Spreadsheet Property Detection With Rule-assisted Active Learning

Zhe Chen[1], Xin Rong[1], Sasha Dadiomov[2], Richard Wesley[2],
Gang Xiao[2], Daniel Cory[2], Michael Cafarella[1], Jock Mackinlay[2]

[1]University of Michigan, [2]Tableau Software
{chenzhe,ronxin,michjc}@umich.edu, {sdadiomov,hawkfish,gxiao,dcory,jmackinlay}@tableau.com

ABSTRACT

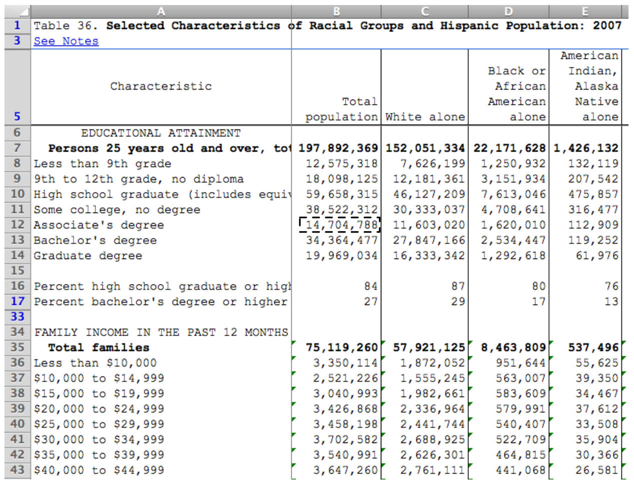
Spreadsheets are a critical and widely-used data management tool. Converting spreadsheet data into relational tables would bring benefits to a number of fields, including public policy, public health, and economics. Research to date has focused on designing domain-specific languages to describe transformation processes or automatically converting a specific type of spreadsheets. To handle a larger variety of spreadsheets, we have to identify various *spreadsheet properties*, which correspond to a series of transformation programs that contribute towards a general framework that converts spreadsheets to relational tables.

In this paper, we focus on the problem of spreadsheet property detection, identifying when a corresponding transformation program should be applied. We propose a hybrid approach of building a variety of spreadsheet property detectors to reduce the amount of required human labeling effort. Our approach integrates an active learning framework with crude, easy-to-write, user-provided rules. Our experiments show that when compared to a standard active learning approach, we reduced the training data needed to reach the performance plateau by 34–44% when a human provides relatively high-quality rules, and by a comparable amount with low-quality rules. A study on a large-scale web-crawled spreadsheet dataset demonstrates that it is crucial to detect a variety of spreadsheet properties in order to transform a large portion of the spreadsheets into a relational form.

1. INTRODUCTION

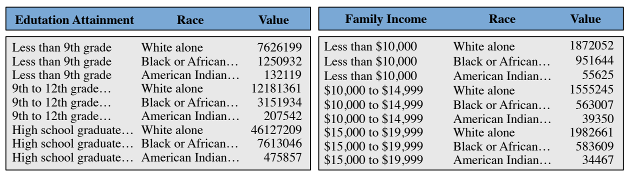
Spreadsheets are widely used for data management and sharing. It is estimated that Microsoft Excel has more than 400 million users, and 50–80% of businesses use spreadsheets.[1] Meanwhile, a large number of spreadsheets are available on the web. For example, the United States Census Bureau publishes thousands of spreadsheets about economics, transportation, public health, and other important social topics every year.

Many spreadsheet files are designed to be interpreted by human, and often cannot be easily consumed by other software applications for complex data analysis and visualization (e.g., R, Tableau). For example, Figure 1 shows a part of a spreadsheet downloaded from the Census Bureau. This spreadsheet is almost *impossible* to be consumed by downstream data analysis programs, if we fail to identify the structural features, such as title (rows 1–3), header (row 5), sub-header (rows 6, 34), and aggregation rows (rows 7, 35).



	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: 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 2: The ideal relational tables for the spreadsheet example shown in Figure 1.

To make it more machine readable, the same spreadsheet can be converted to *relational tables*, as shown in Figure 2. An essential requirement for such relational tables is that each column should be homogeneous, or, belong to the same semantic class.

