Automatic Spreadsheet Data Extraction

Shirley Zhe Chen
University of Michigan
Ann Arbor, MI 48109-2121
chenzhe@umich.edu

Michael J. Cafarella
University of Michigan
Ann Arbor, MI 48109-2121
michjc@umich.edu

ABSTRACT

Spreadsheets contain a huge amount of useful data, but do not observe the relational data model, and thus cannot exploit relational integration tools. Existing systems for extracting relational data from spreadsheets are too labor-intensive to support ad-hoc integration tasks, in which the correct extraction target is only learned during the course of user interaction.

This paper introduces SENBAZURU, a system that automatically extracts relational data from spreadsheets, thereby enabling relational spreadsheet integration. When compared to standard techniques for spreadsheet data extraction on a set of 100 Web spreadsheets, SENBAZURU reduces the amount of human labor by 72% to 92%. In addition to the SENBAZURU design, we present the results of a general survey of more than 400,000 spreadsheets we downloaded from the Web, giving a novel view of how users organize their data in spreadsheets.

1. INTRODUCTION

Spreadsheets have become a critical data management tool. They allow non-experts to perform tasks we traditionally associate with relational systems: selection, projection, sorting, and simple extraction, transformation, and loading jobs, *etc.* Spreadsheets are important, especially for sophisticated users. They are commonly used for managing clinical research data [14]; researchers, scientists, and policymakers often employ spreadsheet files as a crucial part of their workflow. In short, spreadsheets make up an important dataset, but live outside the mainstream data management practices.

Spreadsheets often contain data that are roughly relational, but the schema is often designed for human consumption and entirely implicit. As a result, spreadsheets cannot benefit from society's huge investment in data management tools that work on relational databases. In particular, spreadsheets lack data integration operations. For example, it is easy to imagine an analyst who wants to combine a spreadsheet about company sales with a government-produced spreadsheet about economic performance [19] to predict future sales. But in practice, the analyst would likely have to write custom code to integrate the two spreadsheets.

Extracting relational data from spreadsheets would enable traditional data integration methods to unlock the latent value in spreadsheet data. Recent studies [1, 2, 6, 10] attempted to transform spreadsheet data into the relational model, making further integration among spreadsheets possible. Some extraction systems require explicit sheet-specific user-provided rules [2, 10], which might yield good results for a single spreadsheet. But they are infeasible for our setting: the corpus is large and users are not aware of the target spreadsheets to be processed ahead of time. It is impractical to manually transform all of them to relations. Abraham and Erwig[1], and Cunha *et al.* [6] automatically infer some spreadsheet structure, but they cannot process *hierarchical* spreadsheets. This type of spreadsheets is commonplace and extracting metadata from them presents the central technical challenge of this paper. We will illustrate these challenges using an example of a web spreadsheet downloaded from the U.S. Census Bureau.[1]

The spreadsheet in Figure 1 shows the smoking rate among different U.S. demographic groups. 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 *attribute 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 *attributes*. Often a spreadsheet is a mix of attributes, values, and other elements such as titles and footnotes. These elements are not easily distinguished from each other. Second, the spreadsheet does not explicitly indicate which *attributes* 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 attribute Male. In summary, Figure 1 shows a clean, high-quality spreadsheet, but extracting relational data from it requires us to: (1) detect *attributes* and *values*, (2) identify the hierarchical structure of left and top attributes, and (3) generate a *relational tuple* for each value in the spreadsheet.

