluate the performance of the automatic extraction phase. We evaluate the automatic extraction's accuracy in predicting correct *ParentChild* relationships by using standard metrics of Precision, Recall, and F1. We trained and tested automatic extraction using SAUS **R200** and WEB **R200**. We randomly split each of the two datasets equally for training and testing. We trained parameters on the training set and constructed one graphical model for the test set. We repeated the split-and-test process 10 times, computing average Precision, Recall and F1.

Automatic Models — A naive method AutoBasic to solve the hierarchy extraction problem is to use simple features (i.e. local alignment and indentation information) to classify two annotations as having a *ParentChild* relationship or not and assigns the most probable parent to each child.

We compared four different configurations of the automatic extraction graphical model with AutoBasic to demonstrate the power of each component of our automatic extractor: AutoLR uses node potentials only (with no edge or global potentials, the model is equivalent to the logistic regression, or LR, method)[7]. AutoEdge uses node potentials and edge potentials. AutoGlobal uses node potentials and global potentials. Finally, AutoFull uses all three potential types and reflects the entire contents of

[7]We also tried support vector machines and other non-joint-inference techniques, but they offered no significant gains over AutoLR.

Dataset	Methods	Precision	Recall	F1
SAUS	AutoBasic	0.4641	0.4641	0.4641
	AutoLR	0.8753	0.8750	0.8751
	AutoEdge	0.8801	0.8787	0.8794
	AutoGlobal	0.8834	0.8834	0.8834
	AutoFull	**0.8860**	**0.8860**	**0.8860**
WEB	AutoBasic	0.4736	0.4736	0.4736
	AutoLR	0.7886	0.7898	0.7892
	AutoEdge	0.7979	0.7968	0.7973
	AutoGlobal	0.8122	0.8122	0.8122
	AutoFull	**0.8327**	**0.8327**	**0.8327**

Table 3.6: Performance of the automatic extractor on SAUS and WEB **R200** datasets.

Section 3.3.3.4. [8]

Table 3.6 shows the performance of the five methods. We can see that all of our four graphical models significantly outperformed the baseline AutoBasic. Both partial models — AutoEdge and AutoGlobal — performed better than AutoLR, indicating that both *edge* and *global potentials* independently helped to improve the performance of automatic extraction. AutoFull, the model that includes all three potential types, is the best of all (though AutoFull's margin is small in the case of SAUS). We noticed that many extraction errors are due to contradictory spreadsheet formatting; designers of different spreadsheets may have conflicting designs, but even the format within one spreadsheet may not be consistent.

Training Data — We wanted to know if our supply of training data was limiting the automatic extractor's accuracy. We conducted a test in which we artificially constrained the training set size derived from SAUS **R200** and WEB **R200**, building a series of automatic extraction models with varying amounts of training data. Figure 3.11 shows the F1 of the *ParentChild* pairs for AutoFull as we change the size of the training set. The growth in both SAUS and WEB accuracy plateaus after a

[8]For AutoLR and AutoEdge we chose the probability threshold to maximize F1. For the rest two methods, there is no such flexibility, as the algorithms always select the parent with the maximum *ParentChild* probability for each child.

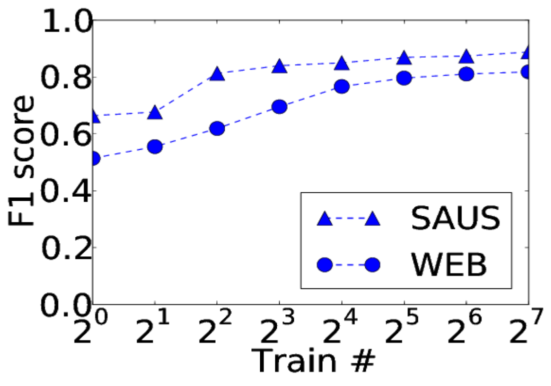


Figure 3.11: Performance for automatic extractor using different amounts of training data.

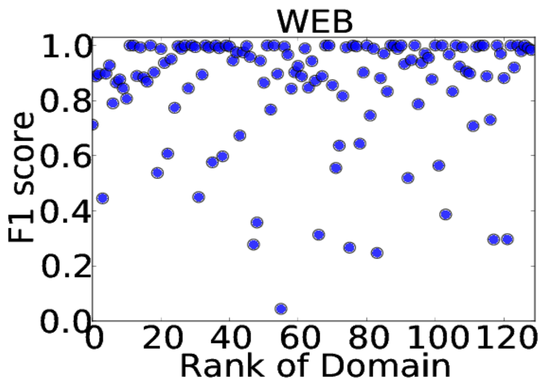


Figure 3.12: Performance for automatic extractor on different domains in WEB.

certain size. This analysis does not mean more training data cannot help, but does indicates that additional gains will likely be expensive.

Domain Sensitivity — We also examined whether the WEB automatic extractor's accuracy varies with the quality of the spreadsheet. It is difficult to precisely describe a spreadsheet's quality, so as a proxy we use the rank of the spreadsheet URL's Internet domain, when sorted in descending order of the number of spreadsheets hosted by the domain. Figure 3.12 shows the average F1 within each Internet domain's spreadsheets. We followed the same training and testing procedure as in the ***Automatic Models*** part above. The figure shows that the publisher's rank (or the quantity of spreadsheets it publishes) does not correlate with extraction performance. However we *did* find that spreadsheets from lower ranked domains are less likely to pass our initial "hierarchical data frame spreadsheet" filter.

In summary, our system shows substantially better performance than the baseline AutoBasic method, a 91% improvement in F1 on SAUS and a 76% improvement in F1 on WEB. We now turn to interactive repair to shrink the user's burden even further.

3.4.2.3 User Repair

We now evaluate the performance of the interactive repair phase. We use the eight datasets described in Section 3.4.2.1. For each **R200** of SAUS and WEB, we again randomly split the dataset into 100 training spreadsheets and 100 testing spreadsheets. We further randomly split the 100 testing spreadsheets into 10 subgroups with 10 spreadsheets in each, as **R10**; we then averaged the performance over the 10 subgroups. We created one model for each test set (health, finance, *etc*), except **R10**, where we created one model for each subgroup. Table 3.7 shows basic statistics for the interactive repair graphical models constructed for our test sets.

The metric of success for interactive repair is the amount of user work reduced when compared to simply fixing all the errors made by automatic extractor. We evaluate the amount of user effort by counting the required number of drag-and-drop repair operations to fix all the extraction errors in an annotation hierarchy, via our visual repair tool (seen in Figure 3.7). In the experiments, we simulated a user who randomly chooses extraction errors to repair, and who never makes a mistake. The user repairs errors until no errors remain. For each dataset, we ran this process 20 times and counted the average number of repairs performed. Notice that the maximum number of possible repair operations for a given hierarchy is the number of annotations in it.

For each result shown in Figure 3.13, Figures 3.14 and 3.15, we normalize the number of required repairs by the maximum possible number of repairs in that dataset (*i.e.*, the number of annotations). Thus, smaller bars are better, and results should be comparable across datasets.

Repair Models — A baseline method RepairBasic to incorporate interactive repair is to tie the *ParentChild* variables in one spreadsheet if the parents share the same formatting and so do the children: if a user changes one decision, the system automatically applies the change to the tied ones.

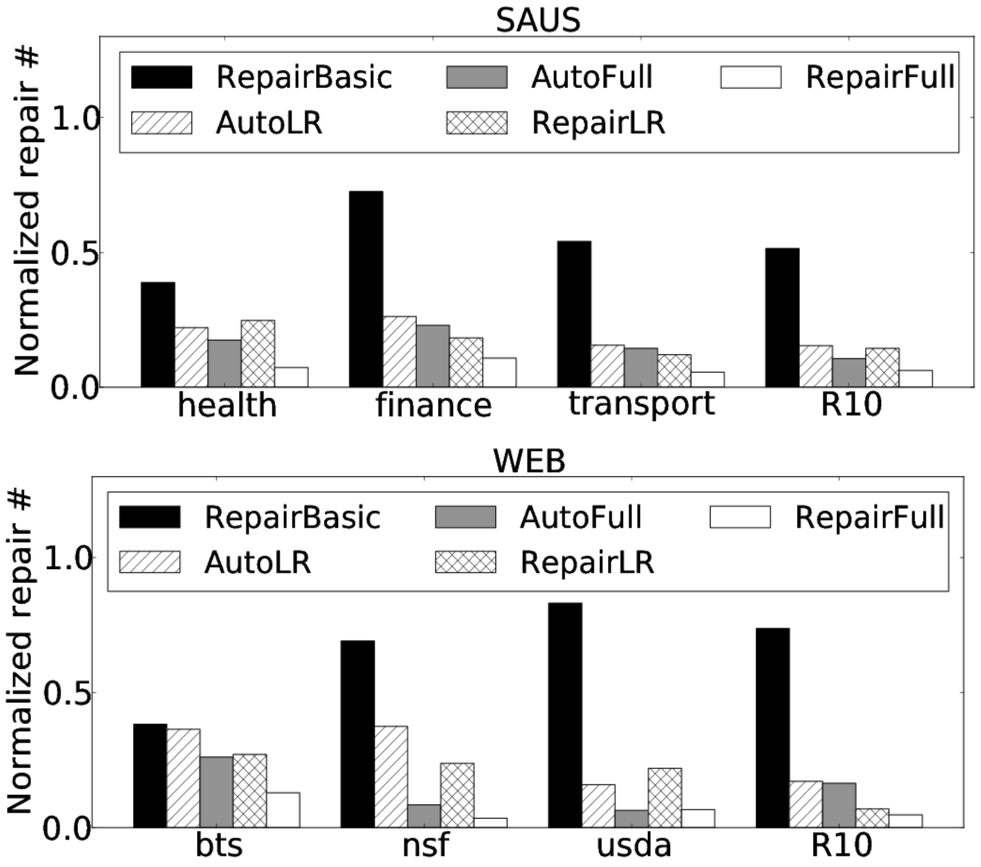


Figure 3.13: The normalized repair number for interactive repair on SAUS and WEB test sets.

		Sheet #	**Node** #	**Correlation Edge # (×1000)** Stylistic	Metadata	Total
SAUS	train	100	11269	87.5	115.7	177.6
	health	10	874	4.9	1.7	5
	fin.	10	1228	8.6	5.5	11.1
	trans.	10	1334	9.5	5.7	12.3
	R10	100	13866	144.4	43.3	161.3
WEB	train	100	31925	724.2	566.9	1069.0
	bts	10	249	0.5	0.0	0.5
	nsf	10	10698	265.1	22.9	283.3
	usda	10	1786	15.1	1.7	15.1
	R10	100	37386	1522.0	289.8	1677.6

Table 3.7: Basic statistics for each test set's interactive repair model.

We also evaluated six different versions of our extraction system. AutoLR and AutoFull are the automatic extractors described in the above section; we assume a user simply fixes all of their extraction errors one after another. RepairLR, RepairEdge, RepairGlobal and RepairFull are created by adding repair potentials to the previous four automatic extraction models. RepairFull is the full system.

Figure 3.13 shows the normalized number of repair operations of different interactive repair systems. RepairFull performed the best of all, requiring just *7.2%* of the maximum number of possible repairs when averaged over all test sets. In contrast, AutoFull (itself a dramatic improvement over the automatic extraction baseline) requires *15.4%* of the maximum; our exploitation of user repairs thus allows us to reduce the user burden by an additional 53%. AutoLR, an automatic extractor without joint inference, yields an even worse average of *23.3%*; we improve by 69%. The absolute number of user repairs is reasonable: RepairFull requires between 2 and 3.5 repairs per sheet for SAUS, and between 1.38 and 2.94 repairs per sheet for WEB.

Note that applying user repair information naively yields terrible results: RepairBasic requires *60.2%* of the maximum possible number of repairs, much worse than even AutoLR.

In all the datasets, RepairFull always improves or matches AutoFull, which indicates that our repair mechanism is genuinely beneficial to users; we managed to prevent backtracking and did not create more work for users. The same is *not* true for AutoLR *vs* RepairLR, which backtracks in the cases of SAUS/health and WEB/usda.

We further investigated interactive repair by considering different possible configurations of the interactive repair model on different test sets (shown in Figure 3.14). The Figure shows that both edge and global potentials are useful in reducing user burden, and using all of them helps the most.

Spreadsheet Grouping — We also investigated the influence of two spreadsheet grouping methods on interactive repair performance. (1) ***By topic***: We group spreadsheets according to their human-given topic labels (such as finance and health) or their URL hostnames (such as bts.gov and nsf.gov); and (2) ***By Jaccard similarity***: We

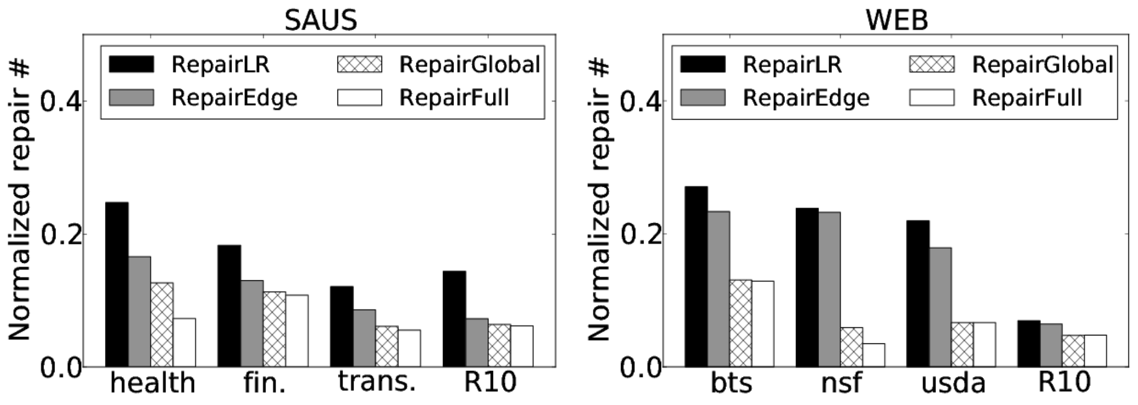


Figure 3.14: The normalized repair number for four interactive repair configurations.

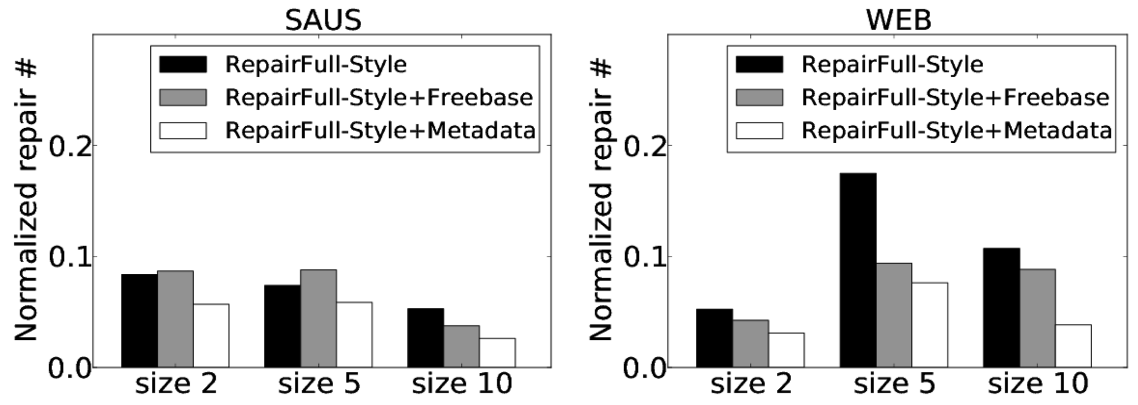


Figure 3.15: The normalized repair number required by different configurations of metadata links.

compute the clusters by creating a graph in which each spreadsheet is a node, and edges exist when two spreadsheets have Jaccard similarity (computed over the non-numeric strings from each spreadsheet) greater than a threshold of 0.6. We find all weakly connected components in the graph as the spreadsheet groups. Note that grouping spreadsheets should only impact metadata affinity, as metadata affinity is the only way to connect *ParentChild* decisions across spreadsheets.

For both SAUS and WEB, we ran each grouping technique, then randomly selected 3 groups of size 2, 3 groups of size 5, and 3 groups of size 10. For each group, we first built one RepairFull on this group of spreadsheets and computed the number of repairs required to eliminate all the extraction errors. We then compared against the sum of repairs needed by RepairFull when running on each spreadsheet of the group in isolation. We found that grouping by topic only reduces repairs up to 5.8% on SAUS and 2.1% on WEB, while grouping by Jaccard similarity reduces repairs by up to 64.0% in SAUS and 84.6% in WEB.

Thus, Jaccard similarity grouping yields a massive reduction in necessary user repairs when compared to topic grouping. We did not present these results in Figures 3.13 and 3.14 because we believe that highly coherent clusterings will only be possible in certain situations. Shared spreadsheet templates is one such situation; another is when the metadata resource is of especially high quality (perhaps even curated by hand), allowing interactive repair to find otherwise invisible connections among independent spreadsheets.

Metadata Resources – The quality of our metadata resource clearly impacts metadata affinity. Figure 3.15 shows the normalized number of repairs required by different metadata resource configurations of RepairFull, when run on Jaccard-clustered spreadsheets mentioned above in ***Spreadsheet Grouping***. We compared the approach based on our metadata resource (RepairFull-Metadata) against a no-metadata technique (RepairFull-Style) and a technique that uses Freebase to discover metadata affinity (RepairFull-Freebase). (In that last case, two annotations have metadata affinity if they share the same Freebase topic.) The figure shows that in all cases, our RepairFull-Metadata technique performs the best, usually followed by RepairFull-Freebase. On average, our induced metadata resource reduces user effort by 34.4% when compared to the Freebase resource. Note that some of the spreadsheets we process are on extremely technical topics (such as currency trading, health care, and minerals processing) that are unlikely to be captured in a general-purpose metadata resource such as Freebase.

Runtime Performance — After each user repair operation, users have to wait for the model to recompute the new result. In our experiments, each repair's inference took 0.7s on **R10** in SAUS and 3.2s on **R10** in WEB on average. All other test datasets took less than 0.7s, except for **nsf** (at 4.7s). The results indicate that interactive repair is computationally feasible, at least for relatively small datasets.

Overall, we have demonstrated that our RepairFull extraction system can extract accurate spreadsheet hierarchies using just 7.2% of the maximum possible user effort, a reduction of 53% compared to AutoFull, our automatic extraction system (itself a significant improvement over previous automatic extraction techniques). These numbers apply to real-world datasets; in certain cases where spreadsheets share a large amount of metadata, we can improve the factor even further. Moreover, our system works well on domain-specific datasets with no explicit user-provided metadata.

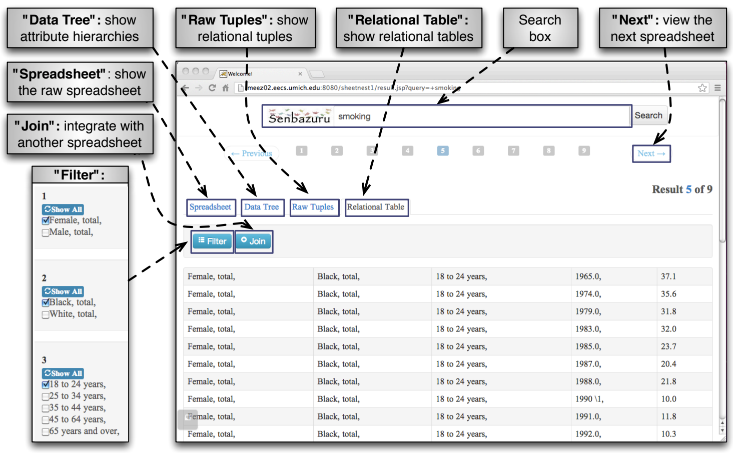
3.5 System Demonstration

In this Section, we demonstrate that Senbazuru, a prototype spreadsheet database management system (SSDBMS), is able to extract relational information from a large number of spreadsheets; doing so opens up opportunities for data integration among spreadsheets and with other relational sources.

3.5.1 User Interface

We have a working prototype of Senbazuru, available as a desktop web application and also as an iPad application. The two client interfaces for Senbazuru are shown in Figure 3.16. They allow users to *search* for data, to view and edit the *extract* results, and to use *query* operations.

Search – As with other search-and-rank tools, a user types keywords in the search box, and obtains a list of relevant spreadsheets. She can then browse results and select the most relevant one. For example, in Figure 3.16 (a), a user enters "smoking" as the search query. She can examine the top hit's raw spreadsheet by



Female, total,	Black, total,	18 to 24 years,	1965.0,	37.1
Female, total,	Black, total,	18 to 24 years,	1974.0,	35.6
Female, total,	Black, total,	18 to 24 years,	1979.0,	31.8
Female, total,	Black, total,	18 to 24 years,	1983.0,	32.0
Female, total,	Black, total,	18 to 24 years,	1985.0,	23.7
Female, total,	Black, total,	18 to 24 years,	1987.0,	20.4
Female, total,	Black, total,	18 to 24 years,	1988.0,	21.8
Female, total,	Black, total,	18 to 24 years,	1990 \1,	10.0
Female, total,	Black, total,	18 to 24 years,	1991.0,	11.8
Female, total,	Black, total,	18 to 24 years,	1992.0,	10.3

(a)

(b)

Figure 3.16: Screenshots of two Senbazuru clients, as a desktop application (a) and an iPad application (b).

clicking "Spreadsheet" or check other relevant spreadsheets by clicking "Next."

Extract – After selecting the most relevant spreadsheet, a user can use *extract* to transform the spreadsheet data into a relational table. She can review the extracted hierarchical metadata by clicking "Data Tree." Figure 3.7 shows the interface for viewing the extracted hierarchies. To repair any extraction errors, she can drag and move a node of the tree from one place to another. After observing the user's repair action, Senbazuru will automatically re-run *extract* and display a new tree. For example, as shown on the left of Figure 3.7, a user performs a repair by clicking and dragging White, total so that it becomes a child of Male, total. Meanwhile, the *extract* component automatically discovers that Black, total should also be moved. Thus, the user's one single repair action can trigger multiple fixes, yielding the hierarchy shown on the right. After repairing extraction errors, the user can review the generated relations by clicking "Relational Table," as shown in Figure 3.16(a).

Query – Users can perform *query* operations on the extracted relational table. They are not required to write SQL statements and can apply *select* and *join* via the interface:

1. Select – The *select* feature, also called *filter*, is similar to executing a selection query on the recovered relation. Clicking "Filter," a user can use a faceted-search interface to specify the filter conditions. This interface is automatically composed by Senbazuru. For example, Figure 3.16 (a) shows that the user limits the displayed data to smoking statistics for people who are Female, Black, and 18 to 24 years old.

2. Join – The *join* feature allows users to integrate two arbitrary spreadsheet-derived relations. The user starts by applying the *search* and *extract* features as described above. Once the user has found a good result, she clicks "Join" and enters a second text query. The query yields a second ranked result list. She chooses a relevant join target from this ranked list and obtains a correctly extracted relation. She indicates which columns from each dataset should be used as an equi-join key.

