Interactive Repair of Tables Extracted from PDF Documents on Mobile Devices

Jane Hoffswell[1,2]
[1]University of Washington
Seattle, Washington
jhoffs@cs.washington.edu

Zhicheng Liu[2]
[2]Adobe Research
Seattle, Washington
leoli@adobe.com

ABSTRACT

PDF documents often contain rich data tables that offer opportunities for dynamic reuse in new interactive applications. We describe a pipeline for extracting, analyzing, and parsing PDF tables based on existing machine learning and rule-based techniques. Implementing and deploying this pipeline on a corpus of 447 documents with 1,171 tables results in only 11 tables that are correctly extracted and parsed. To improve the results of automatic table analysis, we first present a taxonomy of errors that arise in the analysis pipeline and discuss the implications of cascading errors on the user experience. We then contribute a system with two sets of lightweight interaction techniques (gesture and toolbar), for viewing and repairing extraction errors in PDF tables on mobile devices. In an evaluation with 17 users involving both a phone and a tablet, participants effectively repaired common errors in 10 tables, with an average time of about 2 minutes per table.

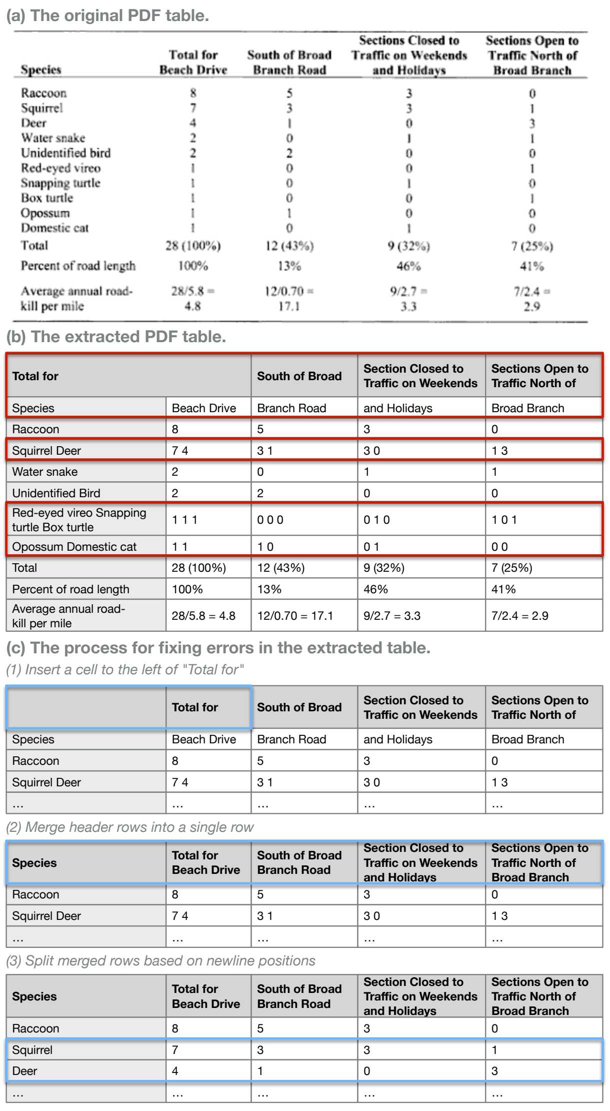
CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**; *Touch screens*; *Gestural input*; • **Applied computing** → *Document management and text processing*.

KEYWORDS

PDF; Data Tables; Table Classification; Error Taxonomy; Error Correction; Mobile Devices; Interaction Techniques

ACM Reference Format:
Jane Hoffswell and Zhicheng Liu. 2019. Interactive Repair of Tables Extracted from PDF Documents on Mobile Devices. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3290605.3300523



(a) The original PDF table.