Automating the conversion of a spreadsheet into a relational table apparently has great appeal for a number of communities. One way of achieving this is by designing a domain-specific language (DSL) to describe the rules for spreadsheet-to-relational-table transformation and implementing a program to support the DSL [2, 11, 13, 14]. However, this approach requires a significant amount of human effort for composing the rules for each spreadsheet variant. Another approach is to make assumptions on the structural features of spreadsheets (e.g., assuming a spreadsheet *only* has headers and sub-headers), and use heuristics or data-driven models to transform certain types of spreadsheets into a relational format [1, 5, 7, 6, 8]. While this approach requires less human effort, the range of the spread-

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

sheets it supports is restricted by its assumptions on the spreadsheet structure.

In this paper, we envision a framework for transforming *any* kind of spreadsheets into relational tables. The center idea of building the framework is to identify and transform *spreadsheet properties*, i.e., the special structural features that distinguish a spreadsheet table from a relational table. Given a spreadsheet table, the pipeline consists of two stages: identifying the existence of *spreadsheet properties*; and applying transformation for each identified property.

Take the table in Figure 1 as an example, the identifiable *properties* and the corresponding *transformations* include:

- *aggregation rows*—Data values in rows 16–17 are aggregated values defined on rows 7–14. Transformation: *remove* the aggregation rows.
- *aggregation columns*—Data values in column B are aggregated values defined on column C–E. Transformation: *remove* the aggregation column.
- *crosstab*—The headers of columns C–E (*i.e.*, "White alone", "Black or ...", etc.) form a horizontal dimension "Race." Transformation: *pivot* this horizontal dimension into a new column "Race."
- *split tables*—Rows 6–17 are about "Education Attainment" and rows 34–43 are about "Family Income." Transformation: *split* as two tables.

If one can identify all the properties above and correctly apply the corresponding transformations, then she can successfully transform the spreadsheet in Figure 1 into relational tables as shown in Figure 2. We argue that accurately detecting the existence of *spreadsheet properties* is essential to such a transformation process. This is because while some transformations are straightforward (e.g., removing aggregation rows or columns), many operations are non-trivial and can be computationally expensive. As suggested by [6], transforming spreadsheet tables with *hierarchical* structure may take $O(N^2)$ time, where N is the number of rows. Thus, spreadsheet property detection can greatly improve the computational efficiency of the overall pipeline by avoiding expensive and unnecessary transformations. In addition, training a transformation model for a given property requires extensive human labeled data. If a technique exists to accurately identify the set of spreadsheets that possess a given property, then it will be much easier to construct a human labeled dataset to train a transformation model for that property. Therefore, in this paper, instead of discussing an end-to-end pipeline converting spreadsheet tables into relational tables, we focus exclusively on the problem of detecting *spreadsheet properties*.

Spreadsheet property detection is a challenging task by itself, for two reasons. First, labeling instances to train property detectors is expensive. For example, to determine whether a spreadsheet contains the property *aggregation rows*, a human labeler may have to review all the header or data cells for potential keywords (e.g., "total", "sum", "average"), as well as checking whether the cells contain calculated values based on a formula. Second, there are a variety of customized spreadsheet datasets, and one might look very different from another. To build high-quality property detectors requires a sufficient number of labeled instances that also cover a large variety of spreadsheet types.

To this end, we propose a novel rule-assisted active learning framework to construct high-quality spreadsheet property detectors, and its goal is to save human labeling effort as much as possible. Our key insight is that a human labeler can not only provide labels to individual training instances, but also write crude heuristic rules based on their intuitions on how a property *might* be detected. An example rule can be, "if a spreadsheet contains a row with formulas, then it has the property *aggregation rows*." Such rules are, obviously, not always reliable. But we design a hybrid framework that integrates such crude user-provided rules and user-provided labels based on their agreement so as to improve the system's tolerance on *low-quality* rules. In addition, we adopt an active learning strategy to iteratively ask human to label the most ambiguous training instances. The hybrid approach can generate additional high-quality labeled data, especially in the initial stage of training, in order to bootstrap the learning process.