For example, Figure 3.16 (b) shows a screenshot of the *join* process on our iPad client. When the user touches a column, Senbazuru highlights it in bright yellow, as shown on the left of the figure. Meanwhile, Senbazuru faintly highlights a column on the right-hand table to indicate a possible join. The user can drag the column, highlighted in bright yellow, from the left-hand relation to the right and release it. This action indicates the join key; Senbazuru executes the appropriate join and shows the user the resulting brand-new relation.

3.5.2 A Walk-through Example

We will demonstrate Senbazuru's workflow through Fred's example, as follows:

Policy expert Fred wants to see whether the strength of the connection between smoking and lung cancer is consistent across all U.S. states. Fred does not have the relevant data at hand, but assumes it is "out there" on the web somewhere.

Using Senbazuru, the policy expert Fred can obtain the requested data through the following steps:

1. Search for "smoking by state."

2. Browse the returned spreadsheets and select the most relevant one.
3. Click "Data Tree" and check the correctness of the extracted hierarchies.
4. Repair the hierarchies if any extraction errors exist.
5. Click "Data Table" to review the extracted relation.
6. Repeat the process from step 1 to get the relational table for the most relevant lung cancer spreadsheet by querying "lung cancer by state."
7. Specify join columns to create a new table.
8. Review the result, using the faceted *select* interface.

In addition to Fred's example, VLDB attendees are welcome to try many other interesting queries, such as "employment statistics 2010" and "Michigan GDP."

3.6 Related Work

Spreadsheet Management – Existing approaches for transforming spreadsheet data into databases fall into a few broad categories. First, *rule-based* approaches [5, 57, 63, 76] often require users to learn a newly defined language to describe the transformation process. The approaches are flexible but often require explicit conversion rules that are difficult and time-consuming for the user to compose. Second, there is a range of *visualization* systems [99] that help the user navigate and understand spreadsheets with visualization techniques, but the mechanisms are not able to extract relational data from spreadsheets. Finally, *automated* approaches are the most similar to ours. Abraham and Erwig [2] attempted to recover spreadsheet tuples, and Cunha *et al.* [42] primarily focused on the problem of data normalization. But their work assumes a simple type of spreadsheets and they did not address the hierarchical structures that are key to understanding a huge portion of the online spreadsheet data.

Tabular Data Extraction – There has been a large amount of work centered on extracting tabular data on the web [23, 24, 54]. Most of these projects have focused on the details of identifying data-centric tables or on applications that can be built on top of them. HTML tables likely contain hierarchical-style data examples, but we are not aware of any research to date focused on this problem.

Programming By Demonstration – The interactive repair component of our work is part of an intellectual thread that ties programming by demonstration [56, 65, 69, 104], mixed-initiative systems [62], and incorporation of user feedback into extraction systems [27]. Many of these systems are driven by a programming language that the user must learn; our system does not require the user to learn a language, just to use a "drag-and-release" interface. Our solution's design, which alternates automatic and user effort, is similar in spirit to Wrangler [56, 69] and mixed initiative systems [62, 64, 65, 104]. However, Wrangler-style techniques cannot be applied to our situation directly, as they generally process data with standard textual cues that are often missing from real-world spreadsheets.

3.7 Conclusion and Future Work

We have described the Senbazuru system, a prototype spreadsheet database management system (SSDBMS). Senbazuru supports three functional components, *search*, *extract* and *query*. The *search* component allows a user to quickly *search* relevant datasets in a huge web spreadsheet corpus; The *extract* component is composed of a background ***extraction pipeline*** that automatically obtains relational data from spreadsheets, and a ***repair interface*** that allows users to manually repair extraction errors. Moreover, this component automatically exploits commonalities among errors to probabilistically reapply one user fix to other similar mistakes, thereby minimizing explicit manual intervention. The *query* component supports basic relational operators, especially selection and join, which the user can apply to spreadsheet-derived relations. As a result, Senbazuru opens up opportunities to use many relational data management tools to conduct data analysis on spreadsheets, especially those with complicated hierarchical metadata structures.

CHAPTER IV

Anthias: Extraction on Spreadsheet Properties

4.1 Problem Overview

In Chapter 3, we have presented Senbazuru, a prototype spreadsheet database management system that is able to extract relational information from a large number of *data frame* spreadsheets. But to handle a large variety of spreadsheets, we have to correctly identify various *spreadsheet properties*, which correspond to a series of transformation programs that contribute toward the spreadsheet-to-relational table transformation framework.

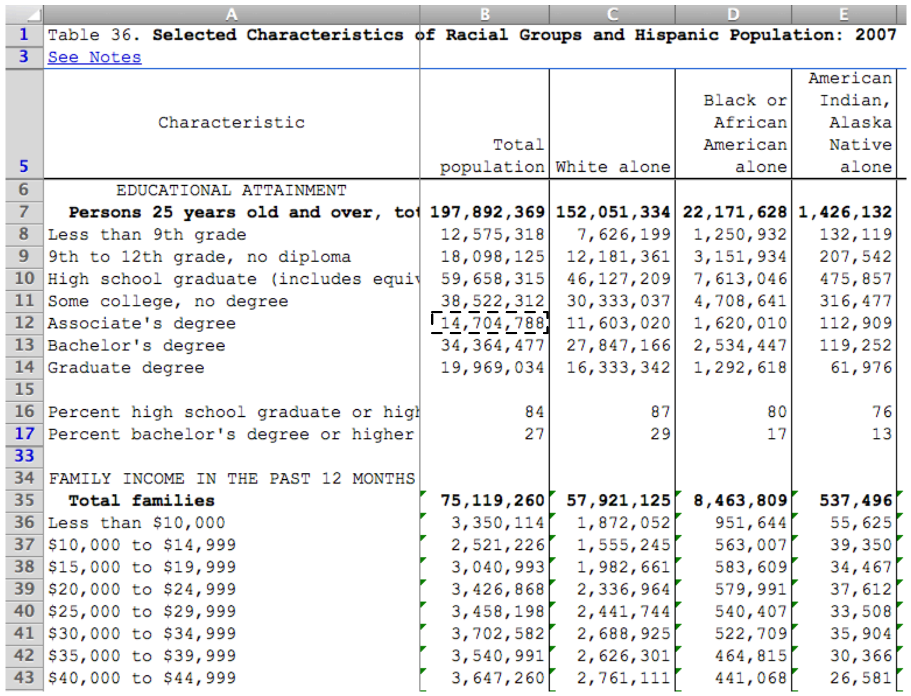
As we discussed in Section 1.3.2, each of the transformation programs describes a piece of process on a specific characteristic of a sheet table that yields a result that is closer to a relational table. We use a *spreadsheet property* (*e.g.*aggregation rows, aggregation columns, crosstab, or split tables) to represent such a transformation program that contributes to the sheet table-to-relational table transformation.

In this chapter, we study the task of *spreadsheet property detection*, which decides whether a spreadsheet contains a specific spreadsheet property (*e.g.*whether a spreadsheet contains "aggregation rows".) The spreadsheet property detection task is the first step toward building this spreadsheet-to-relational table transformation pipeline.

We propose a novel rule-assisted active learning framework to construct high-quality spreadsheet property detectors with little user labeling effort. In the initial stage, the user does not know what the perfect property detectors are, but is often able to write crude heuristic rules to describe their intuition. For example, the user can simply write: If a spreadsheet contains a row with formulas, then it has the property "aggregation row". This simple rule may perform reasonably well, but to improve or maintain rule-based classifiers would require intensive amount of work. On the other hand, we can apply an active learning approach to save training data. During each iteration we select the instance that is the closest to the decision boundary for the user to label. But the active learning approach often suffers from the cold-start problem, and especially in the initial stage it lacks training data to approach the ideal decision boundary.

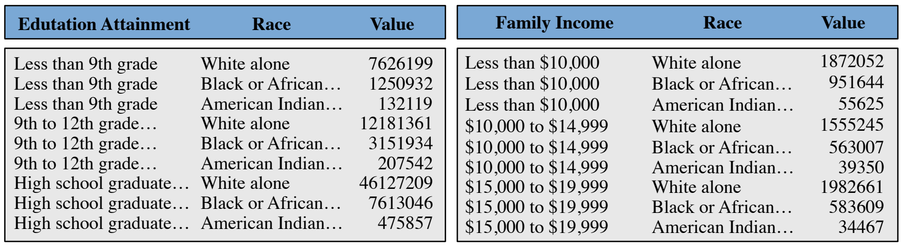
Our hybrid framework is designed to integrate crude user-provided rules with an active learning approach to reduce the user's labeling effort. In addition to the labeled instance suggested by the active learning approach, we bring in crude rules from the user to generate additional labeled data for the property detectors. We produce labeled instances with the agreed decision from both the current trained classifier and the user-provided rules. This bagging-like technique makes it possible for our framework to tolerate *bad* rules. Our hope is that this hybrid approach can generate additional high-quality labeled data especially in the initial stage to warm up the classifiers quickly.

In this chapter, we present Anthias, which is an extension of Senbazuru in order to convert a broader range of spreadsheets. We introduce the data sources, the concept of spreadsheet properties with examples in Section 4.2. we present our hybrid active learning framework for the spreadsheet property detection in Section 4.3. we conduct an extensive study on property detection algorithms and a large scale web spreadsheet study in Section 4.4. Finally we cover related work in Section 4.5, and finally conclude with future work in Section 4.6.



	A	B	C	D	E
1	Table 36. **Selected Characteristics of Racial Groups and Hispanic Population: 2007**				
3	See Notes				
5	Characteristic	Total population	White alone	Black or African American alone	American Indian, Alaska Native alone
6	EDUCATIONAL ATTAINMENT				
7	**Persons 25 years old and over, tot**	**197,892,369**	**152,051,334**	**22,171,628**	**1,426,132**
8	Less than 9th grade	12,575,318	7,626,199	1,250,932	132,119
9	9th to 12th grade, no diploma	18,098,125	12,181,361	3,151,934	207,542
10	High school graduate (includes equiv	59,658,315	46,127,209	7,613,046	475,857
11	Some college, no degree	38,522,312	30,333,037	4,708,641	316,477
12	Associate's degree	14,704,788	11,603,020	1,620,010	112,909
13	Bachelor's degree	34,364,477	27,847,166	2,534,447	119,252
14	Graduate degree	19,969,034	16,333,342	1,292,618	61,976
15					
16	Percent high school graduate or high	84	87	80	76
17	Percent bachelor's degree or higher	27	29	17	13
33					
34	FAMILY INCOME IN THE PAST 12 MONTHS				
35	**Total families**	**75,119,260**	**57,921,125**	**8,463,809**	**537,496**
36	Less than $10,000	3,350,114	1,872,052	951,644	55,625
37	$10,000 to $14,999	2,521,226	1,555,245	563,007	39,350
38	$15,000 to $19,999	3,040,993	1,982,661	583,609	34,467
39	$20,000 to $24,999	3,426,868	2,336,964	579,991	37,612
40	$25,000 to $29,999	3,458,198	2,441,744	540,407	33,508
41	$30,000 to $34,999	3,702,582	2,688,925	522,709	35,904
42	$35,000 to $39,999	3,540,991	2,626,301	464,815	30,366
43	$40,000 to $44,999	3,647,260	2,761,111	441,068	26,581

Figure 4.1: A spreadsheet about population statistics, from the Statistical Abstract of the United States.



Edutation Attainment	Race	Value
Less than 9th grade	White alone	7626199
Less than 9th grade	Black or African…	1250932
Less than 9th grade	American Indian…	132119
9th to 12th grade…	White alone	12181361
9th to 12th grade…	Black or African…	3151934
9th to 12th grade…	American Indian…	207542
High school graduate…	White alone	46127209
High school graduate…	Black or African…	7613046
High school graduate…	American Indian…	475857

Family Income	Race	Value
Less than $10,000	White alone	1872052
Less than $10,000	Black or African…	951644
Less than $10,000	American Indian…	55625
$10,000 to $14,999	White alone	1555245
$10,000 to $14,999	Black or African…	563007
$10,000 to $14,999	American Indian…	39350
$15,000 to $19,999	White alone	1982661
$15,000 to $19,999	Black or African…	583609
$15,000 to $19,999	American Indian…	34467

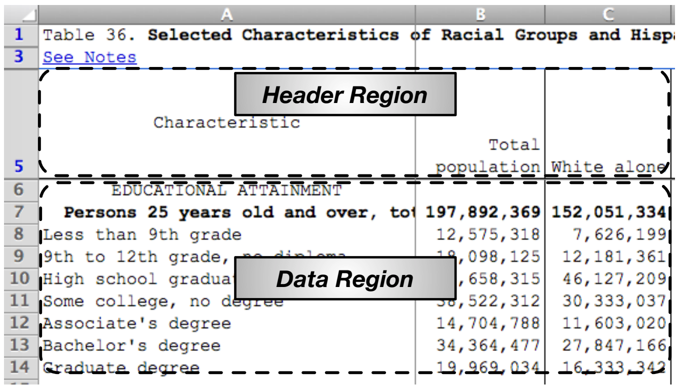
Figure 4.2: The ideal relational tables for the spreadsheet example shown in Figure 4.1.

4.2 Preliminary

In its most generic incarnation, a spreadsheet is simply an $M \times N$ grid of cells, in which each cell can contain a string, a number, or nothing.

Our ultimate goal is to build a spreadsheet-to-relational table transformation framework that takes in sheet tables and outputs relational tables. We now formally describe sheet tables and relational tables, as follows.

Input: Sheet table — We consider a typical portion of a spreadsheet that is



	A	B	C
1	Table 36. **Selected Characteristics of Racial Groups and Hisp**		
3	See Notes		
5	Characteristic	Total population	White alone
6	EDUCATIONAL ATTAINMENT		
7	**Persons 25 years old and over, tot**	**197,892,369**	**152,051,334**
8	Less than 9th grade	12,575,318	7,626,199
9	9th to 12th grade, [illegible]	[illegible],098,125	12,181,361
10	High school gradua[illegible]	[illegible],658,315	46,127,209
11	Some college, no degree	[illegible],522,312	30,333,037
12	Associate's degree	14,704,788	11,603,020
13	Bachelor's degree	34,364,477	27,847,166
14	Graduate degree	19,969,034	16,333,342

Figure 4.3: A spreadsheet's header and data region.

able to be converted into relational tables, and we call it a *sheet table*. A sheet table consists two regions: A header region and a data region, as shown in Figure 4.3. We previously addressed the problem of finding the header and data regions using a linear chain CRF to assign one of the four labels (header, data, title or footnote) to each row in a spreadsheet in Section 3.3.2. Using this CRF mechanism, we recognize each sheet table with a header and data region from a raw input spreadsheet, then use a sheet table as input to our transformation framework.

Output: High-quality relational table — We aim to obtain a high-quality relational table for each sheet table.

Our definition of high-quality has two important characteristics. First, the values of one column in the relational table should be *homogeneous* or belong to the same semantic class. For example, the sheet table shown in Figure 4.1 is not an appropriate relational table, because "Education Attainment" values (*e.g.*"Bachelor's degree") and "Family Income" values (*e.g.*, "less than $10,000") are mixed together in column A. In addition, we want the resulting relational table to be as compact as possible. For example, we remove all the aggregation values in the column B as shown in Figure 4.1.

4.2.1 Data Sources

In this paper, we rely on two spreadsheet data sources: The *WebCrawl* dataset is a large-scale corpus of web-crawled spreadsheets, and the *Web400* dataset is a hand-labeled subset of WebCrawl. We now introduce the two datasets.

WebCrawl data — The WebCrawl dataset is our large-scale web-crawled spreadsheet corpus. It consists of 410,554 Microsoft Excel workbook files with 1,181,530 sheets from 51,252 distinct Internet domains (a workbook file may contain multiple sheets). We found the spreadsheets by looking for Excel-style file endings among the roughly 10 billion URLs in the ClueWeb09 web crawl[1].

Web400 data — The Web400 dataset is a 400 labeled sample from the WebCrawl corpus. We want to avoid sampling too many spreadsheets from one HTTP domain because there are a few domains covering the majority of the web spreadsheets as mentioned in Section 3.2.3. Thus, we obtained this Web400 data via the following procedure. We first grouped spreadsheets by their HTTP domain, and removed the long-tail spreadsheets (*i.e.*, those from HTTP domains containing less than 20 spreadsheets), yielding 2,579 domains with 284,396 sheets in total. Then we selected 20 random domains from the 2,579 domains; from each domain, we again randomly sample 20 sheets, yielding 400 sheets in total as the Web400 dataset. We manually assign correct spreadsheet properties to each Web400 sheet for further evaluation.[2]

4.2.2 Spreadsheet Properties

We use spreadsheet *properties* to reflect the sheet tables to relational tables transformation process.

We propose a list of standard property/transformation pairs. When a *property* exists in a sheet table, applying the corresponding *transformation* operation will yield

[1]`http://lemurproject.org/clueweb09.php`

[2]Notice that if a workbook contains multiple sheets, we select a random non-empty sheet from it for labeling; and if there are multiple sheet tables in a sheet we only consider the first one.

Property	**Transformation Operation**
aggregation row	delete
aggregation column	delete
hierarchical data	unfold [88]
hierarchical header	unfold [88]
crosstab	pivot [6]

Table 4.1: Spreadsheet properties and transformation.

a result that is closer to a relational table. If we can detect all of the appropriate properties in a candidate sheet table, then applying the corresponding transformation operations should yield a valid relational output.

As shown in Table 4.1, we define five spreadsheet properties with corresponding transformation operations (we explain the detail of the five properties in Section 4.2.3). For example, "aggregation row" in Table 4.1 is an appropriate spreadsheet property, because this property makes a sheet table an invalid high-quality relational table. Also we can define a corresponding transformation operation "deletion" which removes all data values in the aggregation row, in order to eliminate this invalidness. Conversely, "the number of rows in a sheet table" is not an appropriate spreadsheet property, because the number of rows does not affect whether a sheet table is a valid, high-quality relational table.

We focus on the problem of *detecting which properties a sheet table contains.* This is the first step toward building the spreadsheet-to-relational table transformation framework. To build such a framework, we first have to define the transformation operations for each property, and we simply borrow the transformation operations from systems such as Wrangler [70] or Potter's Wheel [88]. While these systems focus on designing the transformation operation language, we attempt to automatically construct the transformation programs using their language. In addition, we have to extract the detail for each property. For example, knowing that a spreadsheet has the property "aggregation rows", we still need an extraction program to recognize all the aggregation rows for the transformation process. We also attempted to extract

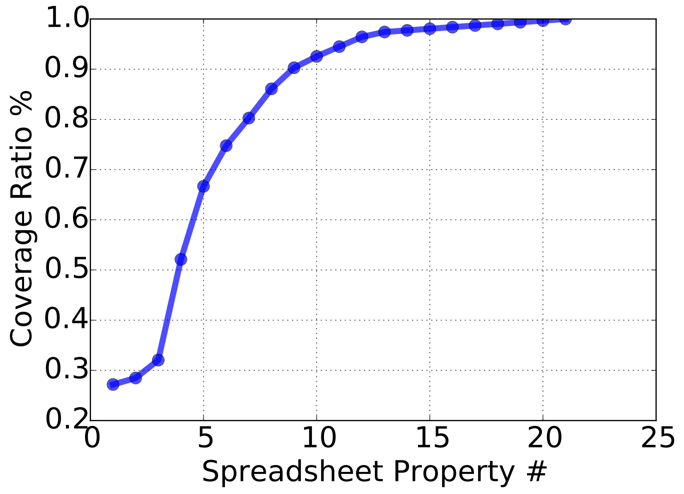


Figure 4.4: Coverage ratio for spreadsheet properties on the Web400 dataset.

some spreadsheet properties, such as hierarchical data and hierarchical header in Section 3.3.3. In this chapter, we do not focus on the extraction programs.

4.2.3 Property Examples

We investigated the spreadsheet properties in the Web400 dataset and made the following observations.

Among the 400 spreadsheets in the Web400 dataset, we found 309 spreadsheets containing sheet tables, and the rest included unfilled forms, text, visualizations and so on. As shown in Figure 4.4 we observe that the five properties shown in Table 4.1 cover 68% (209/309) of the spreadsheets.

Figure 4.4 shows there are 21 spreadsheet properties that cover the transformation process from sheet tables to relational tables for the 400 spreadsheets in Web400 data. The spreadsheet properties also include "split table" (rows 6-17 and rows 34-43 should be in two separate relational tables in Figure 4.1), "rows of different units" (the data values in row 8 is the absolute population number and in row 16 is the percentage in Figure 4.1) and so on, but in this paper, we focus on these top five spreadsheet properties as follows:

1. Aggregation Rows (agg_row) — An aggregation cell is defined as an aggregation function (*e.g.*sum, avg, min, max, *etc*) over a group of cells. An aggregation cell is often indicated by explicit spreadsheet formulas, but sometimes the formula is implicit (the value may be copied from other places). A spreadsheet has the property "agg_row" if it has a row of aggregation cells. For example, the spreadsheet in Figure 4.1 has the property "agg_row" because all the numeric values in row 16 are calculated on the rows 7-14.

2. Aggregation Columns (agg_col) — A spreadsheet has the property "agg_col" if it has a column of aggregation cells. For example, the spreadsheet in Figure 4.1 has the property "agg_col" because column B is an aggregation column.

3. Hierarchical Data (hier_data) — A spreadsheet has the property "hier_data" if there exists a cell in the data region implicitly describing other cells. For example, the sheet in Figure 4.1 has the property "hier_data" because "education attainment" in row 6 implicitly describes rows 7-17.

4. Hierarchical Header (hier_head) — A spreadsheet has the property "hier_head" if there exists a cell in the header region implicitly describing another column. For example, the spreadsheet in Figure 4.1 does not have the property "hier_head" because each cell in the header only describes its own column.

5. Crosstab — A spreadsheet has the property "crosstab" if all of its numeric values can be converted into one column with a new dimension for the associated metadata. For example, the spreadsheet in Figure 4.1 has the property "crosstab" because the numeric values in B-E can be converted into one column with a new dimension "Race".

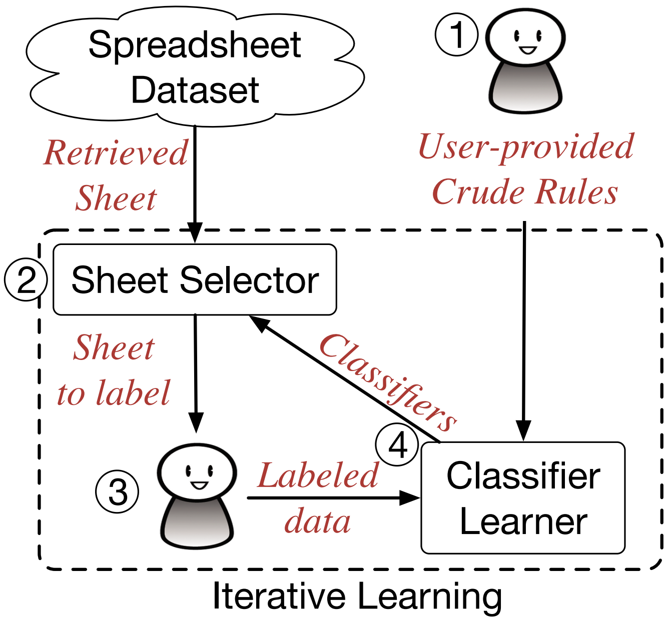


Figure 4.5: The hybrid iterative learning framework for spreadsheet property detection.