Species	Total for Beach Drive	South of Broad Branch Road	Sections Closed to Traffic on Weekends and Holidays	Sections Open to Traffic North of Broad Branch
Raccoon	8	5	3	0
Squirrel	7	3	3	1
Deer	4	1	0	3
Water snake	2	0	1	1
Unidentified bird	2	2	0	0
Red-eyed vireo	1	0	0	1
Snapping turtle	1	0	1	0
Box turtle	1	0	0	1
Opossum	1	1	0	0
Domestic cat	1	0	1	0
Total	28 (100%)	12 (43%)	9 (32%)	7 (25%)
Percent of road length	100%	13%	46%	41%
Average annual road-kill per mile	28/5.8 = 4.8	12/0.70 = 17.1	9/2.7 = 3.3	7/2.4 = 2.9

(b) The extracted PDF table.

Total for		South of Broad	Section Closed to Traffic on Weekends	Sections Open to Traffic North of
Species	Beach Drive	Branch Road	and Holidays	Broad Branch
Raccoon	8	5	3	0
Squirrel Deer	7 4	3 1	3 0	1 3
Water snake	2	0	1	1
Unidentified Bird	2	2	0	0
Red-eyed vireo Snapping turtle Box turtle	1 1 1	0 0 0	0 1 0	1 0 1
Opossum Domestic cat	1 1	1 0	0 1	0 0
Total	28 (100%)	12 (43%)	9 (32%)	7 (25%)
Percent of road length	100%	13%	46%	41%
Average annual road-kill per mile	28/5.8 = 4.8	12/0.70 = 17.1	9/2.7 = 3.3	7/2.4 = 2.9

(c) The process for fixing errors in the extracted table.

(1) Insert a cell to the left of "Total for"

	Total for	South of Broad	Section Closed to Traffic on Weekends	Sections Open to Traffic North of
Species	Beach Drive	Branch Road	and Holidays	Broad Branch
Raccoon	8	5	3	0
Squirrel Deer	7 4	3 1	3 0	1 3
...	...	...	...	...

(2) Merge header rows into a single row

Species	Total for Beach Drive	South of Broad Branch Road	Section Closed to Traffic on Weekends and Holidays	Sections Open to Traffic North of Broad Branch
Raccoon	8	5	3	0
Squirrel Deer	7 4	3 1	3 0	1 3
...	...	...	...	...

(3) Split merged rows based on newline positions

Species	Total for Beach Drive	South of Broad Branch Road	Section Closed to Traffic on Weekends and Holidays	Sections Open to Traffic North of Broad Branch
Raccoon	8	5	3	0
Squirrel	7	3	3	1
Deer	4	1	0	3
...	...	...	...	...

Figure 1: (a) PDF documents often contain rich data tables. (b) However, techniques for extracting the data from static PDF tables can be error prone (errors are shown in red); in this example [27], the header has been split into multiple rows and several rows of the table have been incorrectly merged together (e.g., "Squirrel" and "Deer"). (c) To support reuse of the table data, the reader must first repair the errors by (1) inserting a new cell in the first row, (2) merging the header rows, and (3) splitting the incorrectly merged rows.

1 INTRODUCTION

Data tables are a rich source of information in PDF documents, but it is often difficult to digest or reuse such data. Unlike tables in web pages where the elements are properly tagged, tables in PDF documents are often represented as vector or bitmap graphics with no structural and semantic information. As a result, simple use cases such as extracting data from tables to be used outside of the PDF reader or indexing tables for search can involve tedious additional work. The current static PDF format, originally designed for printing or viewing on desktop monitors, is also not optimized for dynamic or mobile usage contexts. As mobile devices become pervasive, reading large tables on a tablet or a phone requires constant zooming and panning, while also requiring the reader to situate the table within the context of the document text. These cumbersome interactions break the flow of reading and can be cognitively demanding.