Our approach was evaluated on a sample of web spreadsheet dataset of 400 tables labeled with properties. The result indicated that we could reduce the amount of labeled data needed to reach the performance plateau by 34–44% when a human provides high-quality rules, and comparable performance with low-quality rules. We also applied the trained property detectors to a much larger-scale dataset of 1.1 million spreadsheets, and provided insights on how the distribution of identified spreadsheet properties impact the downstream transformations into relational tables.

Our contributions are as follows:

- The concept of *spreadsheet properties*, and its relationship to the transformation from spreadsheets to relational tables.
- A novel, hybrid, rule-assisted active learning framework for spreadsheet property detection. This integrates an active learning framework with crude user-provided rules to save human labeling effort. By using a bagging-like technique, it can tolerate lower-quality user-provided rules (Sections 3 and 4).
- A comprehensive evaluation that demonstrates our hybrid framework outperforms active learning baselines by significantly reducing the training data needed to reach the performance plateau (Section 5).
- Findings on a large-scale web spreadsheet corpus that can serve as a guideline for designing an end-to-end pipeline that transforms spreadsheets to relational tables (Section 6).

2. RELATED WORK

Spreadsheet Management – Existing approaches for transforming spreadsheets into relational tables fall into two categories. First, *rule-based* approaches [2, 11, 13, 14] require users to learn a domain-specific language to describe the transformation process. These approaches are flexible but composing the rules is often difficult and time-consuming. Different from the above approaches, our trained property detector can automatically suggest transformation programs.

Second, *automated* approaches are the most similar to ours. Abraham and Erwig [1] attempt to recover spreadsheet tuples, and Cunha *et al.* [8] primarily focus on the problem of data normalization. Chen and Cafarella [5, 6, 7] focus on extracting hierarchical structure in spreadsheets by incorporating users' feedback. While the existing work mainly focuses on transforming a specific type of spreadsheets, we attempt to build a framework that can handle a much larger variety of spreadsheets.

Active Learning – There are two common active learning strategies [19]. First, the *uncertainty sampling* strategy chooses to label instances that are closest to the decision boundary, and it refines the decision boundaries by heavily exploiting the current knowledge space. The uncertainty sampling approach in [18] selects the instance with the predicted probability closest to 0.5. Second, the *query by committee* (QBC) strategy takes into account the disagreement of multiple "committee" classifiers to select query instances [20]. This is more complicated than uncertainty sampling as it requires careful designs of committee members (i.e., a set of classification models) and a metric to measure disagreement among committee members. While our hybrid iterative framework is based on the basic uncertainty sampling strategy, our learning framework is distinct in that it incorporates the crude user-provided rules to further reduce the amount of required human effort.

Alternative strategies exist for utilizing human resources for model development. Attenberg and Provost [3] use a "guided learning" approach to search explicitly for training examples for each class. Druck *et al.* [10] propose an active learning approach in which the machine solicits labels on features rather than instances. Xiaoxuan *et al.* [21] considers online learning with imbalanced streaming data under a query budget, and the approach utilizes the end-user effort to enable customization and personalization. Similar to these approaches, we ask the user to do more than labeling training instances (in our case, providing crude rules for property detection). But different from their situation, we focus on addressing the the scenario where the user provides low-quality rules by using a bagging-like technique.

We notice that active learning strategies often suffer from the "cold-start" problem [23]: in the beginning stage, the classifier lacks training data to approach the ideal decision boundary and suggest effective instances to label. Zhu *et al.* [23] address this problem by finding clusters of distinct content among the unlabeled instances. Donmez *et al.* [9] 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 to provide heuristic rules. Such rules are used to generate additional labels to warm up the classifiers quickly.

3. PROPERTY DETECTION FRAMEWORK

We define a *spreadsheet table* as the portion of a spreadsheet that can be converted into one or more relational tables. A spreadsheet table consists of two regions: a header region and a data region. For example, in Figure 1, the header region is row 5, and the data region spans across rows 6–43. One can employ a linear-chain conditional random field (CRF) to automatically identify the header and data regions [5].