4.3 Spreadsheet Property Extraction

The property detection task is to build a binary classifier for a spreadsheet property, and we now formally define the property detection task.

Let $Q = \{q_1, ..., q_k\}$ be a set of spreadsheet properties. The ***property detector*** builds a set of binary classifiers: One classifier θ_q for each $q \in Q$, and the classifier θ_q determines whether a sheet table has the property q or not. As a result, given a sheet table x, the property detector generates a subset of properties $\mathbf{q} = \{q\}$ and $\mathbf{q} \subseteq Q$. It represents that x contains and only contains the set of properties $\mathbf{q}$.

4.3.1 The Iterative Learning Framework

We propose a hybrid iterative learning framework to build spreadsheet property detectors. Our framework incorporates crude user-provided rules to the iterative learning process to reduce user effort.

Figure 4.5 shows our hybrid iterative learning framework for spreadsheet property detection. In the initial stage, a user provides the crude heuristic rules (we discuss the user-provided rules in detail in Section 4.3.2.2). During the interactive learning stage, the sheet selector selects a spreadsheet from the dataset, and sends it to the user. The user is responsible for labeling the spreadsheet with all the spreadsheet properties it contains. The classifier learner then accumulates all user labeled spreadsheets together with automatically generated labels using the user-provided rules, to train a classifier for each spreadsheet property. The user iteratively labels a spreadsheet selected by the sheet selector and the classifier learner produces newly trained classifiers for each iteration. In the end, we obtain the most newly trained classifiers from the classifier learner as the output spreadsheet property detectors.

Note that we do not focus on the classification algorithms, and the classifier learner simply employs an existing classification algorithm (logistic regression) to train a classifier for each spreadsheet property. Also, the training data might be imbalanced during the learning process, and we simply duplicate the minority instances until their size is comparable to the size of the majority class [59].

We now describe the user work and highlight the critical techniques of the framework.

4.3.2 User Work

In this section, we introduce the required user work to construct the spreadsheet property detectors and then propose two ways to save user labeling effort.

4.3.2.1 Construct Property Detectors

To construct the property detectors requires user to provide the following three types of data:

1. Features f(x): We generate features $f(x)$ for each sheet table x, and they represent the important signals derived from x to help determine whether x contains a property or not. For example, if a sheet table's data region contains the keyword "total", it is very likely to have the property "agg_row". The significant features might be different for different spreadsheet properties or in different datasets. For simplicity, we use $f(x)$ to represent the universe of the features, and the features we used are as follows:

- whether a cell in the header/data region contains keywords (*i.e.*, "total", "sum", "avg", "average", "median", "mean", "totals", "summary", "subtotal").
- the standard deviation of the header string length.
- the average/maximum p-value for the t-test for data values in two numeric columns.
- the maximum/minimum ratio of formula cells to numeric cells in a data row/column.
- whether a column in data region has different formatting styles, and we test each style as shown in Table 4.2.
- the data/header region has a merged cell.
- there exists two cells in header region, one has a higher column but lower row index than the other.
- sheet tableis empty.
- there is no header/data region.
- the ratio of numeric cells to total cells in sheet table.
- the ratio of non-zero cells to total/numeric cells in sheet table.
- the maximum ratio of non-zero cells to numeric cells in data rows/columns.
- the ratio of numeric rows/columns to all data rows/columns.

#	**Features**
1	a cell contains colon
2	a cell is capitalized
3	a cell's alignment
4	a cell is bold
5	a cell's indentations
6	a cell is italic
7	a cell's height
8	a cell is underlined

Table 4.2: The formatting style of a spreadsheet cell.

- the absolute number of numeric data rows/columns. We create the boolean feature vector by testing whether the absolute number is greater than 1 to 10 by 1.

2. Property Set (Q): It is hard to construct the complete spreadsheet property set Q because there are always unknown properties. Instead, we define a few properties that we are aware of (*i.e.*the five properties in Section 4.2.3). At the same time, we allow new properties to be discovered during the labeling stage and we discuss the details below.

3. Training Data (D = {(x, **q**)}): Given a sheet table x, a user has to determine the properties **q** that x contains. During the labeling process, the user evaluates the transformation process for converting a sheet table x to high-quality relational tables, and decides whether x contains the predefined spreadsheet properties or new properties.

To be more specific, a user first labels a sheet table x for the *predefined properties.* It is straightforward to decide whether a spreadsheet x contains a well-defined property or not. In addition, the user is also responsible for *discovering new properties* via the following procedure: After finding the predefined properties **q** in the given sheet table x, the user attempts to convert the sheet table to relational tables using the transformation operations defined by **q** and determines whether the conversion

Property	Crude User-provided Rules
agg_row	If the data region contains the keyword “total” or has a row with embedded formulas, then true; otherwise false.
agg_col	If the header region contains the keyword “total” or has a column with embedded formulas, then true; otherwise false.
hier_data	If the data region has different formatting styles (*e.g.*, alignment, bold, indentation, and italic), then true; otherwise false.
hier_head	If the header region contains merged cells, then true; otherwise false.
crosstab	If the variance of the string length in the header region is < 0.5, then true; otherwise false.

Table 4.3: Examples of the crude user-provided rules for the five properties in Section 4.2.3.

was successful. If it was, then no new property is necessary; otherwise, the user has to define one or more new spreadsheet properties with corresponding transformation operations, and then add the new properties to $\mathbf{q}$.

For example, for the sheet table shown in Figure 4.1, we first determine whether it contains the five properties defined in Section 4.2.3. We recognize it has the properties “agg_row”, “agg_col”, “hier_data” and “crosstab”. We then use the corresponding transformation operations defined in Table 4.1 to convert this sheet table to relational tables. In this case, we cannot obtain a valid high-quality relational table: We still need to split rows 6-17 (about “Education Attainment”) from rows 34-43 (about “Family Income”) into two separate relational tables. Then the user defines a new spreadsheet property “split table” with the corresponding transformation operation “split”. Thus the sheet table also contains the property “split table”.

In summary, it requires a huge amount of user effort to construct a binary classifier for each spreadsheet property. A user is responsible for a variety of things, including features $f(x)$, property set Q and training data $D = \{(x, \mathbf{q})\}$.

4.3.2.2 Two Ways to Reduce User Effort

In this paper, we focus on reducing the required user effort on *training data* $D = \{(x, \mathbf{q})\}$. We observe two ways to reduce the required training data:

Label Uncertain Spreadsheets — Labeling uncertain instances is essentially an active learning approach [97]. Active learning studies the problem of how to select training data in order to save the required labeled data. One typical way of selecting training data for binary classifiers is ***uncertainty sampling***. Uncertainty sampling often chooses to label instances that are closest to the decision boundary, and it refines the decision boundaries by heavily exploiting the current knowledge space. Often it simply selects the instance with the predicted probability closest to 0.5 [94].[3]

But the active learning approach often lacks training data to approach the ideal decision boundary in the beginning; this is known as the "cold-start" problem.

Crude User-provided Rules — Before labeling any spreadsheet, we bring in a user's intuition on building property detectors by asking for crude and easy-to-write rules. For example, a user can write the simple rules for the property "agg_row": If a spreadsheet contains a row with formulas, it has the property "agg_row". Similarly, we can write rules for all the properties mentioned in Section 4.2.3, as shown in Table 4.3. We ask users to write a piece of program to represent the rules. It is possible to further improve the interaction with users by designing a more user-friendly interface but we do not focus on it in this chapter. These simple rules can be used to generate training data, thus saving required labeled data to build high-quality property detectors.

The quality of the crude rules is hard to know. A user may have to review all the instances to evaluate or improve the quality for the designed rules. In our framework, we only ask for simple rules (like in Table 4.3) and do not need a user to put a huge amount of work to come up with high-quality heuristic rules.

[3]We assume a user always provides correct labels for the data to be labeled.

In summary, our hybrid framework integrates the crude user-provided rules with an active learning approach to save user labeling effort. The hybrid framework does not ask a user to put effort into designing high-quality rules but aims to address the lacking training data problem in the initial stage.

Label Uncertain Spreadsheets — Our sheet selector applies the uncertainty sampling active learning strategy, and it selects the spreadsheet at the decision boundary for the next round of labeling in order to reduce user effort.

Crude User-provided Rules — In addition to the accumulated user-labeled instances, we use crude user-provided rules to automatically generate additional high-quality training data especially in the initial stage. Then we can approach the ideal decision boundary quickly to reduce the amount of required labeled data. We generate these additional labeled instances by finding those with the agreed decision from both the current trained classifier and the user-provided rules. This bagging-like technique makes it possible for our framework to tolerate *bad* user-provided rules.

We now describe the detail algorithms for the hybrid iterative learning framework and the sheet selector.

4.3.3 Iterative Learning Algorithms

Let $\mathbf{x} = \{x\}$ be the random variables representing a set of sheet tables, and θ_q be the learned classifier for the property $q \in Q$ where Q is the property set containing all the discovered spreadsheet properties. Let θ_{q_init} be the user-provided crude rules for the property q.

First we discuss the algorithms of our hybrid iterative learning framework by considering two different situations, with or without user-provided crude rules.

Without User-provided Rules — Without the user-provided rules in the beginning stage, the iterative learning framework is essentially a typical active learning process.

Algorithm 4 Iterative learning without user-provided rules.

```
Input:  Sheet table set $\mathbf{x} = \{x\}$
Output:  Property detectors $\{\theta_q\}$.
1: $D = []$
2: repeat
3:     sheet selector chooses $x$ from $\{x\}$
4:     ask user to label $x$ with properties $\mathbf{q}$
5:     $D \leftarrow D \cup (x, \mathbf{q})$
6:     $Q \leftarrow Q \cup \mathbf{q}$
7:     train classifier $\theta_q$ on $D$ for each $q \in Q$
8: until meet stopping criteria
9: return $\{\theta_q\}$
```

As shown in Algorithm 4, the sheet selector selects a new instance from the sheet table set (we describe the detail later); a user labels the instance and sends it to the classifier learner; and finally the classifier learner trains the property detectors according to all the accumulated labeled instances. We iterate the above process until the stopping criteria. We stop by testing whether the performance reaches the plateau (*i.e.* the standard deviation of K continuous points is less than δ, where δ is a predefined threshold).

With User-provided Rules — Given a spreadsheet property q, the user-provided rules θ_{q_init} produces a set of labels $\{l_{q\_init}\}$ on the sheet table set $\{x\}$, and each label l_{q_init} represents whether the corresponding sheet table x has the property q or not. However, we do not know the quality of the rule-generated labels $\{l_{q\_init}\}$.

Given a property q, we collect the training data for each learning iteration in two parts. First we accumulate all the user-labeled training data as D, and we train the current property detector based on D as θ_{q_tmp}. Second we automatically generate additional training data using the currently trained classifier θ_{q_tmp} and the user-provided rules θ_{q_init}. Our insight is that if the label produced by θ_{q_tmp} agrees with the label assigned by θ_{q_init}, we believe this label is trustworthy; otherwise, we cannot trust either label. But if the consensus label conflicts with user labels D, then we still believe the user labeled data. The idea of finding the consensus labels is similar to

Algorithm 5 Iterative learning with user-provided rules.

```
Input: Sheet table set $\mathbf{x} = \{x\}$ and user-provided rules $\{\theta_{q\_init}\}$.
Output: Property detectors $\{\theta_q\}$.
 1: $D = []$
 2: for $q \in Q$ do
 3:     $\{l_{q\_init}\} = \theta_{q\_init}(\{x\})$
 4: end for
 5: repeat
 6:     sheet selector chooses $x$ from $\{x\}$
 7:     ask user to label $x$ with properties $\mathbf{q}$
 8:     $D \leftarrow D \cup (x, \mathbf{q})$
 9:     $Q \leftarrow Q \cup \mathbf{q}$
10:     for $q \in Q$ do
11:         train classifier $\theta_{q\_tmp}$ on $D$
12:         $\{l_{q\_tmp}\} = \theta_{q\_tmp}(\{x\})$
13:         $D' = D + (\{x, l_{q\_tmp}\} \cap \{x, l_{q\_init}\})$
14:         train classifier $\theta_q$ on $D'$
15:     end for
16: until meet stopping criteria
17: return $\{\theta_q\}$
```

the bootstrap aggregating technique (*i.e.* bagging) [19]: it attempts to find the label agreements of multiple classifiers. Based on the bagging-like technique, our approach is able to tolerate "bad" user-provided rules and provide additional high-quality labels especially in the initial stage to warm up the classifiers quickly.

Algorithm 5 shows the detail. Similar to Algorithm 4, the sheet selector selects a new instance; a user labels the correct properties; and finally the classifier learner trains the property detectors by combining the accumulated user labels with the consensus labels from two sides, the current trained classifier and the user-provided rules. We iterate the above process until reaching the performance plateau.

Sheet Selector Algorithms — Now we discuss the algorithms of the sheet selector by considering two situations, the single-task and multi-task learning scenarios. Note that in both cases, the sheet selector chooses random instances in the initial stage, and we set the initial random selection size to be 10 [77].

Single-task Learning — The single-task learning scenario is when we train one property detector at a time. The sheet selector simply applies the uncertainty sampling active learning approach and selects an instance with the probability closest to 0.5 [94].

To be concrete, the sheet selector selects the sheet table x with the maximum $\min((P(l_q = 1 \mid x), P(l_q = 0 \mid x))$, where $P(l_q \mid x)$ represents the probability distribution of the sheet table x contains the property q according to the current trained classifier θ_q.

Multi-task Learning — The multi-task learning scenario can be complicated if we explore the correlations among multiple classifiers. Previous multi-task active learning work attempted to explore the correlations [90, 87]. But in this paper, we assume each property detector is independent and we simply uses the averaged uncertainty score for selection.

To be concrete, the sheet selector selects the sheet table x with the maximum $\frac{1}{|Q|}\sum_{q \in Q} \min((P(l_q = 1 \mid x), P(l_q = 0 \mid x))$, where $P(l_q \mid x)$ represents the probability distribution of the sheet table x contains the property q according to the current trained classifier θ_q.

4.4 Experiments

We conduct an extensive experimental study to test our two goals as follows:

- **Spreadsheet Property Detection** — We investigate the algorithms to build high-quality property detectors with little labeled data.
- **Large-scale Spreadsheet Study** — We survey the distribution of the top five spreadsheet properties (shown in Section 4.2.3) in the large-scale WebCrawl data, and our findings serve as guidelines for designing the spreadsheet-to-relational table transformation system.

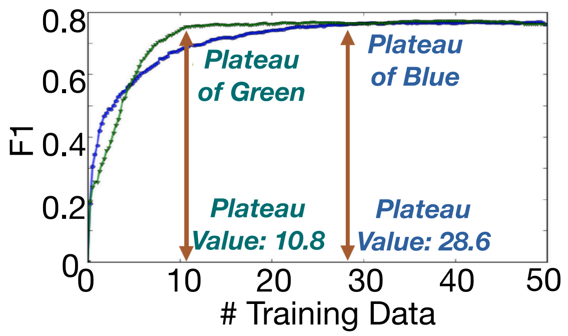


Figure 4.6: An example of "training size to plateau".

	Sheet Selector	**User-provided Rules**
Rand	random selection	N/A
Active	uncertainty sampling	N/A
Hybrid-lb	uncertainty sampling	bad rules
Hybrid	uncertainty sampling	good rules

Table 4.4: Four methods to build property detectors.

Our experiments rely on the two spreadsheet datasets mentioned in Section 4.2.1. The *WebCrawl* data is our large-scale web-crawled spreadsheets containing 410,554 spreadsheets in total, and the *Web400* data is our 400-element hand-labeled sample of the WebCrawl data.

We used a mix of code from several languages and projects. We used the Python xlrd library to access the data and formatting details of spreadsheet files. We extracted the formulas from spreadsheets using the libxl library. We built the classification model using the Python scikit-learn library for its logistic regression, decision tree, and SVM method.

4.4.1 Property Extraction

In this section, we investigate how much labeled data is required to build high-quality property detectors in different situations. We consider the single-task and multi-task learning scenarios as mentioned in Section 4.3.3. We also investigate how the quality of the user-provided rules affects the performance of our hybrid approach.

4.4.1.1 Experimental Setup

We tested the top five spreadsheet properties mentioned in Section 4.2.3. Our experiments are based on the Web400 data. In each of its 20 domains, we split the 20 sheets into 1/2 for potential training and 1/2 for testing, yielding 200 sheets for potential training and 200 for testing.

In the experiments, we simulate the iterative learning framework in Section 4.3.1 and measure the performance of the current trained classifiers for each iteration. We feed the 200 potential training spreadsheets as the spreadsheet dataset for the iterative learning framework. During each iteration, we calculate the F1 score of the currently trained classifiers on the 200 testing data. We simply use the probabilistic model, logistic regression, as the classification method.

We use ***training size to plateau*** as the evaluation metric, and it represents the *least training data size* needed to reach the performance plateau. For example, Figure 4.6 shows the F1 score of a classifier given different sizes of training data. As shown in the Figure, the training size to plateau for the "green" and "blue" methods are 10.8 and 28.6, respectively. This indicates that "green" saves 62.2% of the training data required by "blue" to reach the performance plateau.

Measuring the training size to plateau is similar to the task of knee point detection [116]. For simplicity, we detect the training size to plateau using the following two criteria. First, we use the standard deviation σ to test whether the standard deviation of five consecutive points is less than a threshold δ. To avoid reaching a local optima, we also test whether the current performance (*i.e.*, F1) is above a predefined threshold θ_{F1}. In the experiment, we are able to calculate the F1 score when we use up all the 200 potential training data as $F1_{opt}$, and we simply set $\theta_{F1} = F1_{opt} - \delta$.

We test our iterative learning framework using the four approaches as shown in Table 4.4. Rand randomly selects the next spreadsheet and does not use any user-provided rules; Active employs the uncertainty sampling active learning approach

@δ = **0.01**

Methods	*agg_row*	*agg_col*	*hier_data*	*hier_head*	*crosstab*
Rand	98	170	59	191	113
Active	56	140	42	131	52
Hybrid-lb	56 (0%)	126 (-10%)	45 (+7%)	92 (-30%)	59 (+13%)
Hybrid	**44** (-21%)	**109** (-22%)	**27** (-36%)	**31** (-76%)	**42** (-19%)

@δ = **0.05**

Methods	*agg_row*	*agg_col*	*hier_data*	*hier_head*	*crosstab*
Rand	37	101	33	86	64
Active	28	61	33	98	41
Hybrid-lb	31 (+11%)	66 (+8%)	35 (+6%)	39 (-60%)	45 (+10%)
Hybrid	**16** (-43%)	**52** (-15%)	**18** (-46%)	**22** (-78%)	**31** (-24%)

Table 4.5: The training size to plateau for four property detection methods with $\delta = 0.01$ and $\delta = 0.05$, and the % represents the improvement over Active.

without considering user-provided rules; Hybrid-lb and Hybrid are our hybrid approach that integrates the uncertainty sampling active learning approach with crude user-provided rules. Hybrid-lb assumes "bad" user-provided rules while Hybrid assumes "good" rules. For Hybrid, we use the designed rules for each spreadsheet property as shown in Table 4.3; and for Hybrid-lb, we use the rules for other spreadsheet properties. For example, to build the property detector for "agg_row", we test each of the other four rules (*e.g.*, "agg_col" and "hier_data").

For each method above, we run 100 times to obtained the averaged F1 score for different sizes of training data, and we report the training size to plateau. Except for Hybrid-lb, we run 100 times with each of the four "bad" user-provided rules, totaling 400 times. We then report the averaged training size to plateau for the four configurations.

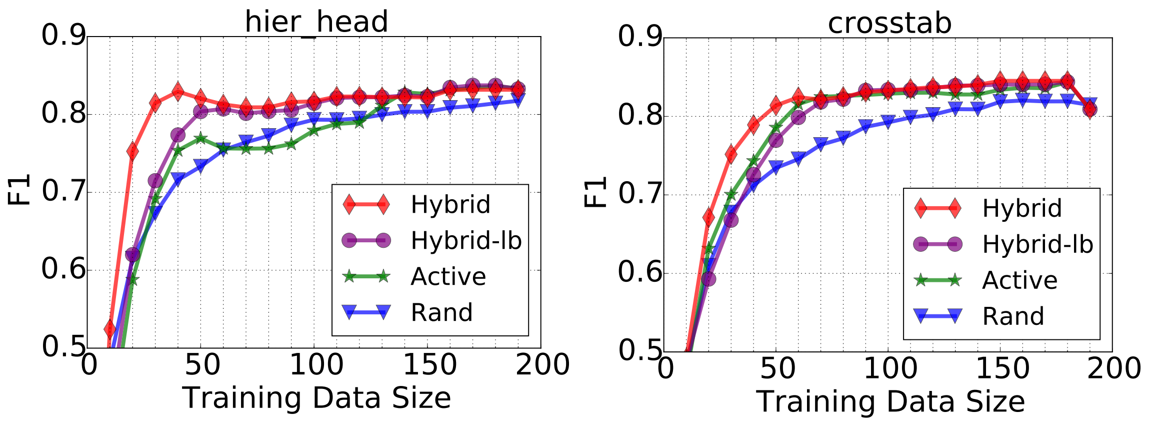


Figure 4.7: The F1 score curve to learn two property detectors individually.

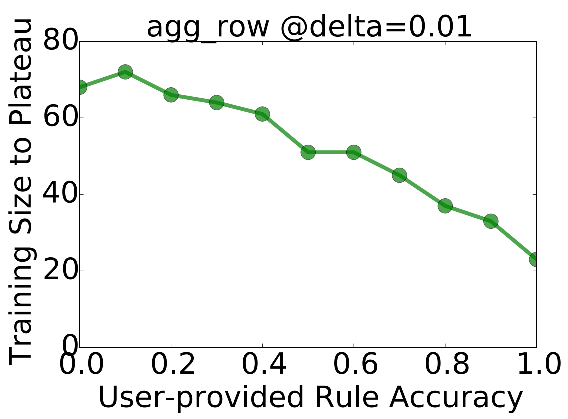
4.4.1.2 Single-task Learning

In this section, we learn the property detectors for the five spreadsheet properties individually.

Table 4.5 shows the training size to plateau for the four testing methods. As shown in the table, Hybrid significantly outperforms all the other three methods. It means that when a user provides with good rules in the beginning stage, we are able to save 35% (when $\delta = 0.01$) or 41% (when $\delta = 0.05$) labeled data in average, compared to the standard active learning method Active. In addition, we can see Hybrid-lb is comparable to the standard active learning approach Active, and it indicates that our hybrid approach is able to tolerate bad user-provided rules. Figure 4.7 shows two examples of the F1 score curve for different sizes of training data when learning the property detectors for "hier_head" and "crosstab".

Rule Qualities — We also test the how the quality of user-provided rules affect the speed to reach plateau.