Thus in this paper, we present SENBAZURU, the first automatic domain-independent spreadsheet extractor that can handle hierarchical spreadsheets. SENBAZURU automatically discovers the implicit metadata structure in spreadsheets

[1] http://www.census.gov/

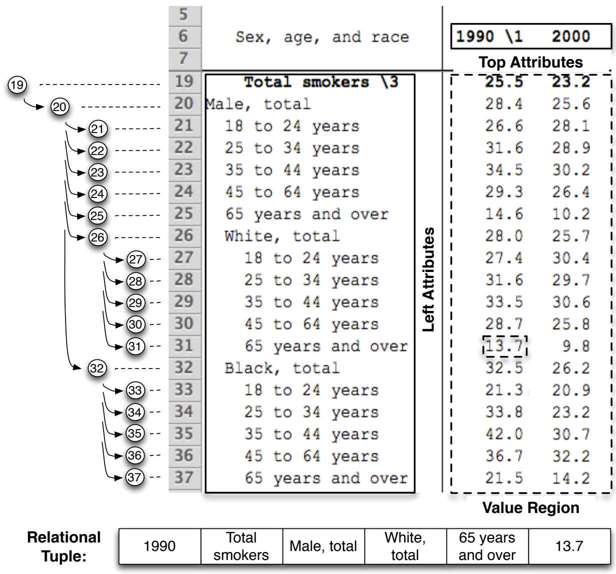

Figure 1: A portion of one spreadsheet from the U.S. Census Bureau, describing U.S. smoking rates among different gender, age, and race groupings from 1965-2007.

and emits relational tuples. SENBAZURU can be built into a compelling spreadsheet integration application, which can be used to combine data from multiple spreadsheets without writing explicit integration or extraction rules.

Contributions – The paper is organized as follows:

- We describe a general survey of spreadsheet data practices based on 410,554 spreadsheets we downloaded from a general Web crawl (Section 2).
- We present SENBAZURU, a domain-independent extractor that obtains relational tuples from raw spreadsheets without any human intervention (Section 3).
- We evaluate SENBAZURU's accuracy on a random sample of 100 Web hierarchical spreadsheets. We find that our methods can accurately obtain relational tuples from a spreadsheet; when compared to a standard technique, SENBAZURU reduces the amount of work a human being must perform between 72% and 92% on average (Section 4).

Finally, we conclude with a discussion of related and future work in Sections 5 and 6.

2. THE WEB SPREADSHEET CORPUS

In this paper, we focus on a three-part spreadsheet structure that we call a *data frame*. As shown in Figure 1, this structure consists of a block of numeric values (indicated by the dashed rectangle), accompanied by *attributes* on the *top* and to the *left* (indicated by the solid rectangles). The *top* and *left* attributes are also called *metadata*. For each value in the *value region*, there is usually (but not necessarily) at least one annotating string in the *top* and *left* regions. For example, in Figure 1, the value 13.7 is annotated by 65 years and over in the *left* region, and 1990 in the *top* region.

From our Web crawl, we obtained 410,554 Microsoft Excel files from 51,252 distinct internet domains. We call this collection the WEB dataset. We located the spreadsheets by looking for Excel-style file endings among roughly ten billion URLs in the ClueWeb09 Web crawl [4]. In this section, we aim to answer several critical questions about the WEB data in order to better design our extraction system:

1. **Source** – Where are those Web spreadsheets from?
2. **Structure** – How many of the Web spreadsheets consist of *data frame* structures?
3. **Hierarchical structure** – How many of the Web spreadsheets are hierarchical like the example shown in Figure 1? Are those hierarchical spreadsheets spread uniformly across the Web?

Source – The Web spreadsheets cover a huge range of topics and show wide variance in cleanliness and quality. Figure 2 shows a "tag cloud" for the top-100 keywords collected from the spreadsheet URLs in the WEB corpus. Clearly, many spreadsheets are related to statistical data, with a heavy emphasis on government, finance, and transportation. We are also interested in the distribution of the spreadsheets from different internet domains. Figure 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 US, Japanese, UK, or Canadian governments. Figure 4 shows the distribution of spreadsheets among hosting domains. We rank the domains according to the size of their hosting spreadsheets in a 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.

Structure – 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 attributes. The other 49.5% non-*data frame* spreadsheets belong to the following categories: 22.0% Relation spreadsheets can be converted to the relational model almost trivially (we can simply translate each spreadsheet column into a relational table column, and each spreadsheet row into a relational tuple); 10.5% Form spreadsheets are not for data storage, and are designed to be filled by a human; 3.5% Diagram spreadsheets are for visualization purposes, and they are often data-intensive without any schema information; and 3% List spreadsheets 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.