Given a spreadsheet table, the property detection task is to build a binary classifier for a spreadsheet property. We formally define the task below.

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 spreadsheet table has the property q. Given a spreadsheet 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 $\mathbf{q}$.

3.1 The Iterative Learning Framework

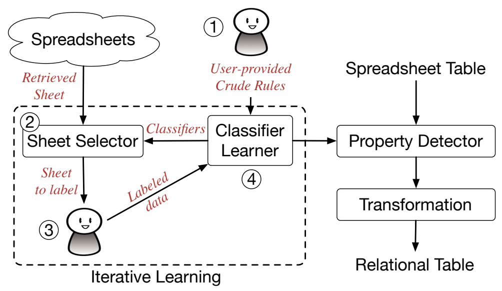


Figure 3: The hybrid iterative learning framework for spreadsheet property detection.

Figure 3 shows our proposed hybrid iterative learning framework for spreadsheet property detection. In the initial stage, a human labeler provides crude heuristic rules (see Section 3.3 for a detailed discussion). During the interactive learning stage, the sheet selector selects a spreadsheet from the dataset, and presents it to the human labeler. The labeler is responsible for labeling the spreadsheet with all the spreadsheet properties it contains. The classifier learner then accumulates all human labeled spreadsheets together with automatically generated labels using the user-provided rules, to train a classifier for each spreadsheet property. The human labeler 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, which can then be used in an end-to-end pipeline that transforms spreadsheet tables into relational ones.

Note that in the cases of imbalanced training data, we duplicate the instances of the minority class until its size is comparable to the size of the majority class [12]. In the rest of this section, we describe the human labeling process and techniques to save human effort.

3.2 Human Labeling Process

3.2.1 Construct Property Detectors

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

1. Features $f(x)$: We generate features $f(x)$ for each spreadsheet 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 spreadsheet table's data region contains the keyword "total", it is very likely to have the property "aggregation rows". 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.

Our features are as follows:

- whether a cell in the header/data region contains one of the keywords: "total", "sum", "avg", "average", "median", "mean", "totals", "summary", "subtotal";
- the standard deviation of the lengths of the strings in the header;
- 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 the data region has different for-

matting styles, and we test each of the 8 styles.[2]

- whether the data/header region has a merged cell;
- whether there exists two cells in the header region, one has a higher column but lower row index than the other;
- whether the spreadsheet table is empty;
- whether there is no header/data region;
- the ratio of numeric cells to total cells in the spreadsheet table;
- the ratio of non-zero cells to total/numeric cells in the spreadsheet table;
- the maximum ratio of non-zero cells to numeric cells in data rows/columns;
- the ratio of numeric to all data rows/columns;
- the absolute number of numeric data rows/columns.

2. Property Set (Q): It is hard to construct a complete spreadsheet property set Q in one shot because there are always unknown properties in new data. Instead, we define a few properties that we are aware of as the set of *predefined properties.* At the same time, we allow new properties to be added during labeling.

3. Training Data $D = \{(x, \mathbf{q})\}$: given a spreadsheet table x, a human labeler has to determine the set of properties $\mathbf{q}$ contained by x. During the labeling process, the human labeler evaluates the transformation process for converting a spreadsheet table x to relational tables, and decides whether x contains the predefined spreadsheet properties or new properties.

To be specific, a human labeler first labels a spreadsheet table x using the *predefined properties.* It is straightforward to decide whether a spreadsheet x contains a well-defined property. In addition, the human labeler is also tasked with *discovering new properties* via the following procedure: after labeling x using the predefined properties, the human labeler attempts to convert x to relational tables using the transformation operations defined by $\mathbf{q}$ and determines whether the conversion is successful. If not, the human labeler has to define one or more new spreadsheet properties with corresponding transformation operations, and then add the new properties to $\mathbf{q}$.