We generate rules of different accuracy synthetically based on the 200 potential training data. Consider generating the user-provided rules with accuracy 0.3. Given a property, we randomly select 200×0.3 spreadsheets and assign them with their *true* labels, and we assign the remaining $200 \times (1 - 0.3)$ spreadsheets with the *false*



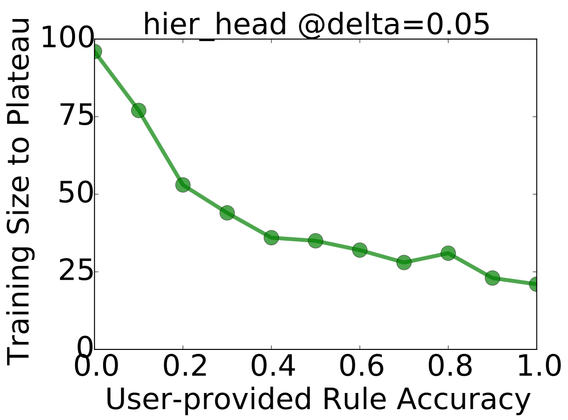


Figure 4.8: The quality of user-provided rules influences the training size to plateau.

labels. We then feed this synthetically labeled data into our hybrid framework as the user-provided crude rules with the accuracy 0.3.

We generate the synthetic rules with the accuracy ranging from 0 to 1 by 0.1 to feed into our hybrid iterative learning framework. We ran 100 times for each accuracy level and obtained the averaged F1 score to calculate the training size to plateau for each spreadsheet property detector.

Figure 4.8 shows two examples of the training size to plateau for rules with different accuracy. As shown in the Figure, the training size to plateau decrease almost linearly when the user-provided rule accuracy improves for "agg_row" at $\delta = 0.01$ and "hier_head" at $\delta = 0.05$. This observation also applies to the rest properties (*i.e.*, "agg_col", "hier_head" and "crosstab").

4.4.1.3 Multi-task Learning

In this section, we learn the property detectors for the five spreadsheet properties together.

Figure 4.9 shows the F1 scores for different sizes of training data when learning the five property detectors together. As shown in the Figure, Hybrid reaches the plateau much sooner than the other three methods. It saves 44% (when $\delta = 0.01$) and 34% (when $\delta = 0.05$) training data, when compared to the standard active learning

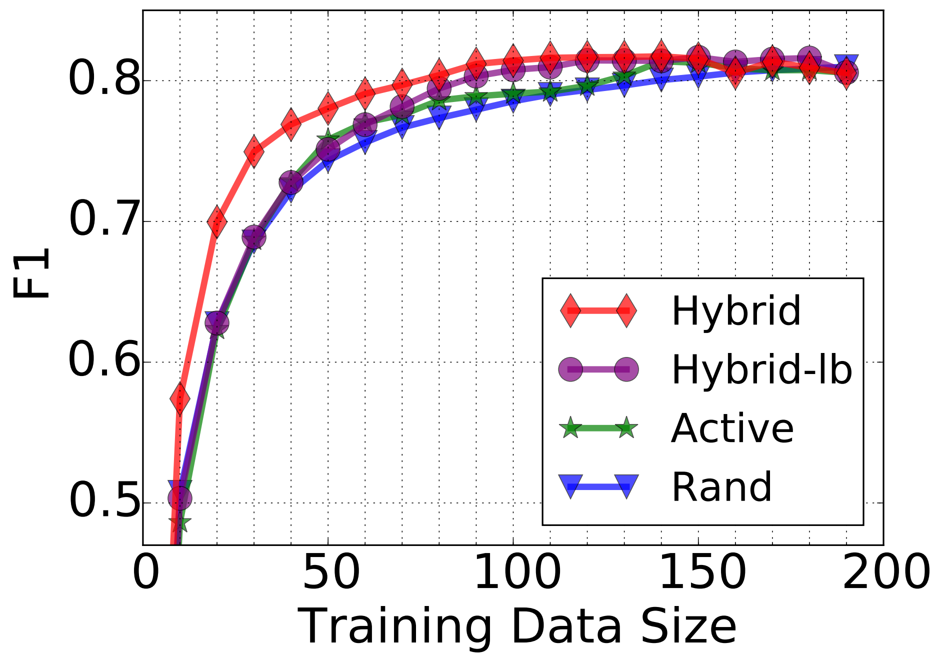


Figure 4.9: The F1 performance curve to learn the five property detectors together.

approach Active. It indicates that "good" user-provided rules do save a significant amount of extra labeling work. In addition, Hybrid-lb is comparable to Active, and it indicates that our hybrid framework is able to tolerate "bad" user-provided rules.

In summary, compared to the standard active learning approach, our hybrid approach is able to save 34%-44% of the training data in average to reach the performance plateau when a user provides relatively high-quality rules, and performs comparably with low-quality rules.

4.4.2 Large-scale Spreadsheets Study

In this section, we investigate the distribution of the five spreadsheet properties mentioned in Section 4.2.3 in the large-scale WebCrawl dataset. We evaluate the performance of the five property detectors using Web400 data, and then show two observations on the large-scale WebCrawl data.

4.4.2.1 Experiment Setup

We obtained 1,181,530 spreadsheets from 410,554 .xls workbook files in the WebCrawl data.[4] To obtain the spreadsheet property statistics on this large-scale data, we pass each spreadsheet to a two-step pipeline that consists of the *frame finder* and the *property detectors*.

Frame Finder — We ran the the frame finder as discussed in Section 3.3.2 on the WebCrawl data, and it identifies the "header" and "data" rows for each spreadsheet. Then we can obtain the sheet table for a spreadsheet by keeping the header rows as the header region and the data rows as the data region.[5]

We evaluated the frame finder on the Web400 data via the 2-fold cross-validation. The averaged F1 for "header" and "data" is 0.807 and 0.996, respectively. It demonstrate the frame finder is fairly accurate in predicting the the head and data rows, and it can be used to apply on the WebCrawl data for more interesting data analysis.

Property Detector — We trained property detectors for the five spreadsheet properties using all the Web400 data and then ran the the five classifiers on the WebCrawl dataset.

We evaluate the performance of the spreadsheet property detectors for the five spreadsheet properties on the Web400 data via the 2-fold cross-validation. We use two common metrics: The *accuracy* measures the percentage of spreadsheets which we correctly recognize whether it contains a given spreadsheet property. The *F1* measures the harmonic mean of precision and recall for each spreadsheet property.

Table 4.6 shows the performance of the spreadsheet property detectors using three classification methods: LR (*i.e.*, logistic regression), DTs (*i.e.*, decision trees) and SVM (*i.e.*, support vector machine with the linear kernel). As shown in the table, logistic regression performs the best among the three classification methods, and thus

[4]One .xls workbook file might contain multiple spreadsheets.

[5]Note that if there are multiple sheet tables in a spreadsheet, we only retain the first one. We assume there are multiple sheet tables if we see a "header" row below a "data" row.

F1

Method	*agg_row*	*agg_col*	*hier_data*	*hier_head*	*crosstab*
LR	0.876	0.844	0.782	0.845	0.798
DTs	0.825	0.788	0.746	0.772	0.689
SVM	0.855	0.823	0.749	0.815	0.766

Accuracy

Method	*agg_row*	*agg_col*	*hier_data*	*hier_head*	*crosstab*
LR	0.894	0.917	0.856	0.923	0.895
DTs	0.849	0.891	0.834	0.892	0.843
SVM	0.876	0.908	0.835	0.912	0.880

Table 4.6: The F1 and accuracy of five spreadsheet property detectors using three different classification methods.

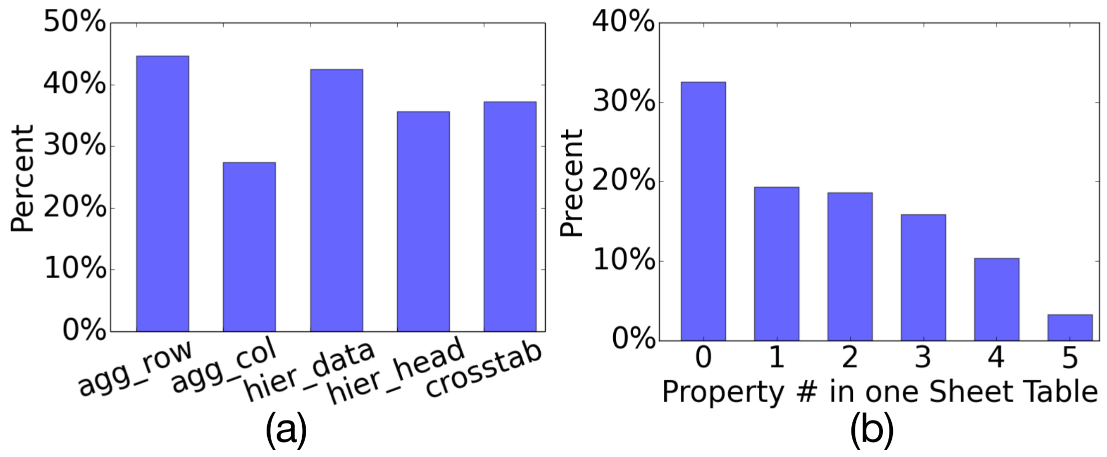


Figure 4.10: The distribution of the five spreadsheet properties in the web.

we used logistic regression as the classification model for the spreadsheet property detection. Note that accuracy is always better than F1, because the spreadsheet properties are unbalanced (few positive examples and more negative examples).

4.4.2.2 Observations on WebCrawl Data

As a result, we obtained the spreadsheet properties assigned to each of the 1, 181, 530 WebCrawl spreadsheets. We have two observations on the web spreadsheets.

Observation 1 — *There is a significant amount of spreadsheets in the web which contain each of the five spreadsheet properties.* Figure 4.10 (a) shows the distribution of the five spreadsheet properties on the web. As shown in the figure, the ratio of the web spreadsheets containing the five spreadsheet properties ranges from 27.4% to 44.7%. It indicates that there is a significant portion of spreadsheets in the web containing each of the five spreadsheet properties. The property "agg_row" is the most popular among the five, followed by "hier_data", and their proportions are all greater than 40%.

Observation 2 — *The majority of the spreadsheets in the web contain at least one spreadsheet property.* Figure 4.10 (b) shows the distribution for the number of properties in one spreadsheet. It shows that there are 32.6% spreadsheets without any of the five spreadsheet properties; there are 67.4% web spreadsheets containing at least one spreadsheet property. It indicates that there is a much larger portion of the web spreadsheets containing a variety of spreadsheet properties than those without any property.

In summary, the majority of the spreadsheets in the web contain one or more than one spreadsheet properties. In order to transform a large number of spreadsheets into a high-quality relational form, we have to identify a variety of spreadsheet properties.

4.5 Related Work

There are two main areas of related work, spreadsheet management and active learning. The related work on spreadsheet management has been discussed in Section 3.6.

Human-assisted Active Learning — The two common active learning [97] approaches are uncertainty sampling and Query by committee (QBC) as we discussed in Section 2.2.3.

Beyond traditional active instance labeling, there are alternative techniques for utilizing human resources for model development. Attenberg and Provost [8] use a "guided learning" approach to search explicitly for training examples of each class. Druck *et al.* [49] propose an active learning approach in which the machine solicits labels on features rather than instances. Like these work, we ask users for more information (in our case, the crude user-provided rules). But different from their situation, the quality of the crude user-provided rules can be very low. But our hybrid approach can tolerate "bad" rules via using a bagging-like technique.

We notice that active learning strategies often suffer from the "cold-start" problem [117]: in the beginning stage, the classifier lacks training data to approach the ideal decision boundary and suggest effective instances to label. Zhu *et al.* [117] address this problem by finding clusters of distinct content among the unlabeled instances. Donmez *et al.* [48] propose to use a robust combination of density weighted uncertainty sampling and standard uncertainty sampling to overcome the cold-start problem. In this paper, we propose an alternative approach to address this problem by asking users with crude heuristic rules. Our hope is that the user-provided rules can provide additional high-quality labels especially in the initial stage to warm up the classifiers quickly.

4.6 Conclusion and Future Work

We have described a hybrid iterative learning framework to construct spreadsheet property detectors quickly, and it is the first step toward building the spreadsheet-to-relational table transformation pipeline that is able to handle a large variety of spreadsheets. Our hybrid approach integrates the active learning framework with crude easy-to-write user-provided rules, and it is able to save more training data to reach the performance plateau when compared to the standard active learning method.

In the future work, we want to build the spreadsheet-to-relational table transformation system using the spreadsheet property detectors. We will also investigate the user interface design to allow more effective interactions with users in order to conduct accurate and low-effort transformation.

CHAPTER V

Lyretail: Extraction on Dictionaries from Webpages

5.1 Problem Overview

In information extraction (IE), a *dictionary* (also known as gazetteers) refers to a set of instances belonging to the same conceptual class; for example, a camera brand dictionary contains "Canon", "Nikon" and so on. Dictionaries are extremely useful in many IE tasks. For example, many information extraction systems use dictionaries along with a set of textual features to extract entities (e.g. Person, Organization, Location) and relationships between entities (e.g. Person's birth date or phone number) [73, 93]. Moreover, dictionaries are a useful primitive for many real-life applications. For example, commercial engines such as Google and Yahoo!, use dictionaries for query analysis, document categorization, and ad matching [83]. The quality and coverage of the dictionaries is essential to the success of those applications [93].

In this chapter, we present Lyretail, an extraction system that is able to build high-quality dictionaries especially in long-tail vocabulary settings.

First, Lyretail generates training data for building high-quality page-specific extractors automatically: a form of distant supervision. In particular, it uses the co-occurrence information from our large-scale crawled web lists dataset to obtain negative examples; our key insight is that negative examples often co-occur with entities that do not belong to the initial dictionary. For example, “Gitzo” is a negative example, as none of its cooccurrence partners (*e.g.*, “Manfrotto”) belong to the initial dictionary generated by a few camera band seed examples. Second, we propose a co-training framework that can incorporate sequential features to jointly infer the high-quality dictionary on each single webpage. We leverage the output from the extractor built on web lists to train a semi-supervised conditional random field (CRF) as the resulting page-specific extractor. As a result, we are able to build high-quality page-specific extractors on each webpage; these are often able to distinguish infrequently-observed true extractions from incorrect ones. These results are then aggregated into high-quality long-tail dictionaries.

There is an additional problem we face when exploiting page-specific extraction: the intended target page-specific dictionary (PSD) is sometimes ambiguous given only a few seeds. For example, if a user provides examples of “Atlanta Braves”, she could have intended a dictionary of Major League Baseball (MLB) teams or a dictionary of all U.S. sports teams. Fortunately, we can address this problem by building training data at different *granularities*, each of which yields a different PSD. For example, when we include “Brooklyn Nets” as a negative example, we would build a page extractor for the MLB teams; otherwise we would extract all the U.S. sports teams. This method is a convenient side effect of our architecture.

In this chapter, we present the extraction system Lyretail. We introduce the system framework and the data sources in Section 5.2. We present the algorithms to extract dictionaries from webpages in Section 5.3. We conduct an extensive experiments in Section 5.4. We discuss the related work in Section 5.5. Finally we conclude and future work in Section 5.6.

5.2 Preliminary

In this section, we present the system framework and discuss the used data sources in the framework.

5.2.1 System Framework

In this section, we present the system framework of Lyretail, as shown in Figure 5.1. Overall, Anthias supports the ***dictionary generation*** process: it takes in seeds and generates a high-quality comprehensive dictionary (CD) as the output. Lyretail consists of three stages, as follows:

1. Webpages Fetcher — First, the seeds are used to ***fetch webpages***. To find webpages that are useful for the page-specific extraction tasks, we use an existing search engine to retrieve the top-k webpages to serve as the input webpages. We simply concatenate all the seeds as the search query. For example, if the seeds are {canon, nikon}, we formulate the disjunctive search query as "canon nikon". We discuss the input webpages in detail in Section 5.2.2.

2. Page-specific Extractor — Second, the page-specific extractor attempts to build a high-quality extractor model for each single input webpage. Lyretail constructs a set of resources for the PSD automatically: training examples and webpage-parameterized features. In particular, the training examples (including both positive and negative examples) are generated from the initial dictionary which is produced based on an existing set expansion method Seal [108]. By generating different sets of training examples, we can subtly change the page-specific extractor to produce PSDs at different "granularities". We discuss this idea further in Section 5.3.3.2.

3. Dictionary Aggregator — After obtaining a PSD on each single webpage, the dictionary aggregator merges the PSDs from many webpages and produces the unified high-quality comprehensive dictionary (CD) as the output.

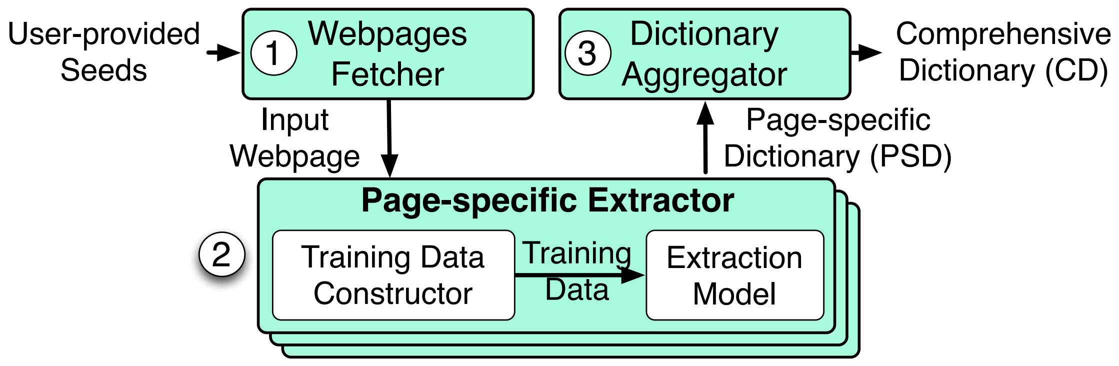


Figure 5.1: The framework of Lyretail.

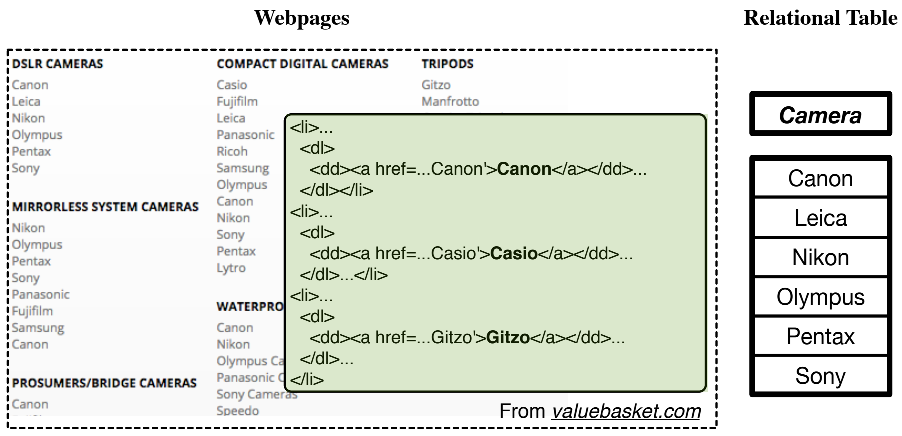
We experimented with making Lyretail an iterative process, like other past extraction systems [4, 21, 109]: the CD of each round of execution can be used as a newer initial dictionary for the next round. In practice, we do not run Lyretail iteratively, as the performance improvement after the 1st round is not significant. (as we discuss in Section 5.4.3).

5.2.2 Data Sources

The page-specific extractor uses two data sources: the input webpages provide the dictionary entity candidates for extraction, and the web lists are used to generate distantly-supervised training data.

Input Webpage — An input webpage is our raw source for target dictionary entities. Thus, we construct a unique *page-specific extractor* for each input webpages, in order to detect the *true* dictionary items as the PSD.

The webpages are generated by a search engine from the user-given seeds and have a variety of formatting styles, as shown in Figure 5.2. Because in this work we do not focus on text segmentation, our crawler uses a handful of rules to retain only "structure-segmented" pages, in which the candidate entities are already segmented by the HTML structure. We treat each leaf HTML element in a webpage as



```
<li>...
  <dl>
    <dd><a href=...Canon'>Canon</a></dd>...
  </dl></li>
<li>...
  <dl>
    <dd><a href=...Casio'>Casio</a></dd>...
  </dl>...</li>
<li>...
  <dl>
    <dd><a href=...Gitzo'>Gitzo</a></dd>...
  </dl>...
</li>
```

Figure 5.2: A list of camera manufacturers from a webpage (with corresponding HTML source code) can be represented in the relational format.

a candidate for the PSD.

Lyretail is currently designed to handle a particular kind of stylized text: those with recurrent patterns. In Figure 5.2, “Canon” and “Gitzo” are in a recurrent pattern (though only “Canon” is a correct camera brand extraction). We use the positive examples to find all the dictionary entities on a webpage. For each such entity, we define its *xpath pattern* to be the list of tags of all the nodes along a path to the root in the HTML DOM tree. Since the initial dictionary is often a high-precision but small dictionary, we assume a xpath pattern is a recurrent pattern if it contains at least one dictionary entity found by the initial dictionary. Thus, we identify all the recurrent-pattern entities $\mathbf{x} = \{x_1, ..., x_n\}$ on the webpage and use these as candidates for extraction by the page-specific extractor. There may be more than one recurrent pattern on a webpage.

Web Lists — To generate resources for the page-specific extractor, we use data from a large number of web-crawled HTML lists. We obtained HTML lists from the ClueWeb09 crawl[1], using the regular expression “$\langle ul(.*?)\langle/ul\rangle$”, where each $\langle li\rangle$ item represents a list item. The web lists data can be very noisy. There are many non-entity elements (such as hyperlinks) in lists. Thus, we used the following heuristics to

[1]`http://lemurproject.org/clueweb09.php`

filter the noisy lists: we only keep lists with more than four items; we remove an item if its string length is above a predefined threshold or it contains more than five tokens. We also de-duplicate the crawled lists by their Internet domains: if a list appears in one domain many times, we only count it once. In the end, we obtained 83 million web HTML lists, then used a 9 million sample of this dataset (due to efficiency issues) as our web lists dataset.

5.3 Dictionary Extraction from Webpages

In this section, we introduce Anthias's page-specific extractor. We formulate the page-specific extraction problem as a classification task. Let $\mathbf{s} = \{s_1, ..., s_m\}$ be the set of m seeds, and $\mathbf{x}$ be the candidates for the PSD on this input webpage. Each entity $x \in \mathbf{x}$ takes a label $l(x) \in \{\mathsf{true}, \mathsf{false}\}$, representing whether x is a *true* dictionary entity or not. Therefore, the page-specific extractor's task is to assign a label to each $x \in \mathbf{x}$ such that $\{x | l(x) = \mathsf{true}\}$ represents the extracted PSD.

Now we introduce our two distinct feature sets and distantly-supervised training examples to construct the page-specific extraction model.

5.3.1 Two Distinct Sets of Features

We are able to derive two distinct sets of features: *web list features* from the web lists dataset, and *webpage-parameterized features* from the target extraction webpage.

5.3.1.1 Web List Features

We are able to define a set of features on the web lists dataset, as some of the lists are more likely to contain positive or negative instances. We formally define the *web list features* as $\mathbf{f_d} = \{f_{d_i}(x)\}$, where each $f_{d_i}(x) \in \mathbf{f_d}$ is a boolean function represents whether an entity x belongs to a unique list L_i from the web lists dataset.

No.	Features
1	Current element's HTML tag
2	Current element's HTML attributes
3	Previous item's HTML tag
4	Previous item's HTML attributes
5	Next item's HTML tag
6	Next item's HTML attributes
7	Parent item's HTML tag
8	Parent item's HTML attributes
9	Preceding word
10	Following word

Table 5.1: The HTML structural property features.