Hierarchical Structure – As just mentioned, 32.5% of the 200 sample Web spreadsheets have hierarchical *top* or *left* attributes 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 shows the fraction of spreadsheets with *data frames* or hierarchical attributes 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% with hierarchical *top*

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

Figure 2: A tag cloud of terms found in URLs of our WEB spreadsheet corpus.

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: 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.

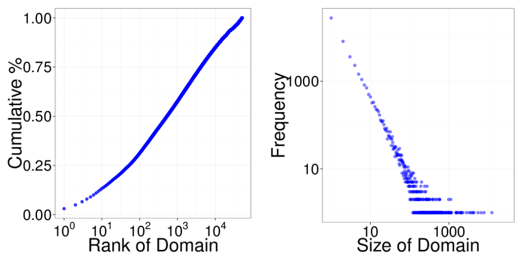

Figure 4: The distribution of Web spreadsheets. The left plot depicts the cumulative ratio of the ranked domains in the corpus. The right plot shows the number of domains given the size of their hosting spreadsheets.

attributes and 6% with hierarchical *left* attributes. 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. SYSTEM PIPELINE

The goal of SENBAZURU is to create a relational model of the data embedded in *data frame* spreadsheets. The extraction pipeline, as shown in Figure 5, consists of three components: 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*.

3.1 Frame Finder

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 *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 $l_{ij} \in \mathrm{L} = \{$top, left, value, other$\}$, where top represents *top attributes*, left represents *left attributes*, 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. In fact we found less than 2% of the 900 spreadsheets in Figure 3 violate the assumption. This assumption allows us to treat *data frame*-finding as a problem of row labeling. Therefore we start with the task, **row labeler**, 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* attributes only; data represents a row that contains *left* attributes or *values*; and footnote is information that annotates the main contents. As in Figure 1, rows 5-7 are labeled header, and rows 19-37 are labeled data. A formal definition is as follows:

DEFINITION *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 $l_i \in L = \{$title, header, data, footnote$\}$.

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*.

Our **row labeler** assigns semantic labels to each non-empty row based on linear-chain conditional random fields (CRFs) [12]. Pinto *et al.* [15] used linear-chain CRFs to obtain labels for textual tables in government statistical reports. Our approach is similar and uses the same training and inference procedure. However, we process spreadsheets with a richer set of features, which fall into two main categories. ***Layout features*** test visual characteristics of a row; *e.g.*, whether a row has a merged cell, whether it contains a bold font cell, and so on. ***Textual features*** test the contents of the row; *e.g.*, whether the row contains the word "table", whether it has indented cells, whether it has a high percentage of numeric cells, and various others.

3.2 Hierarchy Extractor

The **hierarchy extractor** recovers the *attribute 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 attributes in the *top* and *left* regions. For example, in Figure 1, row 31 is annotated by attributes at rows 26, 20, and 19. An example of a *top* hierarchy can be found in Figure 6, where the *attribute* Air-

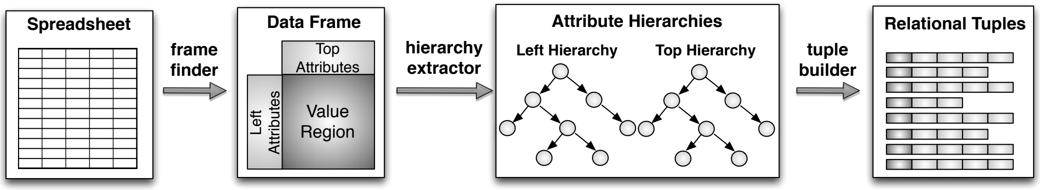

Figure 5: The sequence of steps when SENBAZURU processes a single spreadsheet. The **frame finder** locates the *data frame* within the spreadsheet grid, and the **hierarchy extractor** extracts the *attribute hierarchy* structure within the *left* and *top* regions of the *data frame*. Finally, **tuple builder** recovers the relational tuples.