For example, assume that we have defined two properties, "aggregation rows" and "aggregation columns." For the spreadsheet table shown in Figure 1, we recognize that it contains both properties. We then attempt to use the corresponding transformation programs to convert this spreadsheet table to relational tables. In this case we would fail, because we still need to separate rows 6–17 (about "Education Attainment") and rows 34–43 (about "Family Income") into two separate relational tables. Therefore, we define a new spreadsheet property "split table", and add it to $\mathbf{q}$. We will keep finding new properties until the spreadsheet table can be successfully transformed into relational tables.

As can be seen from the above discussion, it requires a considerable amount of human effort to construct a binary classifier for each spreadsheet property.

3.3 Reducing Human Effort

To reduce the amount of required human effort on generating *training data* $D = \{(x, \mathbf{q})\}$, we adopt the following two strategies:

[2]The 8 styles are: a cell's alignment; a cell's height; a cell's indentations; whether a cell contains colon; whether a cell is bold; whether a cell is capitalized; whether a cell is italic; whether a cell is underlined.

Uncertainty Sampling — In active learning, a typical strategy to pick instances for training a binary classifier is *uncertainty sampling*, which chooses instances closest to the decision boundary. Our sheet selector adopts this strategy. However, during the beginning phase of the training process, there lacks enough training data for the classification model to approach a reasonable decision boundary. The technique introduced below addresses this problem.

User-provided Crude Rules — Before labeling any spreadsheet, we bring in human's intuition on building property detectors by asking for crude and easy-to-write rules. For example, it might be straightforward for a user to assume, "if a spreadsheet contains a row with formulas, then it has the property *aggregation rows.*" In our framework, we ask for simple rules like this (see Table 1 for more examples) and do not need a user to spend a huge amount of effort coming up with high-quality ones.

Now that we have a set of crude rules, in the initial stage of training, we can generate a set of training instances by first applying such rules to the available data, and treating the results as labeled instances. As the training progresses, the number of human-labeled instances increases. This allows us to filter the labeled training instances by finding those with agreement from both the user-provided rules and the trained classifier at each iteration. This makes it possible for our framework to tolerate *low-quality* user-provided rules. Then we can approach the ideal decision boundary quickly to reduce the amount of required labeled data.

In the next section, we describe the training algorithms in detail.

4. ALGORITHMS

Let $\mathbf{x} = \{x\}$ be the random variables representing a set of spreadsheet tables, and θ_q 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.

4.1 Iterative Learning Algorithms

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.

As shown in Algorithm 1, the sheet selector selects a new instance from the spreadsheet table set (we describe the algorithm in Section 4.2); a human labeler 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 — As shown in Algorithm 2, given a spreadsheet property q, the user-provided rules θ_{q_init} produces a set of labels $\{l_{q\_init}\}$ on the spreadsheet table set $\{x\}$, and each label l_{q_init} represents whether the corresponding spreadsheet table x has the property q or not. However, we do not know the quality of the rule-generated labels $\{l_{q\_init}\}$.

For each property q, we collect the training data for each

Algorithm 1 Iterative learning without user-provided rules.

```
Input:  spreadsheet table set $\mathbf{x} = \{x\}$
Output:  property detectors $\{\theta_q\}$.
1: $D = []$    // Initialize training data
2: repeat
3:     Sheet selector chooses $x$ from $\{x\}$
4:     Ask human to label $x$ with properties $\mathbf{q}$
5:     $D \leftarrow D \cup (x, \mathbf{q})$    // Update training data
6:     $Q \leftarrow Q \cup \mathbf{q}$    // Update property set
7:     Train classifier $\theta_q$ on $D$ for each $q \in Q$
8: until meet stopping criteria
9: return $\{\theta_q\}$
```

Algorithm 2 Iterative learning with user-provided rules.

```
Input:  spreadsheet 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 human 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 $\{\theta_q\}$
```

learning iteration in two parts: first, we accumulate all the human-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 and denote it as a *consensus label*; otherwise, we cannot trust either label. If, however, the consensus label conflicts with human labels D, then we still believe the human labeled data. The idea of finding the consensus labels is similar to the bootstrap aggregating technique (*i.e.*, bagging) [4]: it attempts to find the label agreements of multiple classifiers. Based on the bagging-like technique, our approach is able to tolerate "low-quality" user-provided rules and provide additional high-quality labels especially in the initial stage to warm up the classifiers quickly.