The raw number of the web list features could be huge, but we reduce the dimension of the features via two ways: first we remove a feature if it assigns the same value to all the entities, and we also employ the feature selection methods [53] to pick the top-k features.

5.3.1.2 Webpage-parameterized Features

For each webpage, we synthesize a large number of *webpage-parameterized features* without direct human knowledge of the page. These features are not written by hand and do not embody direct human knowledge of each webpage. However, they are also not the same as traditional "domain-independent" features which are fixed for a large set of webpages. Instead, the feature set differs from webpage to webpage. These webpage-parameterized features are feasible because we are able to generate webpage-specific training data.

We parse the HTML code and construct *two* families of features for each dictionary candidate x on a webpage, *entity features* and *sequential dependency*, as follows:

Entity Features — The *entity features* describe the properties associated with each entity x, and they contain *three* categories of properties. The first category describes entities' HTML *structural properties*, as shown in Table 5.1. They operate on each entity $x \in \mathbf{x}$. Each feature serves as a boolean function, representing whether the entity has this property. For example, "Canon" in Figure 5.2 generates the structural property features, including whether the current element's tag is "$\langle a \rangle$" (Feature 1).

The *xpath properties* are boolean functions for each entity $x \in \mathbf{x}$ based on its xpath. The xpath for each entity consists of a set of HTML elements along the path to the root in the HTML DOM tree. For each xpath, we use all the xpath elements with tag information to generate the xpath property feature space. For example, the xpath "/html/table[2]" (*i.e.*, the second "table" HTML element under "html") has the *elements* "html" and "table[2]", where "table" is the *tag* of the element "table[2]". The xpath "/html/table[2]" generates xpath features, including whether "table[2]" is the depth-2 xpath element, and whether "table" is the depth-2 xpath tag.

Finally, we created three *textual properties*: whether the current element contains letters; whether the current element contains numbers; and whether the current element contains punctuations.

In the end, we merge all the features mentioned above but remove a feature if it assigns the same value to all the entities, yielding the entity features $\mathbf{f_k} = \{f_k(x, l)\}$. The resulting number of the webpage-parameterized features is often tractable, and we give more details in Section 5.4.1.

Sequential Dependency — The *sequential dependency* characterizes the sequential pattern of adjacent entities. If we consider all the elements as a sequence according to its appearance in textual order, labels for adjacent elements may follow transition patterns. This observation can be incorporated as linear-chain pairwise transition features $\mathbf{f'_k} = \{f'_k(x_i, l_i, x_{i+1}, l_{i+1})\}$.

5.3.2 Training Data Construction

Training data is a critical need for our page-specific extractor. Direct human supervision is too expensive, so we use a distant supervision based method to generate training data. Figure 5.3 shows the training examples of the webpage entities and the

Webpage Entities	***Training Example***	***Desired Output***
Canon	True	True
Leica	Unknown	True
Nikon	True	True
Olympus	True	True
Pentax	Unknown	True
Gitzo	False	False
Manfrotto	Unknown	False

Figure 5.3: An example of the training data and the desired output for a part of entities in Figure 5.2 (a).

desired output. Now we discuss how to generate the positive and negative training data.

As mentioned earlier, our training examples are generated using the low-precision, high-recall web lists data presented in Section 5.2.2. The web lists data is of a large scale and contains a broader range of entities than existing knowledge bases (*e.g.*, Freebase). The key insight (observed in [60]) is that in web lists, entities belonging to the same set tend to co-occur frequently, and entities that are not members of the same set are less likely to co-occur.

Positive Examples — *Positive examples* are derived from both a *low-recall* but *high-precision* initial dictionary $\mathbf{d}$ and the web lists data.

First, for each entity $x \in \mathbf{x}$, we test whether x is contained in $\mathbf{d}$. If $x \in \mathbf{d}$, then x is a positive instance; otherwise it is not. We generate the initial dictionary $\mathbf{d}$ using an existing set expansion technique (*e.g.*, Seal [108]). For example, given seeds "Canon" and "Nikon", we can use Seal to produce a ranked list of popular camera brand names as $\mathbf{d}$.

In addition, if an entity co-occurs often with the entities in the initial dictionary $\mathbf{d}$, it should also be a positive instance. We use the *initial dictionary similarity* (IDSim) to characterize the similarity between an entity x and the initial dictionary $\mathbf{d}$ as follows:

$$IDSim(x, \mathbf{d}) = \frac{1}{|\mathbf{d}|} \sum_{e \in \mathbf{d}} Sim(x, e) \tag{5.1}$$

where $Sim(x, e) = \frac{|LS_i \cap LS_j|}{|LS_i \cup LS_j|}$, ($LS_i$ and LS_j represent the set of lists that contain the entity e_i and e_j, respectively). It represents how frequently two entities co-occur in one list [60]. If $IDSim(x, \mathbf{d}) \geq \lambda_p$, we know that the entity x is relatively likely to co-occur with entities in $\mathbf{d}$ and x is a positive instance; otherwise it is not.

In summary, for each $x \in \mathbf{x}$, if $x \in \mathbf{d}$ or $IDSim(x, \mathbf{d}) \geq \lambda_p$, x is a positive instance; otherwise it is not. In practice we chose $\lambda_p = 0.01$.

Negative Examples — Negative examples are essential in constructing an effective extraction model, but finding them automatically in our application is not straightforward. The key insight is that if an entity strongly co-occurs only with entities excluded by the initial dictionary, we have some evidence this entity is a negative example. For example, in Figure 5.2, we might observe that “Gitzo” often co-occurs with “Manfrotto” and so on but rarely with an entity in the initial dictionary; thus, we believe “Gitzo” is a negative example.

We choose the negative training example based on the following two properties. First we use the initial dictionary similarity (IDSim) to determine if the testing entity x is relatively unlikely to co-occur with entities in the initial dictionary $\mathbf{d}$: if $IDSim(x, \mathbf{d}) \leq \lambda_d$ (where λ_d is a predefined threshold), we know that the co-occurrence between x and the entities in $\mathbf{d}$ is unlikely.

Second, we ensure an entity must have a certain level of popularity in the web lists, to ensure we have enough evidence to label it as negative. We define an entity x's *popularity* using its frequency, $freq(x)$. Thus, if $freq(x) \geq \lambda_f$, where λ_f is a pre-defined threshold, we assume x is popular; otherwise, it is not.

In summary, we assume an entity $x \in \mathbf{x}$ is a negative example if $IDSim(x, \mathbf{d}) \leq \lambda_d$

and $freq(x) \geq \lambda_f$, where λ_f and λ_d are two predefined thresholds.[2] We also call λ_d the *negative example threshold*. In practice, we set $\lambda_f = 50$ but vary the value of λ_d to produce negative training examples at different granularities. The resulting negative training sets can further be used to produce page-specific dictionaries at different granularities. We discuss this dictionary granularity problem in Section 5.3.3.2.

5.3.3 Implementing Page-specific Extractors

A conventional way to construct the page-specific extractor is to simply combine the two sets of features mentioned in Section 5.3.1 and build a single classifier that labels candidate dictionary items as "in-dictionary" or not. However, in our setting the two feature sets are derived from two distinct data sources, one from the web lists data and one from the input webpage; in similar situations, past researchers have had success with *co-training methods*. These methods train multiple distinct classifiers to maximize their mutual agreement in order to improve learning performance [114]. We pursue a similar strategy (and show in Section 5.4 that we can thereby beat the conventional approach).

Thus, we develop two distinct classifiers defined on two distinct sets of features: the ***list extractor*** E_{list} defined on the web lists features and the ***webpage extractor*** E_{page} defined on the webpage-parameterized features.

The ***list extractor*** E_{list} is a non-page-specific classifier. Let $\mathbf{x} = \{x\}$ be the set of candidates for the PSD on the input webpage. Let $\mathbf{l} = \{l(x)\}$ be the corresponding labels for each $x \in \mathbf{x}$, where each $l(x) \in \{\mathsf{true}, \mathsf{false}\}$ represents whether the entity x is a *true* dictionary entity or not. The probability distribution of the classifier E_{list} is defined as:

$$P_{list}(\mathbf{l} \mid \mathbf{x}) = \frac{1}{Z} \exp\,(\sum_{x} \mathbf{w_d f_d})$$

[2]We also used heuristics to filter out "obviously bad" candidates as negative examples. *E.g.*, we remove items that were extremely long in either character length or token length.

where $\mathbf{f_d}$ are the web list features mentioned in Section 5.3.1.1, $\mathbf{w_d}$ are associated weights, and Z is a normalization factor.

Similarly, the ***webpage extractor*** E_{page} is page-specific and uses a conditional random field defined on the PSD candidates $\mathbf{x}$. The joint distribution of the classifier E_{page} is:

$$P_{page}(\mathbf{l} \mid \mathbf{x}) = \frac{1}{Z'} \exp\left(\sum_{x}(\mathbf{w_k}\mathbf{f_k} + \mathbf{w_{k'}}\mathbf{f_{k'}})\right)$$

where $\mathbf{f_k}$ and $\mathbf{f_{k'}}$ are the webpage-parameterized features mentioned in Section 5.3.1.2, $\mathbf{w_k}$ and $\mathbf{w_{k'}}$ are associated weights, and Z' is a normalization factor.

5.3.3.1 Combining Two Extractors

To combine the two classifiers is not straightforward. The *list extractor* may fail to distinguish items that are lacking information in the web lists: while the web lists is of a large scale, for some extremely rare entities, it is possible that we find *no* information in the web lists. On the other hand, the *webpage extractor* is able to produce expressive features for all testing dictionary candidates, but it requires further information: we are not able to train an accurate CRF with only partial labels of the sequence data.

Thus, we propose a co-training method that trains the two classifiers so as to encourage consensus decisions. As shown in Algorithm 6, we utilize the *list extractor* to construct an accurate *webpage extractor*. Let E_{list} be the *list extractor*, and E_{page} be the *webpage extractor*. Let $D = \{\mathbf{x_0}, \mathbf{l_0}\}$ be the automatically generated training data as discussed in Section 5.3.2. We first train the *list extractor* E_{list} using the training data D, and measure E_{list}'s mean accuracy of prediction on $\mathbf{x_0}$ with respect to $\mathbf{l_0}$. If the accuracy score is fairly accurate and above a predefined threshold θ_a, we obtain E_{list}'s predicted probability distribution on $\mathbf{x_0}$ as the *prior distribution* P_{list}; otherwise we use the default *prior distribution* $P_{default}$.[3]

[3]In practice we simply set $\theta_a = 0.8$, $p_l = 0.95$

Algorithm 6 Co-training Algorithm

```
1: Train $E_{list}$ on $D$
2: $mscore = mean\_accuracy(E_{list}(\mathbf{x_0}), \mathbf{y_0})$
3: $P_{default} = \{p(x = l | x \in \mathbf{x_0}) = p_l^l (1 - p_l)^{1-l}\}$
4: $P_{list} = (mscore > \theta_a) ? E_{list}(\mathbf{x}); P_{default}$
5: Train $E_{page}$ by optimizing Equation 5.2
6: Return $E_{page}(\mathbf{x})$
```

To estimate the parameters for the *webpage extractor* E_{page}, we utilize a semi-supervised conditional random field framework [78] that ensures E_{page} produces a similar probability distribution to E_{list}. We estimate the unknown parameters $\mathbf{w} = \{\mathbf{w_k}, \mathbf{w'_k}\}$ by maximizing the likelihood of the objective function $P_{page}(\mathbf{l} \mid \mathbf{x})$, based on the training data $D' = \{\mathbf{x}, E_{list}(\mathbf{x})\}$. The goal is to maximize the regularized log likelihood, as follows:

$$\begin{aligned} \max_{\mathbf{w}} \sum_x \mathbf{w_k f_k} + \sum_x \mathbf{w'_k f'_k} - logZ(\mathbf{w_{k,k'}}) \\ -\lambda D(P_k || P_{list}) - \frac{\sum_k w^2}{2\sigma^2} \end{aligned} \tag{5.2}$$

where $\frac{\sum_k w_k^2}{2\sigma^2}$ is a common choice of regularization to avoid overfitting, based on the Euclidean norm of $\mathbf{w}$ and on a regularization parameter $\frac{1}{2\sigma^2}$.

After obtaining the values for $\mathbf{w} = \{\mathbf{w_k}, \mathbf{w'_k}\}$, we infer the most likely assignment for each variable $x \in \mathbf{x}$ by employing a dynamic programming algorithm (*i.e.*, Viterbi) [75, 78].

5.3.3.2 Webpage Dictionary Granularity

With just a few user-given seeds, the extraction target may be ambiguous. For example, given seeds "Atlanta Braves" and "Chicago Cubs", it is not clear what is the ideal PSD: the user may intend to get all the MLB teams or all of the U.S. sports teams. It worth noting that the granularity issue only applies to certain categories; not all the dictionary categories have the ambiguous granularity. For example, the

Algorithm 7 PSDGranularities

```
Input:  a set of negative example thresholds $\{\lambda_d\}$, a webpage $wp$
Output:  PSDs of granularities $G$
1: $G = []$
2: for each $\lambda_d$ in $\{\lambda_d\}$ do
3:     Get positive training data $\{x_{pos}\}$
4:     Get negative training data $\{x_{neg}\}$ using $\lambda_d$
5:     Train page-specific extractor $E$ using $\{x_{pos}\}$ and $\{x_{neg}\}$
6:     Apply $E$ on webpage $wp$ and extract a PSD $g$.
7:     $G \leftarrow G \cup g$
8: end for
```

seeds "Amaranth Pink" and "Red" can be used to indicate a dictionary of color in a straightforward way.

We can address this PSD ambiguity problem by manipulating the negative example threshold λ_d. Consider the seeds "Atlanta Braves" and "Chicago Cubs" in relation to entities "Brooklyn Nets" and "Football". If a user intends to extract MLB teams, both "Brooklyn Nets" and "Football" should be negative examples; if a user intends to extract all U.S. sports teams, "Football" should be a negative example while "Brooklyn Nets" should not be. If "Brooklyn Nets" has a higher IDSim than "Football", we are able to construct different training sets by varying λ_d. By using a tighter threshold we can obtain a training set that is tailored for just MLB teams; by using a looser threshold we can obtain a training set that admits all the sports teams on the page. We demonstrate the PSD granularities with two examples in Section 5.4.1.

5.3.4 Dictionary Aggregator

In this section, we present the algorithm of Lyretail's *dictionary aggregator*. The dictionary aggregator merges PSDs from a set of input webpages and produces a high-quality CD as the output. Similar to Seal, we employ the *lazy walk process* technique [108] to merge PSDs. The goal of this method is to formulate a quality score for each unique entity in PSDs, and the score represents the similarity between the entity and items in the initial dictionary.

The *lazy walk process* technique is defined on a random walk graph. To construct the random walk graph, we create a node for each webpage and a node for each unique entity in PSDs. An edge exists between a webpage node and an entity node if the entity appears on the webpage.

Our transition process is similar to Seal [108]. To transit from a source node x, one randomly picks an adjacent node y with a uniform distribution proportional to the degree of node x. More specifically, $p(y|x) = \frac{1}{degree\ of\ x}$. At each step, there is also some probability α of staying at x. Putting everything together, the probability of reaching any node z from x is computed recursively as follows: $P(z|x) = \alpha I(x = z) + (1 - \alpha) \sum_y P(y|x)P(z|y)$, where $I(x = z)$ is a binary function that returns 1 if node x and node z are the same, 0 otherwise. We used Seal's setting for this lazy walk process (*e.g.*, choosing $\alpha = 0.5$), except for the starting source nodes.

To capture the similarity of an entity to items in the initial dictionary $\mathbf{d}$, we define the starting source nodes as a probability distribution $\mathbf{p_0}$ over a set of entity nodes $\mathbf{n_s}$ in the graph s.t. for each $n \in \mathbf{n_s}$, $n \in \mathbf{d}$. Recall that $\mathbf{d}$ is a ranked list and its top m entities are the seeds. It is intuitive that an entity ranked higher in $\mathbf{d}$ should have a higher probability in $\mathbf{p_0}$ than one ranked lower. We use a power-law distribution to characterize this property. Let β be the decay ratio and $0 < \beta < 1$ (in practice we chose $\beta = 0.9$). Let σ is a normalization factor to ensure the sum of $\mathbf{p_0}$ to be 1. The initial distribution $\mathbf{p_0}$ is defined as follows: for $d_i \in \mathbf{d}$ where i represents d_i's rank in $\mathbf{d}$, $p_0(d_i) = \frac{1}{\sigma}$ if $i \leq m$, and $p_0(d_i) = \frac{\beta^{k-i}}{\sigma}$ if $i > m$; for $e \notin \mathbf{d}$, $p_0(e) = 0$.

5.4 Experiments

In this section, we evaluate and demonstrate that Lyretail is able to produce high-quality dictionaries especially in long-tail vocabulary settings. First, we measure the performance of the page-specific extraction and also demonstrate that it can produce

Category	Seed Examples
Common Vocabulary Dictionaries	
country	nepal, hungary, burkina faso
mlb-team	chicago white sox, san francisco giants, pittsburgh pirates
nba-team	new york knicks, charlotte bobcats, san antonio spurs
nfl-team	philadelphia eagles, cleveland browns, san diego chargers
us-president	andrew jackson, john quincy adams, thomas jefferson
us-state	delaware, virginia, michigan
Long-tail Vocabulary Dictionaries	
disease	alopecia, cold sore, scabies
mattress	sealy, serta, simmons
camera	fujifilm, minolta, huawei
cmu-building	resnik house, fraternity quadrangle, mellon institute
color	metal navy, windsor tan, rose quartz

Table 5.2: Three seed examples of 11 dictionary categories for the page-specific extraction evaluation.

page-specific dictionaries (PSDs) at different granularities. Second, we evaluate the quality of the comprehensive dictionary (CD) emitted by the Lyretail dictionary generation process. Finally we test Lyretail's system configuration by trying different parameter settings.

To evaluate our system, we have collected 11 dictionary categories, as shown in Table 5.2. They are cmu-building, country, disease, mlb-team, nba-team, nfl-team, us-president, us-state, camera-marker, color, and mattress-maker. These categories are chosen based on previous work [60, 108]. We use cmu-building, mattress, camera, disease and color to illustrate long-tail vocabularies (*i.e.*, dictionaries contain long-tail entities); we use the rest 6 categories to illustrate the common vocabularies (*i.e.*, dictionaries that only consist of common entities). We selected three random seeds from each of the 11 categories as follows: for the first eight categories, we randomly sample from the dictionary instances collected from [107]; for camera and color, we randomly selected three seeds from Wikipedia;[4] for mattress, the seeds were randomly selected from a manually collected dictionary.

[4]`http://en.wikipedia.org/wiki/List_of_digital_camera_brands`
`http://en.wikipedia.org/wiki/List_of_colors`

Method	Feature	Training Data	Model
List-rand	Section 5.3.1.1	Random	Classification
List-basic	Section 5.3.1.1	Section 5.3.2	Classification
LT-basic	Section 5.3.1.1 and 5.3.1.2	Section 5.3.2	Classification
Lyretail	Section 5.3.1.1 and 5.3.1.2	Section 5.3.2	Section 5.3.3

Table 5.3: Methods for the lesion study.

Lyretail uses a mix of code from several languages and projects. The core Lyretail code is in Python. We used an existing HTML parsing tool, BeautifulSoup, to obtain all the elements from an HTML webpage. Our page-specific extractor was implemented based on the Mallet Java library.[5] We also use the Python scikit-learn library for its logistic regression method.[6]

5.4.1 Page-specific Extraction

In this section, we evaluate the performance of the *page-specific extractor*. We also show the result of extracting PSDs at multiple granularities.

We prepared the data sources as follows. We followed the seed selection process described earlier. the three randomly selected seeds for each category are shown in Table 5.2. For each category, we sent the three seeds to Google and obtained top-100 webpages as the input webpages. We skipped webpages that cannot be downloaded and those that did not satisfy the recurrent pattern criteria mentioned in Section 5.2.2. Also, we only kept webpages containing more than three dictionary instances as the evaluation set. In the end, we kept 444 webpages (56.4% of the total) for all the categories. We then asked human experts to identify all of the correct dictionary entities on a webpage for each of the 11 categories.

Methods — We compared the following methods:

- Seal uses the wrapper of Seal [108] to obtain the PSD.[7]

[5]Mallet: `http://mallet.cs.umass.edu`
[6]scikit-learn: `http://scikit-learn.org/`
[7]Seal's code: `https://github.com/TeamCohen/SEAL`.

- SealDict uses Seal [108] to first generate a dictionary (from multiple pages), then constructs the PSD as the intersection between the Seal dictionary and entities on a target webpage.

- Lyretail is our proposed page-specific extractor as described in Section 5.3.3.

In addition we compare Lyretail with the following three methods to study the influence of each component in the Lyretail framework. The detailed configurations of the three methods can also be found in Table 5.3. We used logistic regression classification for all three basic methods. We also tried other classification methods (*e.g.*, SVM and decision tree), which performed comparably or worse.

- List-rand is a logistic regression classifier based on the web lists features. It randomly selects negative examples from the entities on the webpage excluded by the positive examples.

- List-basic is a logistic regression classifier based on the web lists features. It uses our distant supervised training data as described in Section 5.3.2.

- LT-basic is a basic version of our page-specific extractor: it is a logistic regression classifier using both the webpage-parameterized and web lists features.

We report the averaged per-page F1 score for each category, and we tried a set of settings for the above 5 methods: For SealDict, we vary the dictionary size from 10 to 2000 and report the best F1 score. For List-rand, we randomly selected the negative examples for 10 times and report the averaged per-page F1. For List-basic, LT-basic and Lyretail, we obtained the top 20 items using Seal as the initial dictionary, and tested 7 settings from 0 to 0.01 for the negative example threshold and chose the best per-page F1 score.[8]

[8]We choose the top 20 items produced by Seal as the initial dictionary to maintain a high-precision though low-recall set. The averaged precision of the initial dictionary is 0.999 for common vocabularies and 0.746 for long-tail vocabularies by randomly picking seeds from the ground-truth for 10 rounds.

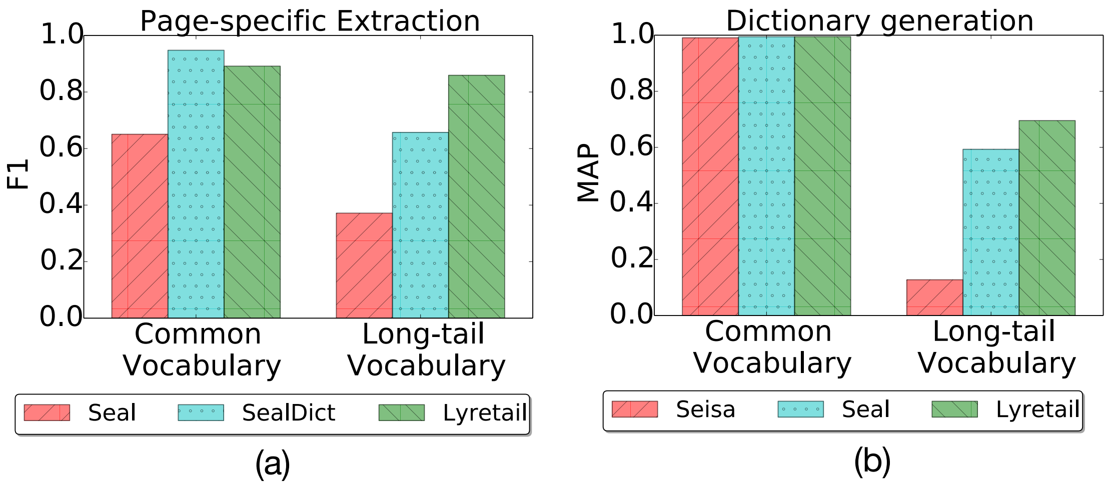


Figure 5.4: The summarized performance on Lyretail's page-specific extraction and dictionary generation on common and long-tail vocabulary settings.