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 6: An example of hierarchical top attributes.

plane pilots annotates the *attribute* Airline transport. Now we formally describe the problem of recovering the *attribute hierarchy* for a single region as follows:

DEFINITION *3.* (Hierarchy Extractor) Let $A = \{a_1, ..., a_N, root\}$ represents the set of cells in an attribute region. Given $a_i, a_j \in A$, we say (a_i, a_j) is a *ParentChild* pair if a_i is a parent attribute of a_j in the annotation hierarchy. The **hierarchy extractor** identifies all the *ParentChild* pairs in the attribute region.

For example, consider the hierarchy on the left of Figure 1 where each node represents an attribute at the corresponding row. We can see *(20, 26)* is a *ParentChild* pair while *(20, 31)* and *(24, 25)* are not. Therefore, the goal of the **hierarchy extractor** is to find all such *ParentChild* pairs in an attribute region, thus constructing a hierarchical tree.

It may seem that a simple heuristic can recover the *annotation hierarchies*, but in practice a correct heuristic is extremely hard to obtain and to generalize to a large number of spreadsheets. As in Figure 1, a simple heuristic, such as the indentations of *left* attributes, may identify some *ParentChild* pairs but fails to recognize *(19, 20)* as a *ParentChild* pair. Therefore, to obtain a correct hierarchy, we need to use a probabilistic method to exploit a variety of signals.

Classification – Our first proposed technique is a *classification* method, a scalable approach that can incorporate a variety of signals in a spreadsheet to extract *attribute hierarchies*. We enumerate all the possible attribute pairs in an attribute region as the *ParentChild* candidates and train a classifier to label each of them true or false. This classification process emits a set of *ParentChild* pairs to construct the attribute hierarchy. If the classification is entirely correct, the produced *ParentChild* pairs will construct a tree. But any error in the classification might yield an inaccurate result. The classifier utilizes two types of features: *unary features*, such as the textual and layout features described in Section 3.1; and *binary features*, such as whether there are blank cells between the parent and child attribute, and whether the two attributes are adjacent.

Enforced-tree classification – Our second proposed technique is *enforced-tree classification*, which embeds simple heuristics into the classification results, to ensure the

Algorithm 1 Enforced-tree Classification

```
function EXTRACTHIERARCHY(attributeRegion)
    parentChildPairs = []
    for a1 in attributeRegion do
        maxProb, maxParent = 0, root
        for a2 in attributeRegion do
            cProb = ClassifyParentChild(a2, a1)
            if cProb > maxProb then
                maxProb, maxParent = cProb, a2
            end if
        end for
        if maxProb > θ then
            parentChildPairs.append((maxParent, a1))
        end if
    end for
    attributeHierarchy = BreakCycles(parentChildPairs)
    return attributeHierarchy
end function
```

produced pairs construct a strict hierarchical tree. Of course, in a tree structure each node has only one parent (except the root). Thus for each attribute, we select the one with the maximal probability as its parent attribute. We obtain the probability associated with each *ParentChild* pair during the classification. This *one-parent* constraint does not guarantee that the output will be a tree: cycles may still exist in the results. We then iteratively break cycles by deleting the pairs with the minimal probability until there are no more cycles in the output. We assume one's parent attribute is Root by default. Therefore, the two steps, the *one-parent* constraint and the *breaking-cycles*, enforce the classification results to generate a strict tree. The classifier uses the same set of features as the **classification** method, and the details of the algorithm are shown in Algorithm 1.

3.3 Tuple Builder

The **tuple builder** generates a tuple for each value in the *value region*. For example, the bottom of Figure 1 shows the recovered tuple for the highlighted value 13.7. The **tuple builder** is algorithmically straightforward. It processes the extracted *attribute hierarchies* and the *value region* to generate a series of relational tuples. Described in Algorithm 2, the **tuple builder** relies entirely on the **frame finder** and **hierarchy extractor** for correctness.