Similar to Algorithm 1, the sheet selector selects a new instance; a human labeler labels the correct properties; and finally the classifier learner trains the property detectors by combining the accumulated human 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.

4.2 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 by following the configuration used in [15].

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 as used in [18].

To be concrete, the sheet selector selects the spreadsheet table x to be

$$\arg\max_x \left[\min \Big((P(l_q = 1 \mid x), P(l_q = 0 \mid x) \Big) \right] \tag{1}$$

where $P(l_q \mid x)$ represents the probability distribution of the spreadsheet 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 [16, 17]. For simplicity, 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 spreadsheet table x to be

$$\arg\max_x \frac{1}{|Q|} \sum_{q \in Q} \min \Big((P(l_q = 1 \mid x), P(l_q = 0 \mid x) \Big) \tag{2}$$

where $P(l_q \mid x)$ represents the probability distribution of the spreadsheet table x contains the property q according to the current trained classifier θ_q.

5. EXPERIMENTS

In this section, we conduct experiments to test our two goals as follows:

- **Spreadsheet Property Detection** — We investigate the algorithms to build high-quality property detectors with a small labeled dataset.
- **Large-scale Spreadsheet Study** — We survey the distribution of the five most popular spreadsheet properties in the large-scale web data, and our findings serve as guidelines for designing the spreadsheet-to-relational table transformation system.

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.

5.1 Data Sources and Experiment Setup

We rely on two spreadsheet data sources:

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.[3]

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 [5]. Thus, we obtained this Web400 data via the following procedure: we first grouped spreadsheets

[3]http://lemurproject.org/clueweb09.php

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 as the Web400 dataset.

5.1.1 Spreadsheet Properties on Web400 Dataset

Before doing experiments on the Web400 dataset, we investigated its spreadsheet properties as follows: we manually assigned correct spreadsheet properties to each Web400 sheet.[4] Among the 400 spreadsheets, we found 309 spreadsheets containing spreadsheet tables, while the rest included unfilled forms, text, and visualizations.

We had two major observations. First, we identified 21 simple spreadsheet properties that covered the transformation process from spreadsheet tables to relational tables for the 309 spreadsheets in Web400.[5]

Second, we observed that the five most popular properties covered the transformation process for 68% (209/309) spreadsheets. In the rest of the experiment, we only focused on these five properties for simplicity. These five properties include:

1. Aggregation Rows (agg_row) — An aggregation cell is defined as an aggregation function (*e.g.*, sum, and avg) over a group of cells. An aggregation cell could be indicated by spreadsheet formulas in an implicit way, for example, 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 1 has the property "agg_row" as discussed in Section 1.

2. Aggregation Columns (agg_col) — Similarly, a spreadsheet has the property "agg_col" if it has a column of aggregation cells.

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 1 has the property "hier_data" because "education attainment" in row 6 implicitly describes rows 7-17.

4. Hierarchical Header (hier_head) — Similarly, a spreadsheet has the property "hier_head" if there exists a cell in the header region implicitly describing another 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 1 has the property "crosstab" as discussed in Section 1.

5.2 Spreadsheet Property Detection

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 4.2. We also investigate how the quality of the user-provided rules affects the performance of our hybrid approach.

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

[5]The 21 spreadsheet properties are: aggregation rows, aggregation columns, hierarchical header, hierarchical data, crosstab, vertical split table, spanning cell, horizontal split tables, redundant column, redundant row, no header, truncated header, merge rows, duplicate headers, complicated hierarchical structure, horizontal orientation, mixed orientation, blank rows, redundant header, truncate data, complicated hierarchical header.

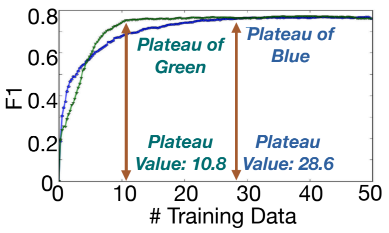


Figure 4: An example of "training size to plateau".

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 1: Crude user-provided rules for the five properties in Section 5.1.1.