Performance — We now compare Lyretail with Seal and SealDict, simple methods based on Seal. Figure 5.4 (a) shows the average F1 performance for Lyretail's page-specific extraction on common and long-tail vocabularies. As shown in the Figure, we can see that Lyretail (with an average F1 of 0.892) is able to produce high-precision, high-recall PSDs in almost all the categories. Moreover, we achieved comparable performance in the common vocabulary settings and obtained 30.7% improvement in the long-tail vocabulary settings. It demonstrates that Lyretail is able to construct high-quality page-specific extractors that can obtain even infrequently-observed dictionary entities.

Figure 5.5 shows the detail results for the averaged F1 of the three methods on the five long-tail vocabulary categories. SealDict (with the averaged precision of 0.827 and recall of 0.569) loses to Lyretail because SealDict uses the dictionaries generated by Seal, thus failing to recognize some out-of-set items on webpages. We noticed that many extraction errors of SealDict is due to failing to detect infrequent observed dictionary items and also items that can be written in many different ways. For example, "New York Yankees" can also be written as "NY Yankees". For the

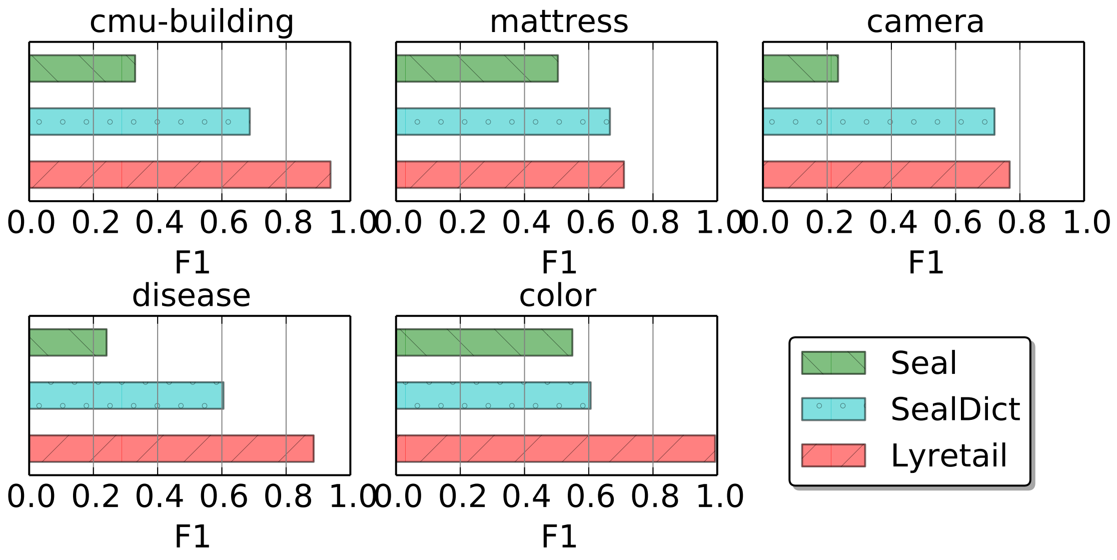


Figure 5.5: The F1 performance of the page-specific extractors in long-tail vocabulary categories. We compare Lyretail with simple methods based on the previous work Seal.

six common vocabulary categories (*e.g.*, mlb-team and us-states), our system always beats Seal, but loses slightly to SealDict. The reason is that Lyretail attempts to infer *true* dictionary entities based on a small initial dictionary of size 20, while SealDict uses an already high-quality dictionary to find any matching dictionary entities on the webpage.

Lesion Study — We now examine which components of Lyretail are the most influential, by comparing Lyretail with three basic versions of the page-specific extractor: List-rand, List-basic and LT-basic. Figure 5.6 shows the F1 performance of the four methods in long-tail vocabulary settings: Lyretail is better than the other three methods. For the six common vocabulary settings, all four methods perform similarly. It demonstrates that our mechanism of training data generation and co-training framework is effective for building a high-quality page-specific extractor.

Granularity — We now demonstrate our page-specific extractor is able to produce different granularities of the PSD, by varying the negative example threshold from 0

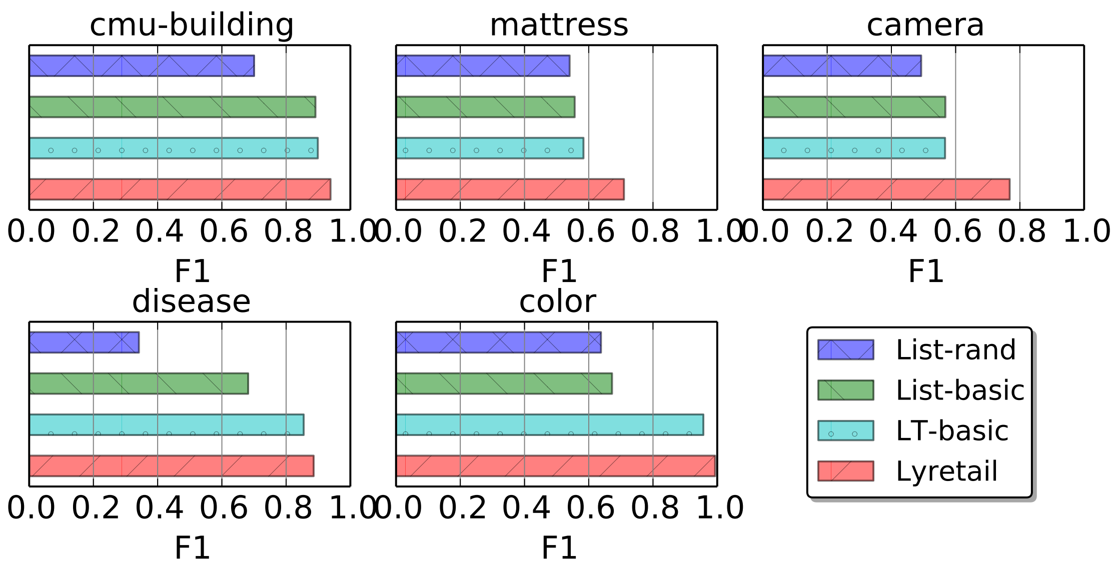


Figure 5.6: The F1 performance of the page-specific extractors in long-tail vocabulary categories. We compare Lyretail with two baseline methods.

to 0.3.

Figure 5.7 shows the dictionary sizes under each setting of the negative example thresholds on two webpages, rantsport.com and bestchoicemattress.com. As shown in the Figure, we identified five semantically meaningful plateaus out of nine granularities, and measured the page-specific precision and recall for each identified granularity. For example, the granularity at the 2nd plateau is identified to be MLB teams, with page-specific precision and recall both equal to 1. It indicates that at this granularity, we correctly obtained all the available MLB teams on rantsport.com.

By varying the negative example threshold, we observe that the plateaus exist in almost every webpage, though not all the granularities generated are semantically meaningful.

In summary, the page-specific extractor is able to produce a high-quality dictionary per webpage, especially in long-tail vocabulary settings. Also, it is able to present meaningful dictionary granularities.

5.4.2 Dictionary Generation

To evaluate Lyretail's ability to generate CDs, we again selected three random seeds for each of the 11 dictionary categories. We followed the seed selection process as descried earlier, and we fetched top-100 webpages for each set of seeds from Google. We repeated this seed selection and webpage fetching process 10 times and report the averaged precision for top-k results.[9] We removed the seeds if none of the methods can produce any results (as was the case for 2 seed selections for cmu-building and 1 seed selection for color). For each instance in the top-k result, we asked human experts to determine whether it belongs to the dictionary.

Methods — We compared the following methods:

- Seal, a previous set expansion algorithm [108].
- Seisa, another previous set expansion algorithm [60].
- Lyretail, our proposed system.

We also compare Lyretail with List-rand, List-basic, and LT-basic for a lesion study. List-rand, List-basic, and LT-basic are all built on the Lyretail framework but use the List-rand, List-basic, and LT-basic page-specific extractor, respectively.

We report the averaged precision for top-k results over 10 times, and we set up the above five methods as follows: For Seal, we directly used its code from Github. It is not possible to compare to Seisa directly, because their original web list dataset is not available; we reimplemented the Seisa's algorithm on our web crawled HTML lists. For List-basic, LT-basic, and Lyretail, we obtained the top 20 items using Seal as the initial dictionary, and set the negative example threshold to be 1e-6.

Performance — We now compare Lyretail with the two previous techniques, Seal and Seisa. Figure 5.4 (b) shows the mean averaged precision (MAP) for Lyretail's

[9]We do not use the actual recall because it is usually difficult to enumerate the universe of instances for every category.

dictionary generation on common and long-tail vocabularies. As shown in the Figure, we can see that in common vocabulary settings, Lyretail matches Seal and Seisa; in long-tail vocabulary, Lyretail substantially outperforms those other methods by 17.3%. It demonstrates that Lyretail is able to construct high-quality dictionaries in both common and long-tail vocabulary settings.

Figure 5.8 shows the detailed results for the precision of the top-k results of the three methods on the five long-tail vocabulary categories. For the other six categories (*e.g.*, mlb-team and us-states), there is not much difference among the three methods. These are relatively small vocabularies, and existing methods largely do well at them. Lyretail is able to match the previous systems' performance.

Lesion Study — We now examine which components of Lyretail are the most influential, by comparing Lyretail with the three baselines, List-rand, List-basic and LT-basic. Figure 5.9 shows the precision of the top-k results for the four methods in long-tail vocabulary settings: Lyretail outperforms the three baselines. For the other six common vocabulary settings, all four methods perform similarly. Again, it demonstrates that Lyretail's good results are primarily due to our training data and extractor construction mechanism.

In summary, we have demonstrated that Lyretail is able to generate high-quality dictionaries. Especially in long-tail settings, Lyretail has obtained 17.3% improvement on mean averaged precision (MAP) for the dictionary generation process and 30.7% improvement on F1 for page-specific extraction, comparing to the state-of-the-art methods.

5.4.3 System Configuration

In this section, we evaluate the influence of three factors that influence the quality of CD for Lyretail: the number of iterations, the negative example threshold, and the initial dictionary size. We used the same setting as in Section 5.4.2.

Iterative Process – First we evaluate the influence of Lyretail's iterative framework on the dictionary quality. We used the same setting as in Section 5.4.2 (initial dictionary size of 20 and negative example threshold of 1e-6), but emit dictionaries of 5 iterations. The performance increases slightly when we do multiple rounds of iterations, but the differences of precision for top-k results are not huge at less than 0.05. During the iterations, the initial dictionary is the only changed input, and it indicates that the changes on the 20 initial dictionary entities are not huge enough to influence the performance significantly.

Negative Example Threshold — We tried a set of settings for the negative example threshold λ_d from 0 to 0.1 on all categories. We observed that a lower λ_d — perhaps unsurprisingly — tends to lead to a lower precision for highly-ranked values of k, but higher recall later. In most of the cases, the differences are not huge at less than 0.1, But the precision can drop significantly in categories such as camera and color. For example, the top-100 precision dropped by 0.2 by changing λ_d from 0.005 to 0.1 in camera. However, the differences of the precision are not significant at less than 0.1 when we varying λ_d between 0 and 1e-5.

Initial Dictionary Size — To evaluate the influence of the initial dictionary size on the dictionary quality. We tried a set of settings for the initial dictionary size from 10 to 500 on all categories, and measured the precision of the top-k results. We observe that the performance tends to be better when we use a high-precision initial dictionary of a larger size. But, the differences of the precision on top-k results are not huge at less than 0.1, with the exception of color, where the precision of the top-1000 results increases by 0.12 when changing the initial dictionary size from 10 to 20.

5.5 Related Work

The goal of Lyretail is similar that of research into ***set expansion***, including Seal [108, 109, 110, 92] and Seisa [60]. Seal uses a simple pattern matching (*i.e.*, suffixes and prefix-based extractors) method to obtain candidate entities from many webpages, then employs a graph-based random walk to rank candidates entities according to their closeness to the seeds on the graph. As a result, the method is able to accurately identify frequently mentioned items but fail to detect infrequently mentioned long-tail items. Seisa is an iterative algorithm to aggregate dictionaries using the co-occurrence frequency between each pairwise dictionary item in the web HTML lists dataset and Microsoft query logs. The method relies entirely on the co-occurrence frequency information, and thus also tends to ignore infrequently-observed entities. Rong *et al.* [92] focuses on distinguishing multi-faceted clusters of the expanded entities by fusing ego-networks and existed ontology. In contrast, we build high-quality page-specific extractors to better identify long-tail dictionary items.

The series of work on ***wrapper induction*** [46, 74, 41] can be used for page-specific extraction, but these methods are rule-based and are inapplicable to produce high-quality page-level webpage extractors in our cases. More recently, Dalvi *et al.* [43] proposed the framework for wrapper induction on large scale data, and Pasupat and Liang [84] developed page-specific models to extract entities of a find-grained category. However both methods require a decent amount of training data while we at most have a few seeds.

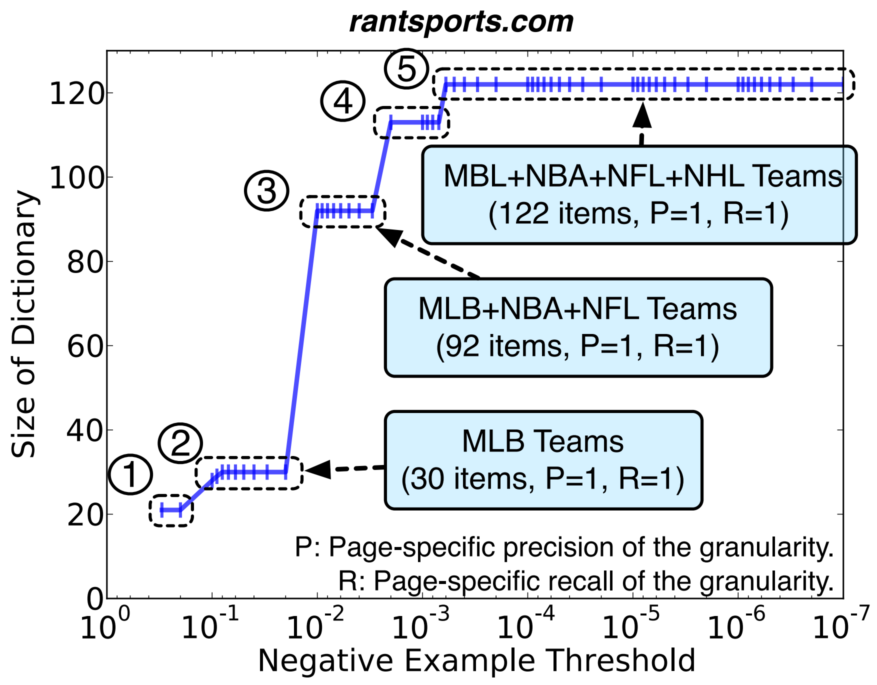
Our page-specific extractor is similar to ***co-training*** work [16, 39, 29, 66], which exploits conditionally-independent features to improve classification performance. Unlike past work, our page-specific extractors contain sequential properties and it is not practical to train it with only partial labeled data.

Our page-specific extractor uses a ***distant supervision*** based method. Distant supervision based method has been widely used on entity and relationship extraction from the web [9, 51, 81, 115], and these work derive training examples from predefined heuristics or from existing KBs. However, those approaches are not practical in our case: heuristics cannot capture the semantics of the entities; and in the existing KBs, it is hard to find an entry for a great number of the entities on the webpages due the KBs' poor coverage [47]. Instead we uses a noisy but large-scale web list dataset to derive the negative training data. Hoffmann, *et al.* [61] also uses web HTML lists but their goal is to learn semantic lexicons in order to expand the training data.

5.6 Conclusions and Future Work

In this paper, we have demonstrated that Anthias is able to generate high-quality dictionaries: as good as previous work in the case of small vocabularies, and substantially better than previous work in large vocabulary settings. In the future we aim to improve Anthias by exploiting the dictionary information in novel ways, such as new search engines that combine data search with some elements of data integration systems.

In the future we aim to improve Anthias in several ways. First, we will move beyond the simple "seeds-only" model that many dictionary systems have used, to try to incorporate more flexible but still succinct domain knowledge from the user. Second, we would like to extend the input datasets to include nontraditional sources of dictionary information, such as spreadsheets, relational databases, and even social media utterances. Finally, we are looking to exploit the dictionary information in novel ways, such as new search engines that combine data search with some elements of data integration systems.



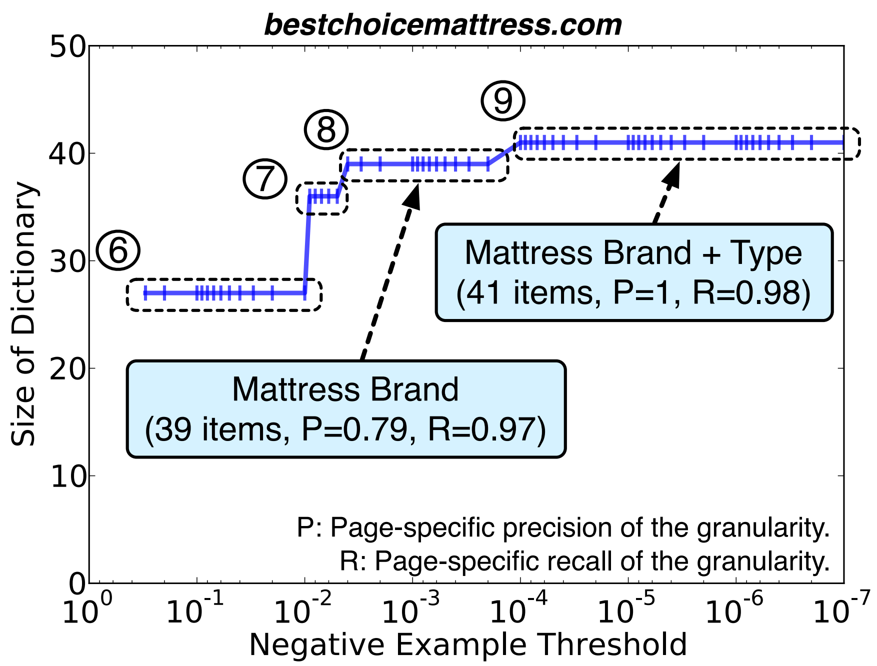


Figure 5.7: The granularities of the dictionary produced from two websites by trying a set of negative example thresholds.

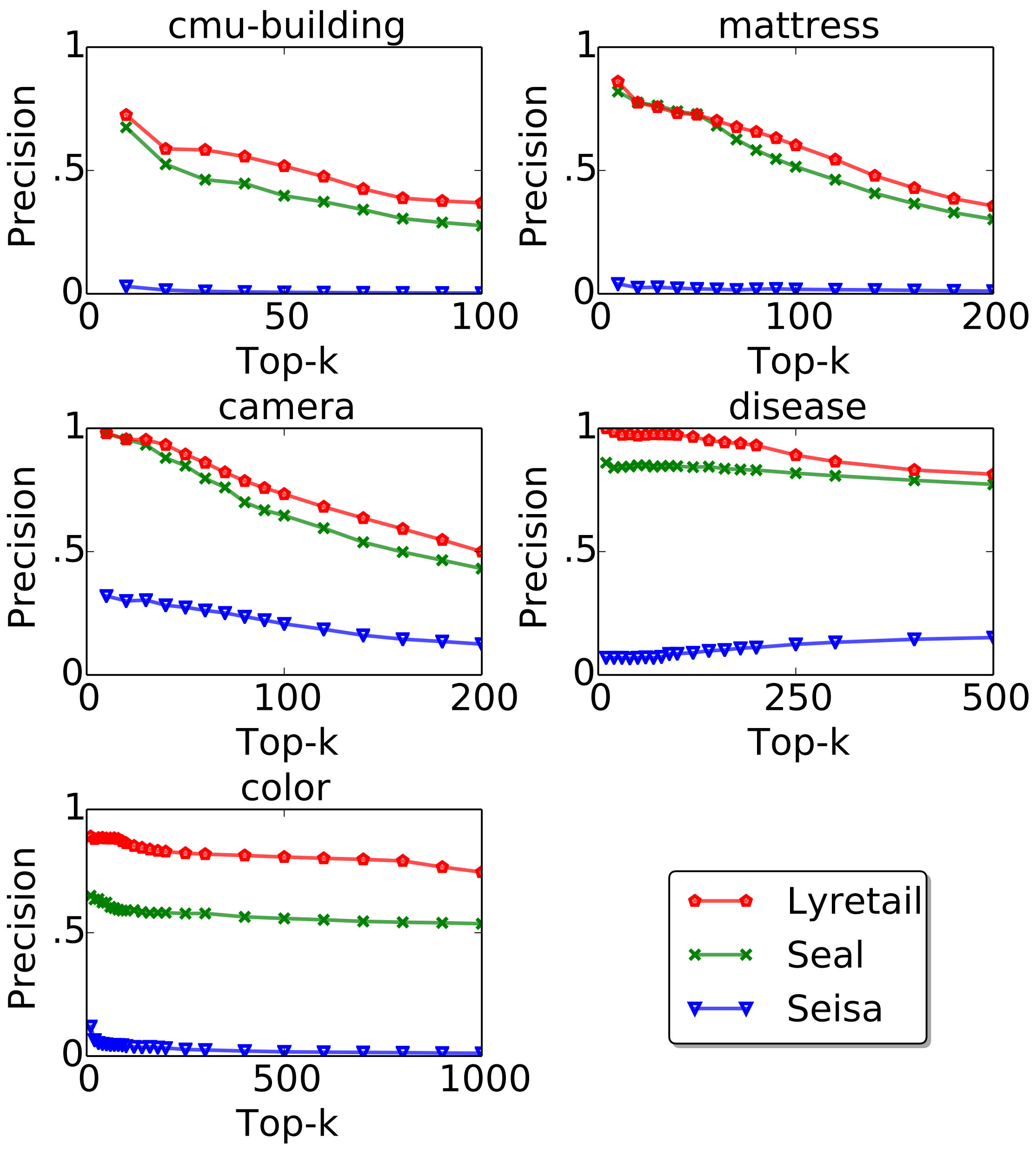


Figure 5.8: The precision of the top-k dictionary. We compare Lyretail with two previous methods.