4. EXPERIMENTS

We can now quantify the performance of the system by 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.

In the following experiments, we use 100 random hier-

Algorithm 2 Tuple Builder

```
function BuildTuples(leftHier, topHier, valueRegion)
    tuples = []
    for v in valueRegion do
        t = [v]
        t.append(leftHier.getAnnotatingAttributes(v))
        t.append(topHier.getAnnotatingAttributes(v))
        tuples.append(t)
    end for
    return tuples
end function
```

archical *data frame* spreadsheets (*data frame* spreadsheets with hierarchical *top* attributes or hierarchical *left* attributes). We obtained this dataset by randomly sampling spreadsheets from WEB and only keeping those with a hierarchical *data frame*. For *top*, the average hierarchy depth of the dataset is 2.14, with a maximum depth of 5; for *left*, the average hierarchy depth is 2.61, with a maximum depth of 9. The training and testing procedures for both the **row labeler** and the **hierarchy extractor** are as follows: we randomly split the dataset into equal-sized training and testing sets, repeating this process 10 times. Then we report the average `Precision`, `Recall`, and `F1` measure for each class. We also present the `mean` and standard deviation (`std`) for *errors per sheet*, which is defined as follows:

Definition *4.* (Errors per sheet) A classification task produces two types of errors, false positive (fp) and false negative (fn), on N spreadsheets. We define *errors per sheet*:

$$Errors_{sheet} = (fp + fn)/N$$

For the experiment setup, we used several open-source packages: for **frame finder**, our CRFs were implemented on CRF++ [5]; for **hierarchy extractor**, we used the SVM library from the LIBSVM package [3], and the Weka package [8] for the logistic regression and naive Bayes method.

4.1 Frame Finder

We now evaluate the performance of the **frame finder** described in Section 3.1 by evaluating the **row labeler**. Our experiment spreadsheets contain 27,531 non-empty rows that are correctly assigned with semantic labels by a human expert. In [15], CRFs were used to label the lines of tables in plain-text government statistical reports using *textual* features. Our **row labeler** also uses CRFs but incorporates richer features: both *textual* and *layout* features. The *layout* features, such as bold font and alignment, are hard to obtain from plain text but accessible in spreadsheets through the Python xlrd library. Therefore, we compare two CRFs with different sets of features: Base-CRF for *textual* features, and Full-CRF for *textual + layout* features.

As shown in Table 1, Full-CRF performs significantly better than Base-CRF on all the metrics, including precision, recall, and errors per sheet. According to precision and recall, both methods do a decent job of predicting all the labels, but they show big difference in the number of errors, especially for the label data. For data, the two methods have very close precision and recall, but Base-CRF produces many more errors than Full-CRF. This occurs because of the huge number of data rows in the dataset, and a small difference in the F1 measure will make a big difference to the absolute number of errors. Overall, Table 1 shows that the Full-CRF method is superior to the baseline Base-CRF method and can work effectively as a part of the system: Full-CRF predicts each

		Performance			**Errors**	
		Precision	Recall	F1	mean	std
title	Base-CRF	0.561	0.605	0.582	3.534	4.532
	Full-CRF	0.818	0.734	0.774	0.872	0.150
header	Base-CRF	0.624	0.606	0.615	2.348	0.621
	Full-CRF	0.812	0.740	0.774	1.316	0.343
data	Base-CRF	0.995	0.970	0.982	6.526	5.239
	Full-CRF	0.995	0.993	0.994	1.528	0.330
footnote	Base-CRF	0.550	0.786	0.647	4.208	3.414
	Full-CRF	0.843	0.826	0.834	1.208	0.223

Table 1: Performance of the **row labeler**.