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

Table 2: Four methods to build property detectors.

5.2.1 Experiment Setup

We tested the top five spreadsheet properties mentioned in Section 5.1.1. Our experiments were 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 simulated the iterative learning framework in Section 3.1 and measured the performance of the current trained classifiers for each iteration: we fed the 200 potential training spreadsheets as the spreadsheet dataset for the iterative learning framework. During each iteration, we calculated the F1 score of the currently trained classifiers on the 200 testing data. We simply used 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 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 [22]. 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

@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-noisy	56 (0%)	126 (-10%)	45 (+7%)	92 (-30%)	59 (+13%)
Hybrid-clean	**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-noisy	31 (+11%)	66 (+8%)	35 (+6%)	39 (-60%)	45 (+10%)
Hybrid-clean	**16** (-43%)	**52** (-15%)	**18** (-46%)	**22** (-78%)	**31** (-24%)

Table 3: The training size to plateau for four property detection methods with $\delta = 0.01$ and $\delta = 0.05$. The % represents the improvement over Active.

training data as $F1_{opt}$, and we simply set $\theta_{F1} = F1_{opt} - \delta$.

We tested our iterative learning framework using the four approaches as shown in Table 2. Rand randomly selects the next spreadsheet and does not use any user-provided rules; Active employs the uncertainty sampling active learning approach without considering user-provided rules; Hybrid-noisy and Hybrid-clean are our hybrid approach that integrates the uncertainty sampling active learning approach with crude user-provided rules. Hybrid-noisy assumes low-quality user-provided rules while Hybrid-clean assumes high-quality rules. For Hybrid-clean, we used the designed rules for each spreadsheet property as shown in Table 1; and for Hybrid-noisy, we used 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 ran 100 times to obtain the averaged F1 score for different sizes of training data, and we report the training size to plateau. Except for Hybrid-noisy, we ran 100 times with each of the four "bad" user-provided rules, totaling 400 times. We then report the average training size to plateau for the four configurations.

5.2.2 Single-task Learning

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

Table 3 shows the training size to plateau for the four testing methods. As shown in the table, Hybrid-clean significantly outperforms all the other three methods. It means that when a human 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 when averaged over all properties, compared Active. In addition, we can see Hybrid-noisy is comparable to the standard active learning approach Active, and it indicates that our hybrid approach is able to tolerate bad user-provided rules.

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 rest $200 \times (1 - 0.3)$ spreadsheets with the *false* 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 average F1 score to calculate the training size to plateau for each spreadsheet property detector.

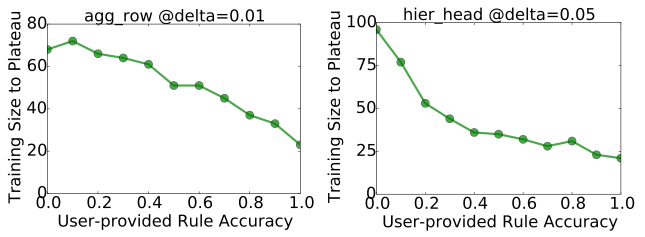


Figure 5: The quality of user-provided rules influences the training size to plateau.

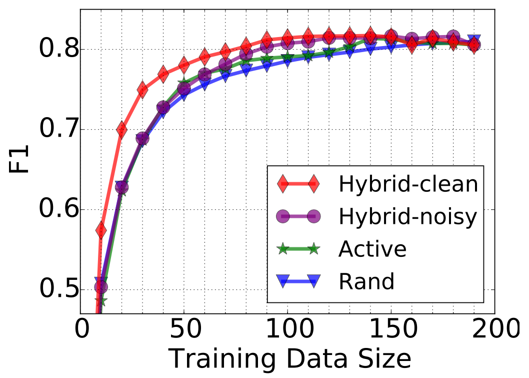


Figure 6: The F1 performance curve to learn the five property detectors together.

Figure 5 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.