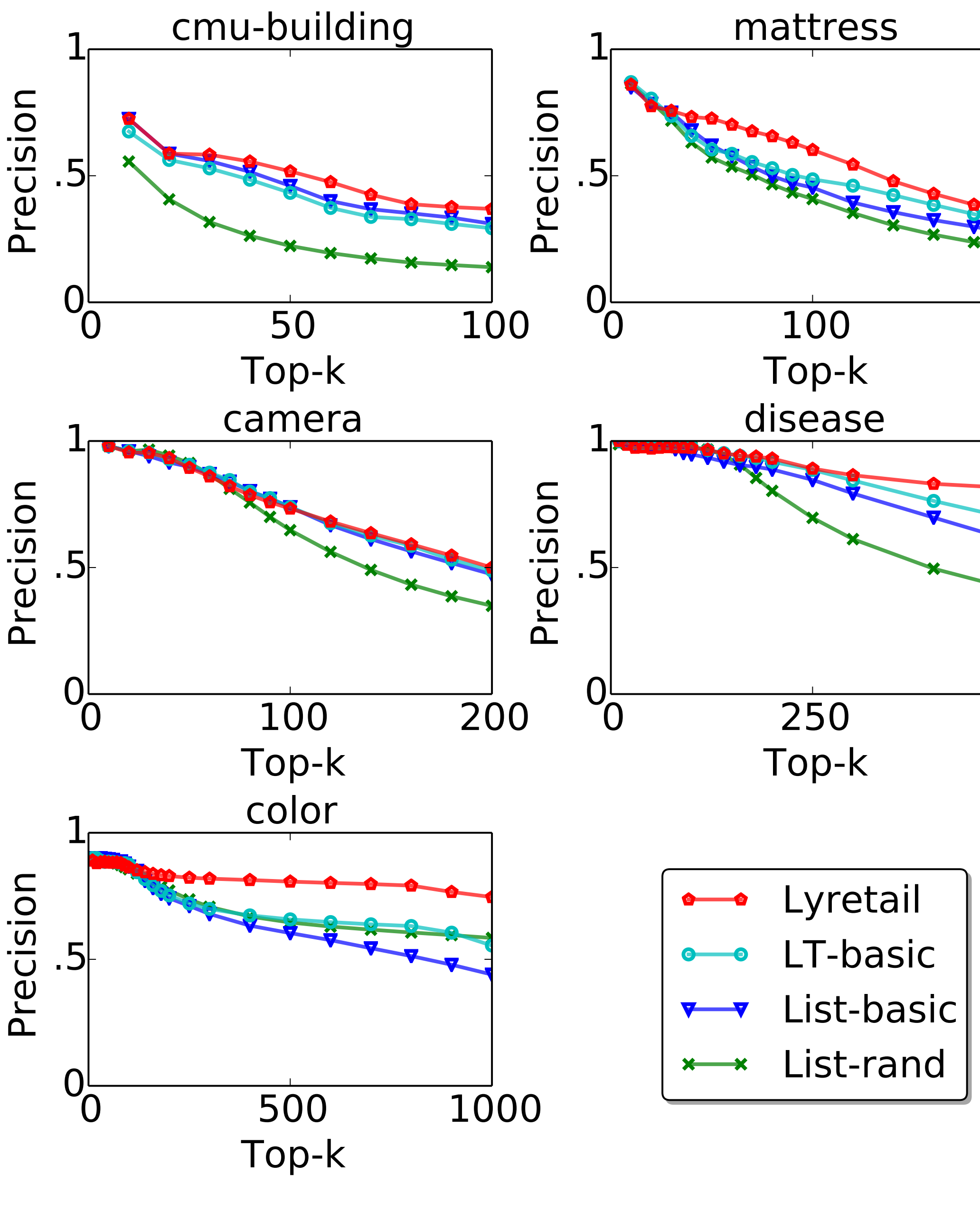


Figure 5.9: The precision of the top-k dictionary. We compare Lyretail with two baseline methods.

CHAPTER VI

DiagramFlyer: Extraction on Diagrams from PDFs

6.1 Problem Overview

Data-driven diagrams (or statistical graphics) are an important type of web statistical data for communicating complex information. Diagrams, a stylized mixture of graphics and text, offer succinct quantitative summaries of data that motivate the overall document's content. For many technical documents, the diagrams may be readers' *only* access to the raw data underlying the documents' conclusions. Especially for quantitative disciplines such as finance, public policy, and the sciences, certain diagrams could be even more valuable than the surrounding text.

In this dissertation, we focus on the 2-D diagrams, but we observe two *distinctive properties* that serve as the important design criteria for building DiagramFlyer. First, there are a variety types of 2-D diagrams in the web, including scatter plots, bar chars, line plots, and so on, and the rise of visualizations was also witnessed in the web, ranging from political art projects to New York Times stories. Second, even though data-driven diagrams are often consistently stylized, the diagram metadata is completely implicit. Also some textual regions, especially legends and titles, do not reliably appear in one location. For example, Figure 6.1 shows that the regions of diagram text can appear in surprising locations: Figure (a) has its legend on the right-hand side, whereas Figure (b) has it above the title.

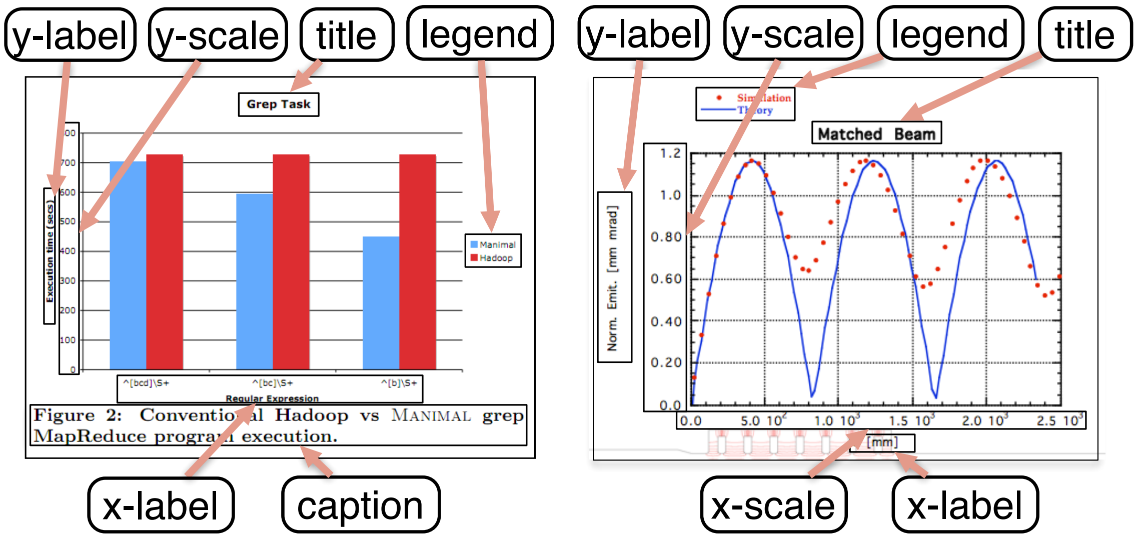


Figure 6.1: A diagram contains several characteristic regions of text as metadata, and the position of metadata may vary in different diagrams.

In this chapter we present DiagramFlyer, a search system that is able to automatically transform a relational table to diagrams, while allowing users to interactively personalize the visualizations with little user effort. DiagramFlyer consists of two main components, *extract* and *search*. First, the *extract* discovers the implicit schema information from a large scale of web data and constructs a visualization-specific knowledge base, which can then be used to automatically generate high-quality visualizations. Second, the *search* enables searching visualizations by their latent schema information. At the same time, the *search* should allow users to personalize the automatically generated visualizations while requiring little effort from users.

In the rest of the paper, we will give an overview of the system architecture in Section 6.2, demonstrate the working system in Section 6.3, and conclude with a brief summary of the technical problem the system addresses in Section 6.4.

6.2 System Framework

In this section, we introduce DiagramFlyer's diagram extractor, query interface and the software architecture.

6.2.1 Diagram Metadata Extraction

The *diagram extractor* attempts to find all the elements of a diagram's graphical specification. As mentioned earlier, we extract 8 types of diagram metadata: `title`, `legend`, `caption`, `scale` and `label` for both the `x-` and `y-`axes, and `type`. We obtain a diagram's type (a bar, line, or scatter chart) using the classification method proposed by ReVision [96].

Even though data-driven diagrams are often consistently stylized, obtaining correct labels can be challenging for several reasons. Some textual regions, especially `legend`s and `title`s, do not reliably appear in one location. For example, Figure 6.1 shows that the regions of diagram text can appear in surprising locations: Figure (a) has its `legend` on the right-hand side, whereas Figure (b) has it above the `title`. Also, multiple sub-diagrams can appear within a single image; for example, diagrams could be stacked to share one `caption`. In a randomly selected test collection (379 diagrams), more than 36% of diagrams are small ones contained within a single larger diagram image.

In order to process as large a range of diagrams as possible, our *diagram extractor* focuses entirely on the text embedded in each diagram; we ignore the graphical contents of the image, apart from the texts' 2D positioning. The *diagram extractor* uses a three-stage pipeline.

Segment Recovery — In **segment recovery**, crawled PDFs are processed by the JPedal [68] PDF processor, which produces a stream of individual extracted words, each with a two-dimensional bounding box of coordinates. Because this processing is applied on the entire document, there are many spurious words pulled out that are not part of genuine diagrams.

Extracted word boxes are grouped into sequences, termed *segments*, which we hope reflect semantically coherent regions of text. This step is based on the proximity and orientation of nearby word boxes. Some segments will simply be lines of text from the body of the document; others will be diagram components such as the `caption` or `context`.

Finally, we remove the bulk of segments that have nothing to do with diagrams. We do so by first locating segments that strongly indicate a diagram, such as a sequence of values that indicate a potential axis scale, or a line of text that contains the word "Figure" or "Fig". For each identified signal, we use a "sliding window" to obtain close segments before and after as a *diagram group.* For those that overlap, we merge them as one *diagram group.* We dispose of all segments that are not in such a group.

Metadata Classification — The **metadata classification** assigns each segment in a diagram group with one of the eight previously-mentioned labels. To identify these segments, we apply two trained classifiers in sequence. The first, *simple* classifier, is fast but generates false positives. It is based on purely textual features derived from each segment, as shown in Table 6.1.

To further refine our dataset we now require that each diagram group more strongly reflect qualities of a true diagram. Each group *must* contain either `label` or `scale` information for both the `x-` and `y-` axes. Segments in diagram groups that do not meet this relatively low bar are probably not part of a true diagram and are filtered out. As shown in our experiments, this filter is effective at raising the final output quality, removing 15 non-diagram segments for every 1 diagram segment that we incorrectly remove.

The final *position-sensitive* classifier again assigns a label to each segment. It uses the initial *simple* labels to derive the label- and geometry-sensitive features listed in

Does segment text start with a capitalized word?
Percentage of words in segment that are capitalized
Percentage of tokens in segment that are numbers
words in segment
Ratio of segment width to height
Does segment start with the word "Fig"?
Does segment contain a capitalized word after "Fig"?
Percentage of words in segment that are nouns
Percentage of words in segment that are verbs

Table 6.1: Selected textual and segment-centric features used in the *simple* classifier.

Segment distance to origin
Angle of segment to origin
Segment dist. to nearest upper `x-axis scale`
Segment dist. to nearest left `y-axis label` or `scale`
Segment intersects with upper `x-axis scale`?
Segment intersects with left `y-axis`?
Direction to nearest diagram box
% of segments in diagram group sharing left boundary
Is segment the leftmost in diagram group?
Is segment underneath a `caption`?

Table 6.2: Selected positional features used in the *position-sensitive* classifier.

Table 6.2. For example, the geometric intersection of segments with `x-axis scale` and `y-axis scale` labels is called the *origin*, which can then be used to compute the first two *position-sensitive* features. These refined labels are more accurate than when using *simple* alone.

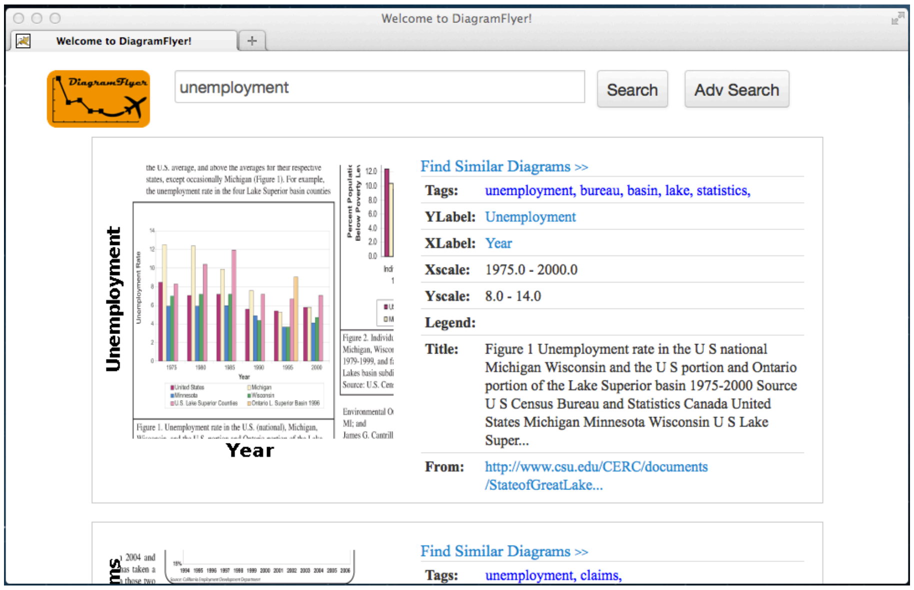
Diagram Regrouping — In this last stage of extraction, we use the new segment labels to perform **diagram regrouping**. This step is relevant when a *diagram group* contains multiple smaller diagrams. In this step, we refine these groups so that they contain just a single diagram. This method relies on finding a bounding box formed by the axes of each true diagram, then assigning segments to the nearest bounding box. The process depends on the semantic label assigned in the previous step; for example, a `caption` will be assigned to the bounding box above rather than below. The final output from this stage, and thus the *diagram extractor* overall, is a set of single diagrams with labeled text segments.

6.2.2 Query Interface and Language

Prior to any search engine activity, the web's population of users produces diagrams. Diagram generation is a unique process. First, an underlying database has to be collected ahead of time, then a customized graphical specification has to be designed (either through a specification or direct manipulation), then finally the specification is "compiled" to render the diagram images. The graphical specification describes how to visualize the contained elements that define the structure of the diagram. For example, as mentioned in [112], given a database, a user must specify the data variables for both x- and y- axis, the transformation of the variables, the scale of axes (log or linear), and other characteristics, in order to generate a two-dimensional scatterplot.

DiagramFlyer attempts to extract all the necessary elements of the graph generation specification for the diagrams. We call this specification a *diagram template* or *diagram metadata*. To be more specific, we try to find all textual and visual elements that are necessary to render the final images. In our prototype we focus on 8 key fields that can be used to generate a unique diagram image: `x-label`, `x-scale`, `y-label`, `y-scale`, `title`, `legend`, `caption`, `scale` and `type`. (`Type` identifies what kind of two-dimensional chart it is: bar, line, scatter or other.) For example, Figure 6.1 shows two sample data-driven diagrams and the diagram metadata that Anthias found in each. These diagrams, plus the accompanying diagram metadata, form the corpus our search engine will index.

DiagramFlyer's interface is similar in appearance to traditional web search engines, accepting input into a search box (or boxes in the faceted "advanced" mode) and presenting the results using a top-10-style Search Engine Results Page (SERP). Figure 6.2 shows the current DiagramFlyer prototype SERP, with a query for `unemployment` and



unemployment

Find Similar Diagrams >>

Tags: unemployment, bureau, basin, lake, statistics,

YLabel: Unemployment

XLabel: Year

Xscale: 1975.0 - 2000.0

Yscale: 8.0 - 14.0

Legend:

Title: Figure 1 Unemployment rate in the U S national Michigan Wisconsin and the U S portion and Ontario portion of the Lake Superior basin 1975-2000 Source U S Census Bureau and Statistics Canada United States Michigan Minnesota Wisconsin U S Lake Super...

From: http://www.csu.edu/CERC/documents /StateofGreatLake...

Find Similar Diagrams >>

Tags: unemployment, claims,

Figure 6.2: The user interface of the DiagramFlyer system.

one visible hit. A score for each retrieved image is calculated by combining the similarity of each individual fields (*e.g.*, how well do the x-labels match? how much does the x-scale overlap?, and so on). We will discuss the scoring mechanism in detail in Section 6.2.

DiagramFlyer's query language supports complex, face-ted queries which allows end-users to create highly targeted searches. Thus, DiagramFlyer is able to support querying on features that are part of the descriptive pipeline that generated the diagram. DiagramFlyer's query language is composed of the 8 *field operators* (based on the fields described above) and a *fuzzy expansion* function. Field operators operate against a faceted index of the diagrams (each field is stored separately). For example,

Example 1. If a user wants to get diagrams about population statistics over the year 1990 to 2014, she can formulate the query as follows:

```
x-label: year AND

x-scale: from: 1990 to: 2014 AND

y-label: population
```

DiagramFlyer's query language also supports a variety of diagram search applications including "similar diagram" search. DiagramFlyer supports fuzzy matching. When processing a search query, a unique component of DiagramFlyer is its *query expander*, which is able to expand the query by generating semantically similar terms. The goal of lexicon expansion is to retrieve diagrams using information that may have been removed as a side effect (either intentionally or not) in the diagram production process. For example, given a term "Michigan", the *query expander* could recognize that terms such as "California" and "Wisconsin" are highly relevant terms belonging to the same category. This lexicon was built by analyzing hundreds of millions of web pages, which we will discuss in detail in Section 6.2.

Example 2. After finding a relevant diagram *d_q*, a user may want to retrieve all the relevant diagrams that could potentially be generated from the same underlying dataset. The query can be formulated as:

```
x-label:expand(d_q.x-label) AND
y-label:expand(d_q.y-label) AND
title: d_q.title AND
caption: d_q.caption
```

6.2.3 Software Architecture

The DiagramFlyer system proceeds in two stages. In an initial *offline* stage, it processes a corpus of diagrams and prepares them for search. In the subsequent *online* stage, DiagramFlyer offers three distinct methods for giving users access to the diagrams.

The system architecture is seen in Figure 6.3. It employs a pipeline of offline corpus-processing steps that produce output then used by an online search query system. The ***offline*** pipeline has three components.

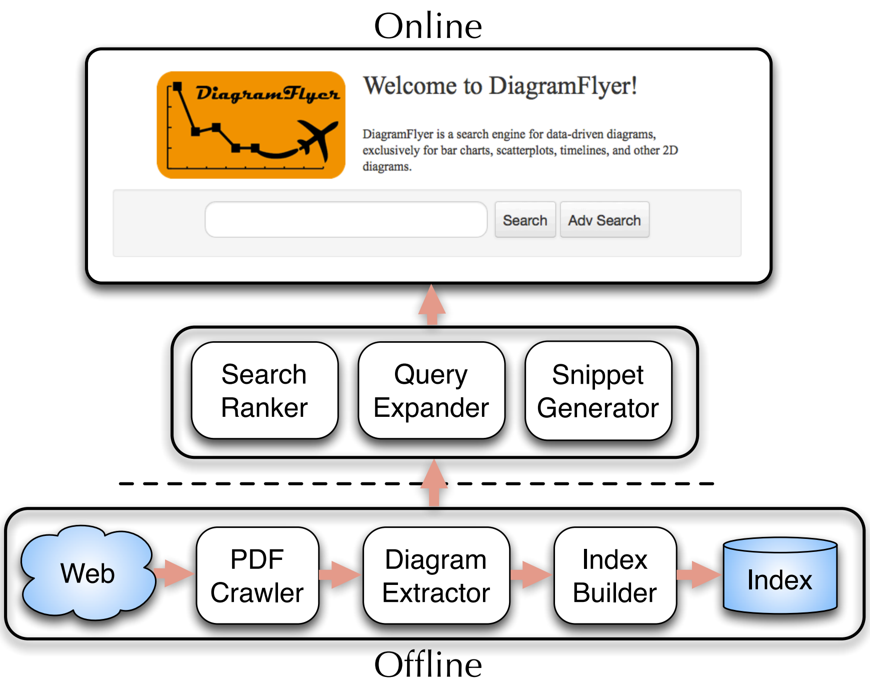


Figure 6.3: DiagramFlyer's data processing pipeline.

First, the ***PDF crawler***, which is based on the Nutch open-source crawler [25], downloads a large number of PDFs from public web pages on academic Internet domains. In our current testbed we concentrate on diagrams extracted from web-hosted scientific PDFs (found by targeting .edu websites). We found 153K documents. We focus on PDFs that contain diagrams with explicit text. Thus, we avoid the use of optical character recognition (OCR) software. Although the quality of many OCR systems is reasonably high for basic document types such as book pages and business cards, the significant modification they require to function on diagrams is beyond the scope of this paper.

These PDFs are then fed to the ***diagram extractor***. This extractor identifies all the diagrams in the corpus and extracts their metadata at the same time. The system extracted 319K diagrams (*i.e.*, slightly more than 2 diagrams per paper). For the testbed system we target two-dimensional data-driven plots (including scatter, time series, and bar plots) as these have been found to represent a large portion of data-driven diagrams (*e.g.*, 70% of diagrams found in news magazines [105] are time-series).

Finally, the ***index builder*** uses Lucene [7] to construct an inverted search index over the extracted and annotated diagrams. The index tracks each extracted field separately so that keyword matches on individual parts of the diagram can be identified.

All three parts of the ***online*** query system are implemented in Java running as a web application, using Lucene for query processing during inverted index retrieval. They are the *search ranker*, *query expander*, and *snippet generator*.

Search Ranker — Given a keyword query, the *search ranker* computes a relevance score for each diagram and presents a ranked list of diagrams as the results. We implemented the scoring mechanism, *weight-rank*, for assessing a diagram's relevance to a user's query using Lucene [7]. The *weight-rank* mechanism looks for matches in each distinct metadata field of a searching diagram, then computes the standard TF-IDF relevance score of each metadata field. It allows each of the eight fields to have a different weight when computing the total diagram relevance score. We obtained the weights by using a Support Vector Machine to find the optimal weight assignment based on a supervised training set of more than 430 human-annotated *(query, diagram, relevance)* triples. By separately finding search hits among fields that are distinctive and meaningful, a ranking system has greater ability to assign useful (and different) weights to each field. The ranker then sorts diagrams according to their relevance scores and presents the top results to the user.

Query Expander — The *query expander* extends the query to retrieve a broader range of relevant diagrams. We aim to recover diagrams that are related to a target diagram but have a connection obscured by the lossy production pipeline.

The main component of the *query expander* is a lexicon generator built on 14 million HTML lists crawled from ClueWeb09 [37]. We used the lexicon generation algorithm proposed in [60]. Given a term, a lexicon generator produces a ranked list of terms belonging to the same category. For example, the lexicon generator might take "Michigan" as input and emit many other states in the US. The web list dataset is not domain specific, and thus we believe our lexicon generator can cover a large number of different topics.