To address these challenges, "next-generation PDF" aims to introduce new flexibility to PDF documents through the use of web technologies [2]. Such flexibility could lead to new interactive applications to improve the reusability of document data and enhance the way readers engage with the document. In dynamic reading environments, the placement of supplemental information (e.g., footnotes, figures, or tables) can have a significant impact on the utility of this information [34]. Flexible PDF documents could support reflow of the contents enabling them to adapt to the accessibility needs of the user or the device form factor. Recent work has also explored methods to automatically link text and tables [20], approaches to generate contextual visualizations from tables [3], and interactive techniques for highlighting connections between text and visualization [21].

To enable interactive experiences for data tables requires a system that can understand the structure and semantics of tables. Automatic extraction enables generalized techniques to augment documents regardless of the underlying creation mechanism or document age. Unfortunately, achieving such an understanding is not straightforward. Despite recent advances in machine learning approaches [4, 5, 8, 10], accurate table detection and content extraction remain challenging. This problem is exacerbated by poor table schema and layout designs for the original document table. Consequently, automated table analysis is often error-prone, which in turn impacts downstream reuse of the table data.

We first review state-of-the-art techniques for automated extraction and analysis of PDF tables in the form of a six-stage pipeline: table detection, content extraction, type classification, structure analysis, data parsing, and finally interactive reapplication. We implement and run this pipeline on a corpus consisting of 447 labeled PDF documents. The results show that only 11 of the original 1,171 tables were correctly extracted, analyzed, and parsed. To better understand how to improve this process, we contribute a taxonomy of errors and complications that arise throughout the pipeline. We discuss the sources and significance of these complications and the impact of cascading errors on the analysis pipeline.

Based on our analysis, we argue that human input is necessary in the analysis pipeline for dynamic reapplication of PDF tables. We identify several errors from our taxonomy that can be interactively rectified on the smaller form factor of a mobile device, and contribute two approaches for lightweight table editing interactions. To evaluate these approaches, we conduct a comparative study with 17 participants and measure users' performance in terms of the time and number of actions required to complete the task. We found that participants could quickly fix even complex tables, with task times lasting about 2.22 minutes on average and only 3.85 minutes on average for the most challenging task. Across all 10 tasks, participants fixed the errors in each table using about 18.6 actions on average.

2 RELATED WORK

This paper leverages prior work from several domains including data table extraction and cleaning, research on mediating the display and resolution of recognition and inference errors, and work on gesture interaction interfaces and techniques.

2.1 Data Table Extraction and Cleaning

Processing data tables is a well studied area of related work, spanning table detection, classification of table types, extraction of relational data, and data cleaning. Tabula [1] is a tool that allows users to select tables for extraction from text-based PDF documents. He et al. [16] has explored how to automatically detect and extract tables from PDF documents.

DeepDive [25] is a system for constructing a knowledge-base of facts from the web. Extensive prior work has focused on classification of web tables and leverages the HTML formatting information in the classification [8, 10, 22]. These techniques use the table contents and stylistic information to build machine learning classifiers for the table types. Crestan and Pantel [8] present a taxonomy of nine types of web tables, whereas Eberius et al. [10] focus on first distinguishing data from layout tables, and then classifying the table type. We leverage many of these techniques in our analysis pipeline.

Chen and Cafarella explore techniques for automatically detecting hierarchies in spreadsheet data using stylistic information and similarities amongst cells [4]; their system Senbazuru [6] supports the process of extracting hierarchical data and enabling user repair operations for fixing incorrect hierarchies [5]. We are similarly interested in how users can resolve system errors in a semi-automatic table analysis pipeline; for our work, however, we examine a larger range of errors that arise on automatically extracted PDF tables.

Data wrangling techniques are often necessary to clean or restructure complex data. Wrangler [19] supports direct manipulation of data tables to produce complex data transformation scripts that can be reused across multiple tables; such approaches have now been realized in recent commercial systems [9, 30]. Our system is motivated by this work on data cleaning, but places an emphasis on lightweight mobile interactions rather than producing a fully robust system; our goal is for readers to quickly reach a usable state rather than to support the full range of data restructuring operations.