5.2.3 Multi-task Learning

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

Figure 6 shows the F1 scores for different sizes of training data when learning the five property detectors together. As shown in the Figure, Hybrid-clean 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 approach Active. It indicates that "good" user-provided rules do save a significant amount of extra labeling work. In addition, Hybrid-noisy is comparable to Active, and it indicates that our hybrid framework can 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 when averaged over all properties to reach the performance plateau when a human provides relatively high-quality rules, and performs comparably with low-quality rules.

6. LARGE-SCALE SPREADSHEETS STUDY

In this section, we investigate the distribution of the five spreadsheet properties mentioned in Section 5.1.1 in the large-scale WebCrawl dataset. We evaluate the performance

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

of the five property detectors using Web400 data, and then show two observations on the large-scale WebCrawl data.

6.1 Experiment Setup

We obtained 1,181,530 spreadsheets from 410,554 .xls workbook files in the WebCrawl data.[6] We first recognize the spreadsheet tables in an input spreadsheet using the approach mentioned in [5], and then use the property detectors to collect the the spreadsheet property statistics.[7]

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: *accuracy* measures the percentage of spreadsheets which we correctly recognize whether it contains a given spreadsheet property; and *F1* measures the harmonic mean of precision and recall for each spreadsheet property.

Table 4 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 we used logistic regression as the classification model for the spreadsheet property detection. Note that accuracy is always higher than F1, because the spreadsheet properties are unbalanced: few positive examples and more negative examples.

6.2 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 portion of spreadsheets in the web which contain each of the five spreadsheet properties.* Figure 7 (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 7

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

[7]Note that if there are multiple spreadsheet tables in a spreadsheet, we only retain the first one.

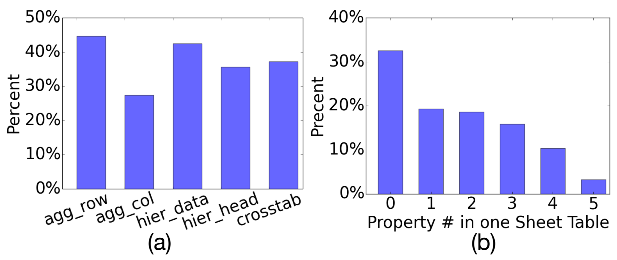


Figure 7: The distribution of the five spreadsheet properties in the web.

(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.

7. CONCLUSION AND FUTURE WORK

We have described a hybrid iterative learning framework to construct spreadsheet property detectors quickly, and it is the first step towards 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.

8. 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] J. Attenberg and F. 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.

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

[5] Z. Chen and M. Cafarella. Automatic web spreadsheet data extraction. In *Proceedings of the 3rd International Workshop on Semantic Search over the Web*, page 1. ACM, 2013.

[6] Z. Chen and M. 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.

[7] Z. Chen, M. Cafarella, J. Chen, D. Prevo, and J. Zhuang. Senbazuru: A prototype spreadsheet database management system. *Proceedings of the VLDB Endowment*, 6(12):1202–1205, 2013.

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

[9] P. Donmez, J. G. Carbonell, and P. N. Bennett. Dual strategy active learning. In *Machine Learning: ECML 2007*, pages 116–127. Springer, 2007.

[10] G. Druck, B. Settles, and A. 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.

[11] M. Gyssens, L. V. S. Lakshmanan, and I. N. Subramanian. Tables as a paradigm for querying and restructuring. In *PODS*, pages 93–103, 1996.

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

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

[14] L. V. S. Lakshmanan, S. N. Subramanian, N. Goyal, and R. Krishnamurthy. On query spreadsheets. In *ICDE*, pages 134–141, 1998.

[15] C. Li, Y. Wang, P. Resnick, and Q. 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.

[16] G.-J. Qi, X.-S. Hua, Y. Rui, J. Tang, and H.-J. 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.

[17] R. Reichart, K. Tomanek, U. Hahn, and A. Rappoport. Multi-task active learning for linguistic annotations. In *ACL*, volume 8, pages 861–869, 2008.

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

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

[20] K. Tomanek and U. 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.

[21] X. Zhang, T. Yang, and P. Srinivasan. Online asymmetric active learning with imbalanced data. In *KDD*, 2016.

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

[23] J. Zhu, H. Wang, T. Yao, and B. 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.