The *query expander* chooses a single expansion for each user query term (to avoid the resulting search query to strongly favor one term over another). The lexicon entry we choose for the expansion of query term t_s will be the lexicon term that maximizes the lexicon similarity score the lexicon similarity score $S_{Lex}(t, t_s)$. Given a query term t_s, let $L = Lex(t_s)$ be its generated lexicon and t be a term in a searching document d. A direct way to measure the semantic similarity between t_s and t is to measure the probability of how often t_s and t co-occur in the same list, as $S_{co}(t_s, t)$. But web lists are noisy, so directly using $S_{co}(t_s, t)$ to represent how close the two terms are can be misleading. Thus we compute t's lexicon similarity to t_s based on two parts. Similarity of the document term to the query term and to the query term's overall generated lexicon:

$$S_{lex}(t, t_s) = S_{co}(t, t_s) + \frac{1}{|L|} \sum_{t' \in L} S_{co}(t, t') \tag{6.1}$$

Snippet Generator — Finally, the *snippet generator* generates a brief visual summary of each search hit in the SERP, as shown in Figure 6.2. Textual snippets in traditional web search are a query-relevant compact document representation that help users scan the result list quickly and find high-quality matches. To achieve these goals in DiagramFlyer, we annotate a thumbnail image (*i.e.*, a scaled image of the diagram) with diagram metadata. We found the annotation to be useful as the text in a scaled-down thumbnail image is often too difficult to read. By overlaying text in a larger font on top of the diagram thumbnail we allow the end-user to quickly identify good matches in the SERP list.

6.3 Demonstration

The online demo video is available on YouTube[1]. We demonstrate the working data-driven diagram search system DiagramFlyer via the following three functions.

Keyword Search — First, DiagramFlyer supports keywords queries. For example, when a user types the search query "birth rate" in the search box, she is able to browse a ranked list of diagram objects. The snippet (as shown in Figure 6.2) presents all the extracted elements of a diagram specification for fast browsing. In addition, the user can reach back to the original document by clicking its URL.

Advanced Facet Search — The DiagramFlyer also supports querying by the diagram generating process. By clicking the "Adv Search" button on the search interface (as shown in Figure 6.2), a user can query by the diagram template language (as shown in Example 1 & 2). For example, a user can easily obtain diagrams with "year" to be the x-axis from 1990 to 2000 and with "population" to be the y-axis using the "Adv Search" interface.

Find Similar Diagrams — The query language of DiagramFlyer makes it possible to support many interesting application, including finding similar diagrams. For example, when a user gets the initial results of querying for "birth rate" related diagram, she can click a resulting diagram's x-label to find all the diagrams with a similar x-label. The user can also click "Find similar diagrams" on the top of each resulting diagram snippet (as shown in Figure 6.2) to obtain a list of similar diagrams.

6.4 Conclusions and Future Work

DiagramFlyer is a working search engine that searches 319k diagrams extracted

[1]`http://youtu.be/B7I1_o23N38`

from thousands of PDFs in the web. We have implemented the software architecture and algorithms for implementing DiagramFlyer's diagram pipeline extractor, as well as query tools that perform diagram relevance ranking, similar item finding via lexicons, and snippet generation. In addition, we have demonstrated that the system is able to perform many interesting applications, including traditional keywords search, advanced facet search, and searching for similar diagrams.

There are many ways to extend DiagramFlyer in the future. One is to improve the raw quality and range of diagram information extraction. Diagrams consist not only of textual annotations but also other features including "marks" corresponding to data values and signals of transformations (e.g., log scales) [14, 112], all of which may become query-able features when extracted. A different approach is to perform deeper levels of semantic extraction from the diagrams to summarize the diagrams. By identifying higher level "conclusions" from the text that are supported by the diagram. Thus, a diagram that compares system performance might be distilled to "System X is fastest". Finally, additional support for non-keyword based queries (e.g. similar images, correlated datasets, or sketched queries [111]) will further enhance the system.

CHAPTER VII

Conclusion and Future Work

This dissertation describes the challenges associated with para-relational data extraction and presents four working systems that contribute toward converting para-relational data into a relational format: Senbazuru, Anthias, Lyretail, and DiagramFlyer. The four projects make substantial steps toward addressing the challenges posed in extracting the implicit semantics of para-relational data, with little user assistance. We summarize our contributions in Section 7.1, and suggest future work in Section 7.2.

7.1 Contributions

We developed four working systems that contribute to the conversion from para-relational data to the relational format. Each system addresses a specific type of para-relational data. *Senbazuru* is a prototype spreadsheet database management system that extracts relational information from a large number of spreadsheets. *Anthias* suggests an extension of the Senbazuru system to convert a broader range of spreadsheets into a relational format. *Lyretail* is an extraction system that detects long-tail dictionary entities on webpages. *DiagramFlyer* is a web-based search system that obtains a large number of diagrams automatically extracted from web-crawled PDFs. These four systems demonstrate that converting para-relational data into the relational format is possible, and also suggest directions for future systems.

7.1.1 Senbazuru Contributions

Senbazuru is a prototype spreadsheet database management (SSDBMS) that extracts relational information from a large number of spreadsheets. Doing so opens up opportunities for data integration among spreadsheets and with other relational sources. Our contributions include:

- **Novel Spreadsheet Management System** — Senbazuru is a prototype spreadsheet database management system, that extracts relational information from a large number of spreadsheets. In addition, Senbazuru supports three functional components. It allows a user to quickly *search* relevant datasets in a huge web spreadsheet corpus. It contains a background *extraction pipeline* that automatically obtains relational data from spreadsheets and has a *repair interface* that allows users to manually repair extraction errors by exploiting commonalities among errors to probabilistically reapply one user fix to other similar mistakes to minimize explicit manual intervention. Finally it supports basic relational operators, especially selection and join, which the user can apply to spreadsheet-derived relations.

- **Data Frame Extraction** — We identify that the data frame structure is common among spreadsheets and propose an extraction model to automatically recognize the data frame structure.

- **Hierarchical Structure Extraction** — To the best of our knowledge, we are the first to present a semiautomatic extraction approach to obtain the hierarchical structure accurately in spreadsheets. We propose a novel two-phase *semiautomatic* approach based on an undirected graphical model to extracting spreadsheet hierarchical structure accurately and with little user effort.

7.1.2 Anthias Contributions

Anthias envisions an extension of the Senbazuru system to convert a broader range of spreadsheets into a relational format. In particular, Anthias makes the following contributions:

- **Spreadsheet Property Extraction** — To the best of our knowledge, we are the first to propose the spreadsheet property detection problem, which is the first step toward building the spreadsheet-to-relational-table pipeline.
- **Novel rule-assisted active learning framework** — We propose a novel, hybrid, rule-assisted active learning framework to construct high-quality spreadsheet property detectors with little user labeling effort. The hybrid framework integrates an active learning approach with crude user-provided rules to save labeling effort. By using a bagging-like technique, it can tolerate bad user-provided rules.

7.1.3 Lyretail Contributions

Lyretail is a novel extraction system for constructing high-quality dictionaries with only a few user-given seeds, especially for long-tail vocabularies (*i.e.*, dictionaries that contain long-tail items).

Lyretail's contributions mainly lie in **page-specific extraction**. Lyretail builds and applies many page-specific extractors to obtain high precision and recall on long-tail items. It leverages the webpage's structural and textual information to conduct more accurate extraction on each single webpage, making it possible to detect long-tail items that rarely appear. In addition, Lyretail builds a customized *joint-inference extractor* for each of many hundreds of webpages, while requiring just three explicit seeds from the user. By combining the outputs of multiple page-specific extractors, we can produce high-precision, high-recall dictionaries.

7.1.4 DiagramFlyer Contributions

DiagramFlyer is a working search engine that provides search services for a corpus of 319k diagrams extracted from 153K web-crawled PDFs.

We summarize DiagramFlyer's contributions as follows: First, we implemented the **software architecture** and set of algorithms for implementing DiagramFlyer's diagram pipeline extractor, as well as query tools that perform diagram relevance ranking, similar items finding via lexicons, and snippet generation. Second, we demonstrate that the system can perform many **interesting applications**, including searching diagrams via keywords, advanced faceted queries to allow highly targeted searches, and searching for similar diagrams.

7.2 Future Work

Although our four systems have made substantial contributions to para-relational extraction, there is future work that can be done either on further improving the current four systems or on new directions. The future work includes:

- **Complete Pipeline for Spreadsheet-to-relational table Conversion** — We envision the spreadsheet-to-relational table transformation system using the spreadsheet property detectors mentioned in Chapter 4. We can also investigate the user-interface design to allow more effective interactions with users to conduct accurate, low-effort transformation.

- **Effective User Interaction** — There are several ways to extend Lyretail (mentioned in Chapter 5). First, we can move beyond the simple "seeds-only" model that many dictionary systems use, in order to incorporate more flexible, but still succinct, domain knowledge from the user. Second, we could extend the input datasets to include nontraditional sources of dictionary information, such as spreadsheets, relational databases, and even social media utterances.

- **New Types of Para-relational Data** — In addition to the three types of para-relational data mentioned in this dissertation, there are many other types of para-relational data, such as PDF or HTML tables. In the future, we can explore various ways to extract the para-relational data accurately from other different data sources by combining user effort and machine learning algorithms in an effective way.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L Wiener. The lorel query language for semistructured data. *International journal on digital libraries*, 1(1):68–88, 1997.

[2] Robin Abraham and Martin Erwig. Ucheck: A spreadsheet type checker for end users. *J. Vis. Lang. Comput.*, 18(1):71–95, 2007.

[3] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 85–94, 2000.

[4] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 85–94, 2000.

[5] Yanif Ahmad, Tudor Antoniu, Sharon Goldwater, and Shriram Krishnamurthi. A type system for statically detecting spreadsheet errors. In *ASE*, pages 174–183, 2003.

[6] Michael Alexander and Bill Jelen. *Pivot Table Data Crunching.* Pearson Education, 2001.

[7] Apache Lucene, http://lucene.apache.org/java/docs/index.html.

[8] Josh Attenberg and Foster Provost. Why label when you can search?: alternatives to active learning for applying human resources to build classification models under extreme class imbalance. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 423–432, 2010.

[9] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 2670–2676, 2007.

[10] Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction for the web. In *IJCAI*, volume 7, pages 2670–2676, 2007.

[11] Anders Berglund, Scott Boag, Don Chamberlin, Mary F Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. Xml path language (xpath). *World Wide Web Consortium (W3C)*, 2003.

[12] Philip A Bernstein. Applying model management to classical meta data problems. In *CIDR*, volume 2003, pages 209–220. Citeseer, 2003.

[13] Philip A Bernstein and Sergey Melnik. Model management 2.0: manipulating richer mappings. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1–12. ACM, 2007.

[14] J. Bertin. *Semiology of graphics: diagrams, networks, maps.* University of Wisconsin Press, 1983.

[15] Eckhard Bick. A named entity recognizer for danish. In *LREC*, 2004.

[16] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

[17] Scott Boag, Don Chamberlin, Mary F Fernández, Daniela Florescu, Jonathan Robie, Jérôme Siméon, and Mugur Stefanescu. Xquery 1.0: An xml query language, 2002.

[18] Olivier Bodenreider and Pierre Zweigenbaum. Identifying proper names in parallel medical terminologies. *Studies in health technology and informatics*, 77:443, 2000.

[19] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[20] Sergey Brin. Extracting patterns and relations from the world wide web. In *The World Wide Web and Databases*, pages 172–183. Springer, 1999.

[21] Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB*, pages 172–183, 1999.

[22] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[23] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: Exploring the power of tables on the web. In *VLDB*, pages 538–549, 2008.

[24] Michael J. Cafarella, Alon Y. Halevy, and Nodira Khoussainova. Data integration for the relational web. *PVLDB*, 2(1):1090–1101, 2009.

[25] Mike Cafarella and Doug Cutting. Building nutch: Open source search. *ACM Queue*, 2, 2004.

[26] Stefano Ceri, Sara Comai, Ernesto Damiani, Piero Fraternali, Stefano Paraboschi, and Letizia Tanca. Xml-gl: a graphical language for querying and restructuring xml documents. *Computer networks*, 31(11):1171–1187, 1999.

[27] Xiaoyong Chai, Ba-Quy Vuong, AnHai Doan, and Jeffrey F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD Conference*, pages 87–100, 2009.

[28] Don Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An xml query language for heterogeneous data sources. In *The World Wide Web and Databases*, pages 1–25. Springer, 2001.

[29] Nitesh V Chawla and Grigoris I Karakoulas. Learning from labeled and unlabeled data: An empirical study across techniques and domains. *J. Artif. Intell. Res.*, 23:331–366, 2005.

[30] Shirley Zhe Chen, Michael Cafareela, and Eytan Adar. Searching for statistical diagrams. In *Frontiers of Engineering, National Academy of Engineering*, pages 69–78, 2011.

[31] Zhe Chen and Michael Cafarella. Automatic web spreadsheet data extraction. In *VLDB workshop on SSW*, Trento, Italy, 2013.

[32] Zhe Chen and Michael Cafarella. Integrating spreadsheet data via accurate and low-effort extraction. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1126–1135. ACM, 2014.

[33] Zhe Chen, Michael Cafarella, and Eytan Adar. Diagramflyer: A search engine for data-driven diagrams. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pages 183–186. International World Wide Web Conferences Steering Committee, 2015.

[34] Zhe Chen, Michael Cafarella, Jun Chen, Daniel Prevo, and Junfeng Zhuang. Senbazuru: A prototype spreadsheet database management system. In *VLDB Demo*, 2013.

[35] Zhe Chen, Michael Cafarella, and H. V. Jagadish. Long-tail vocabulary dictionary extraction from the web. In *WSDM*, 2016.

[36] Zhe Chen, Sasha Dadiomov, Richard Wesley, Gang Xiao, Daniel Cory, Michael Cafarella, and Jock Mackinlay. Spreadsheet property detection with rule-assisted active learning. In *In Submission*, 2016.

[37] ClueWeb09, http://lemurproject.org/clueweb09.php/.

[38] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[39] Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *Proceedings of the joint SIGDAT conference on empirical methods in natural language processing and very large corpora*, pages 100–110. Citeseer, 1999.

[40] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artif. Intell.*, 42(2-3):393–405, 1990.

[41] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, pages 109–118, 2001.

[42] Jácome Cunha, João Saraiva, and Joost Visser. From spreadsheets to relational databases and back. In *PEPM*, pages 179–188, 2009.

[43] Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. Automatic wrappers for large scale web extraction. *Proc. VLDB Endow.*, 4(4):219–230, 2011.

[44] Ofer Dekel and Ohad Shamir. Vox populi: Collecting high-quality labels from a crowd. In *COLT*, 2009.

[45] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for xml. *Computer networks*, 31(11):1155–1169, 1999.

[46] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. *Principles of Data Integration.* Morgan Kaufmann, 2012.

[47] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, and Ni Lao. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.

[48] Pinar Donmez, Jaime G Carbonell, and Paul N Bennett. Dual strategy active learning. In *Machine Learning: ECML 2007*, pages 116–127. Springer, 2007.

[49] Gregory Druck, Burr Settles, and Andrew McCallum. Active learning by labeling features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 81–90. Association for Computational Linguistics, 2009.

[50] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2(1):1078–1089, 2009.

[51] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *ARTIFICIAL INTELLIGENCE*, 165:91–134, 2005.

[52] Michael Fleischman. Automated subcategorization of named entities. In *ACL (Companion Volume)*, pages 25–30. Citeseer, 2001.

[53] George Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of machine learning research*, 3:1289–1305, 2003.

[54] Wolfgang Gatterbauer, Paul Bohunsky, Marcus Herzog, Bernhard Krüpl, and Bernhard Pollak. Towards domain-independent information extraction from web tables. In *WWW*, pages 71–80, 2007.

[55] Mark Graves and Charles F Goldfarb. *Designing XML databases.* Prentice Hall PTR, 2001.

[56] Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *UIST*, pages 65–74, 2011.

[57] Marc Gyssens, Laks V. S. Lakshmanan, and Iyer N. Subramanian. Tables as a paradigm for querying and restructuring. In *PODS*, pages 93–103, 1996.

[58] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: the teenage years. In *Proceedings of the 32nd international conference on Very large data bases*, pages 9–16. VLDB Endowment, 2006.

[59] Haibo He, Edwardo Garcia, et al. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284, 2009.

[60] Yeye He and Dong Xin. Seisa: Set expansion by iterative similarity aggregation. In *WWW*, pages 427–436, 2011.

[61] Raphael Hoffmann, Congle Zhang, and Daniel S. Weld. Learning 5000 relational extractors. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 286–295, 2010.

[62] Eric Horvitz. Principles of mixed-initiative user interfaces. In *CHI*, pages 159–166, 1999.

[63] Vu Hung, Boualem Benatallah, and Regis Saint-Paul. Spreadsheet-based complex data transformation. In *CIKM*, pages 1749–1754, 2011.

[64] David F. Huynh, Robert C. Miller, and David R. Karger. Potluck: Data mashup tool for casual users. In *ISWC/ASWC*, pages 239–252, 2007.

[65] Zachary G. Ives, Craig A. Knoblock, Steven Minton, Marie Jacob, Partha Pratim Talukdar, Rattapoom Tuchinda, José Luis Ambite, Maria Muslea, and Cenk Gazen. Interactive data integration through smart copy & paste. In *CIDR*, 2009.

[66] Rosie Jones. *Learning to extract entities from labeled and unlabeled text.* PhD thesis, University of Utah, 2005.

[67] Frédéric Jouault and Ivan Kurtev. Transforming models with atl. In *satellite events at the MoDELS 2005 Conference*, pages 128–138. Springer, 2006.

[68] JPedal, http://www.jpedal.org/.

[69] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.

[70] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *ACM Human Factors in Computing Systems (CHI)*, 2011.

[71] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[72] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1568–1583, 2006.

[73] Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. Systemt: A system for declarative information extraction. *SIGMOD Rec.*, 37(4):7–13, 2009.

[74] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *IJCAI*, 1997.

[75] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[76] Laks V. S. Lakshmanan, Subbu N. Subramanian, Nita Goyal, and Ravi Krishnamurthy. On query spreadsheets. In *ICDE*, pages 134–141, 1998.

[77] Cheng Li, Yue Wang, Paul Resnick, and Qiaozhu Mei. Req-rec: High recall retrieval with query pooling and interactive classification. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 163–172. ACM, 2014.

[78] Gideon S. Mann and Andrew McCallum. Generalized expectation criteria for semi-supervised learning with weakly labeled data. *J. Mach. Learn. Res.*, 11:955–984, March 2010.

[79] Diana Maynard, Valentin Tablan, Cristian Ursu, Hamish Cunningham, and Yorick Wilks. Named entity recognition from diverse text types. In *Recent Advances in Natural Language Processing 2001 Conference*, pages 257–274, 2001.

[80] Andrew McCallum. Joint inference for natural language processing. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 1–1. Association for Computational Linguistics, 2009.

[81] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, pages 1003–1011, 2009.

[82] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[83] Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. Web-scale distributional similarity and entity set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 938–947, 2009.

[84] P. Pasupat and P. Liang. Zero-shot entity extraction from web pages. In *ACL*, 2014.

[85] Rakesh Pimplikar and Sunita Sarawagi. Answering table queries on the web using column keywords. *PVLDB*, 5(10):908–919, 2012.

[86] David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. Table extraction using conditional random fields. In *SIGIR*, 2003.

[87] Guo-Jun Qi, Xian-Sheng Hua, Yong Rui, Jinhui Tang, and Hong-Jiang Zhang. Two-dimensional active learning for image classification. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[88] Vijayshankar Raman and Joseph M Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[89] Lisa F Rau. Extracting company names from text. In *Artificial Intelligence Applications, 1991. Proceedings., Seventh IEEE Conference on*, volume 1, pages 29–32. IEEE, 1991.

[90] Roi Reichart, Katrin Tomanek, Udo Hahn, and Ari Rappoport. Multi-task active learning for linguistic annotations. In *ACL*, volume 8, pages 861–869, 2008.

[91] Ellen Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the national conference on artificial intelligence*, pages 1044–1049, 1996.

[92] Xin Rong, Zhe Chen, Qiaozhu Mei, and Eytan Adar. Egoset: Exploiting word ego-networks and user-generated ontology for multifaceted set expansion. *WSDM*, 2016.

[93] Sudeepa Roy, Laura Chiticariu, Vitaly Feldman, Frederick R. Reiss, and Huaiyu Zhu. Provenance-based dictionary refinement in information extraction. In *SIGMOD*, pages 457–468, 2013.

[94] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. *Readings in information retrieval*, 24(5):355–363, 1997.

[95] Sunita Sarawagi. Information extraction. *Foundations and trends in databases*, 1(3):261–377, 2008.

[96] Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. Revision: Automated classification, analysis and redesign of chart images. In *UIST*, 2011.

[97] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

[98] Alex Smola and SVN Vishwanathan. Introduction to machine learning. *Cambridge University, UK*, pages 32–34, 2008.

[99] M. Spenke, C. Beilken, and T. Berlage. Focus: The interactive table for product comparison and selection. In *UIST*, pages 41–50, 1996.

[100] Michael Stonebraker and Joseph M Hellerstein. *Readings in database systems.* Morgan Kaufmann San Francisco, 1998.

[101] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, pages 697–706, 2007.

[102] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In *Introduction to Statistical Relational Learning*, 2008.

[103] Katrin Tomanek and Udo Hahn. Reducing class imbalance during active learning for named entity annotation. In *Proceedings of the fifth international conference on Knowledge capture*, pages 105–112, 2009.

[104] Rattapoom Tuchinda, Pedro A. Szekely, and Craig A. Knoblock. Building data integration queries by demonstration. In *IUI*, pages 170–179, 2007.

[105] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 2001.

[106] 2007. http://www.di.ens.fr/ mschmidt/Software/UGM.html.

[107] Richard C. Wang. *Language-Independent Class Instance Extraction Using the Web.* PhD thesis, Carnegie Mellon University, 2009.

[108] Richard C. Wang and William W. Cohen. Language-independent set expansion of named entities using the web. In *ICDM '07*, pages 342–350, 2007.

[109] Richard C. Wang and William W. Cohen. Iterative set expansion of named entities using the web. In *ICDM*, pages 1091–1096, 2008.

[110] Richard C. Wang and William W. Cohen. Character-level analysis of semi-structured documents for set expansion. In *EMNLP*, pages 1503–1512, 2009.

[111] Martin Wattenberg. Sketching a graph to query a time-series database. In *CHI '01*, pages 381–382, New York, NY, USA, 2001. ACM.

[112] L. Wilkinson. The grammar of graphics. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2005.

[113] Ian H Witten, Zane Bray, Malika Mahoui, and William J Teahan. Using language models for generic entity extraction. In *Proceedings of the ICML Workshop on Text Mining*, 1999.

[114] Chang Xu, Dacheng Tao, and Chao Xu. A survey on multi-view learning. *CoRR*, abs/1304.5634, 2013.

[115] Wei Xu, Raphael Hoffmann, Le Zhao, and Ralph Grishman. Filling knowledge base gaps for distant supervision of relation extraction. In *Association for Computational Linguistics (ACL)*, 2013.

[116] Qinpei Zhao, Ville Hautamaki, and Pasi Fränti. Knee point detection in bic for detecting the number of clusters. In *Advanced Concepts for Intelligent Vision Systems*, pages 664–673. Springer, 2008.

[117] Jingbo Zhu, Huizhen Wang, Tianshun Yao, and Benjamin K Tsou. Active learning with sampling by uncertainty and density for word sense disambiguation and text classification. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 1137–1144, 2008.