		Performance			**Errors**	
		Precision	Recall	F1	mean	std
top	SVM	0.921	0.918	0.919	1.834	0.398
	EN-SVM	0.923	0.918	0.920	1.829	0.395
left	SVM	0.852	0.700	0.769	19.554	5.107
	EN-SVM	0.850	0.776	0.811	16.154	4.332

Table 2: Performance of the **hierarchy extractor**'s classification of *ParentChild* pairs for both *left* and *top*.

category fairly accurately, and the number of errors produced by Full-CRF is tolerable for all the labels, with about one error per sheet.

4.2 Hierarchy Extractor

We evaluate the performance of the *hierarchy extractor* discussed in Section 3.2 by measuring its accuracy in retrieving correct *ParentChild* pairs from a spreadsheet. The hierarchical metadata in spreadsheets is unique and we are not aware of any previous method to automatically extract such hierarchical metadata. Hung *et al.* [10] propose the most recent method for transforming spreadsheet data into a structural format. While Hung *et al.*'s method does not specifically address hierarchical metadata extraction, their method can be used to create a rule-based program to extract the hierarchical metadata. However, this program, called Hung, requires a user to manually enumerate all the *ParentChild* pairs, which is exactly what Senbazuru infers automatically. Therefore, we first evaluate the performance of our two approaches: SVM for *classification* and EN-SVM for *enforced-tree classification*. We then compare our two methods with the state-of-the-art method, Hung, on the metric *user repair #*.

Definition *5.* (User Repair #) We assume that a user reviews every *ParentChild* pair candidate with an assignment, true or false, in an attribute region. **User repair #** is the number of assignments the user must modify in order to obtain the correct hierarchy.

For SVM and EN-SVM, *user repair #* equals *errors per sheet* in a given attribute region. For Hung, and *user repair #* is the total number of true *ParentChild* pairs in an attribute region (we assume that all the *ParentChild* pair candidates are labeled false by default).

Table 2 shows the performance of our two methods. We also tried logistic regression and naive Bayes for classification. Overall, SVM performs the best. As seen in the Table 2, EN-SVM performs the best, especially on *left*. Note that for *left*, EN-SVM has a higher recall than SVM, but a slightly lower precision. The reason is that given an attribute, all the *ParentChild* pair candidates containing its parent attribute may be labeled false by the classifier, and then the attribute will not have any parent attribute. However, EN-SVM is able to recover its parent attribute by selecting the most probable *ParentChild* pair from the false

		Repairs
top	Hung[10]	22.469
	SVM	1.834
	EN-SVM	1.829
left	Hung[10]	58.598
	SVM	19.554
	EN-SVM	16.154

Table 3: User repair # for the **hierarchy extractor**.

group. Table 3 presents the *user repair #* for the three methods, and it shows that both SVM and EN-SVM require a much smaller number of user repairs than Hung. Therefore, we conclude that our method EN-SVM is superior to the state-of-the-art Hung: EN-SVM predicts the *ParentChild* pairs fairly accurately, and it beats Hung on *user repair #*.

One limitation of SENBAZURU lies in the fact that the absolute number of required repairs on *left* is not trivial. According to Table 3, the number of repairs on *top* is almost negligible, but not on *left*. We will try to reduce the user burden even further in future work.

5. RELATED WORK

Recently there has been extensive research into combining database-like operators with a spreadsheet-style interface [13, 20, 21, 22]. The *QueryByExcel* project [21, 22] uses a spreadsheet as a front end of the relational database. It translates Excel formulas using an extension of SQL relational operations and performs on RDBMS tables. Liu *et al.* [13] implemented an extended set of database functionalities operating on spreadsheets, and the operations are executed by a classic database engine in the background. Tyszkiewicz [20] also attempted to combine SQL with spreadsheets, but implemented the functionality inside spreadsheet software instead of using an additional database engine. Unlike our work, these papers exploit the spreadsheet interface rather than extract spreadsheet-embedded data.

The goals of some spreadsheet-specific systems [2, 10] are similar to ours, but require explicit user-provided rules. Two exceptions are the work of Abraham and Erwig [1] and Cunha *et al.* [6]. The former attempts to infer spreadsheet metadata, but they do not address the hierarchical structures in spreadsheets. The latter tries to convert spreadsheets to relational databases, but their primary technical focus is to address the lack of data normalization in spreadsheets that use very conventional layouts. The approach also does not address hierarchical spreadsheets.

There has also been spreadsheet-related work in the visualization community [7, 11, 16, 17, 18]. Focus [18] is an interactive tool that builds flexible table overviews, using incremental database queries to restrict the table to a relevant subset of the data. Its "fisheye" visualization approach is based on the TableLens system [17]. Raman and Hellerstein [16] provide users with an interactive spreadsheet-like interface to perform a set of predefined transformation operators. Vizql [9] unifies languages including SQL and MDX. More recently, Guo *et al.* [7] developed Wrangler, a data transformation tool that interactively helps users build small transformation sequences, by repeatedly producing a ranked list of operation recommendations.

6. FUTURE WORK AND CONCLUSIONS

We have described a domain-independent system, SENBAZURU, for converting spreadsheet data into relational tuples. Our system consists of three components that detect the structure of a spreadsheet, extract hierarchical metadata, and generate relational tuples. Our experiments show that our proposed methods are significantly superior to the state-of-the-art approaches, and can work effectively as a part of the whole framework. As a result, SENBAZURU can help bring relational-style data management techniques to the mass of data currently locked in spreadsheets.

One area of future work lies in incorporating manual user repairs to improve the performance. How to effectively utilize the repairs should be an interesting problem to investigate in the future. Moreover, performing data integration on the relational tuples with heterogeneous relational sources is our ultimate goal. Our initial experience with integrating extracted relational data from spreadsheet corpora has shown that finding a good joining dataset itself is already a difficult task.

7. REFERENCES

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

[2] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi. A type system for statically detecting spreadsheet errors. In *ASE*, pages 174–183, 2003.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[4] 2009. ClueWeb09, http://lemurproject.org/clueweb09.php/.

[5] 2009. http://crfpp.sourceforge.net.

[6] J. Cunha, J. Saraiva, and J. Visser. From spreadsheets to relational databases and back. In *PEPM*, pages 179–188, 2009.

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

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11.

[9] P. Hanrahan. Vizql: a language for query, analysis, and visualization. In *SIGMOD*, page 721, 2006.

[10] V. Hung, B. Benatallah, and R. Saint-Paul. Spreadsheet-based complex data transformation. In *CIKM*, pages 1749–1754, 2011.

[11] T. Igarashi, J. D. Mackinlay, B.-W. Chang, and P. Zellweger. Fluid visualization for spreadsheet structures. In *VL*, pages 118–125, 1998.

[12] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[13] B. Liu and H. Jagadish. A spreadsheet algebra for a direct data manipulation query interface. In *ICDE*, pages 417–428, 2009.

[14] M. Nahm and J. Zhang. Operationalization of the ufurt methodology for usability analysis in the clinical research data management domain. *Journal of Biomedical Informatics*, 42(2):327–333, 2009.

[15] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *SIGIR*, 2003.

[16] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, 2001.

[17] R. Rao and S. K. Card. The table lens: merging graphical and symbolic representations in an interactive focus + context visualization for tabular information. In *CHI*, pages 318–322, 1994.

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

[19] 2010. Statistical Abstract of the United States, http://www.census.gov/compendia/statab/2010/.

[20] J. Tyszkiewicz. Spreadsheet as a relational database engine. In *SIGMOD*, pages 195–206, 2010.

[21] A. Witkowski, S. Bellamkonda, T. Bozkaya, G. Dorman, N. Folkert, A. Gupta, L. Sheng, and S. Subramanian. Spreadsheets in rdbms for olap. In *SIGMOD Conference*, pages 52–63, 2003.

[22] A. Witkowski, S. Bellamkonda, T. Bozkaya, A. Naimat, L. Sheng, S. Subramanian, and A. Waingold. Query by excel. In *VLDB*, pages 1204–1215, 2005.