2.2 Mediating Recognition and Inference Errors

Errors or ambiguities often occur in complex systems. Prior work has explored approaches for mediating recognition errors for handwriting [28] and speech [15]. Mankoff et al. [24] present OOPS: a general purpose toolkit for designing ambiguity resolution approaches for systems relying on accurate input recognition (including speech and handwriting). Ambiguities are a common complication in natural language systems that can limit the effectiveness of the system. DataTone [14] supports the resolution of ambiguities in a system for generating visualizations from natural language input via ambiguity widgets. We add to this body of work by first considering the types of complications that arise in the table analysis process and then designing lightweight table editing operations to resolve some of these errors.

2.3 Interaction Interfaces for Mobile Data Analysis

For our table editing operations, we target lightweight modifications to the table on the fly using mobile devices. Interactive text editing has been extensively explored for mobile devices [7, 13, 33]. However, tables can be particularly complex and hard to navigate on mobile devices. Related work has therefore explored the use of speech [29], gesture [17, 18, 23], or handwriting [35] for editing tables on mobile devices. We extend this work with a system for fixing common extraction errors in table structures using both a toolbar and gestural approach. There is also a large body of related work on the design of gestural systems [31, 32] that can further inform the design of our table editing interactions.

3 BACKGROUND

Structural properties of a table are essential for determining how to properly extract the underlying data entities. In this section, we provide relevant background on the types of tables described in this work and their structural properties.

We leverage a table type taxonomy described by Crestan and Pantel [8] to categorize our tables. In this taxonomy, tables are defined as either **data tables** (e.g., tables with structured, relational knowledge) or **layout tables** (e.g., tables used purely for formatting). Throughout the analysis pipeline, we only focus on data tables and regularly remove

	Rock Creek	**Wise Road**
Total	19	5
Grey Fox	1	
Opossum	18	4
Box Turtle		1

(a) Matrix

Location	**Species**	**Roadkill Incidents**
Rock Creek	Grey Fox	1
Rock Creek	Opossum	18
Wise Road	Opossum	4
Wise Road	Box Turtle	1

(b) Vertical Listing

Location	Rock Creek	Rock Creek	Wise Road	Wise Road
Species	Grey Fox	Opossum	Opossum	Box Turtle
Roadkills	1	18	4	1

(c) Horizontal Listing

Location of Opossum Roadkills:	
Rock Creek	18
Wise Road	4

(d) Attribute-Value

Figure 2: An example (a) *matrix,* (b) *vertical listing,* (c) *horizontal listing,* and (d) *attribute-value* table. The cells corresponding to a single entity are highlighted in blue for each table; the procedure for extracting elements varies depending on the table type and structure.

layout tables from our analysis. In the original taxonomy, the data tables can be further subdivided as one of the following types: *vertical listing*, *horizontal listing*, *matrix*, *calendar*, *attribute-value*, *enumeration*, or *form*. For our analysis, we reclassify *forms* as layout tables and introduce the type *other* for data tables that are not clearly defined by another type.

The table type determines how data should be extracted using an entity-attribute-value approach. Figure 2 shows how the same data is represented in four types of tables. An **entity** is a roadkill record (highlighted in blue), which has three **attributes** (Location, Species, and Roadkill Incidents), and the corresponding **values**. Notice that attribute names are not included in a matrix. In the attribute-value table, the entity and attribute names are only included in the title.

In addition to the table type, the table orientation, the presence of hierarchies, and the rows or columns that represent headers and keys are all essential for understanding the table structure. The table **orientation** describes the layout direction of the entities and can be defined as either *vertical*, *horizontal*, or *not applicable.* In Figure 2, both the *vertical listing* and *attribute-value* tables have a vertical orientation, whereas the *horizontal listing* has a horizontal orientation, and the orientation of the *matrix* is not applicable.

Hierarchies can occur in the *rows*, *columns*, and/or *data*, and reflect additional relationships amongst the entities that are otherwise not explicitly defined; hierarchies may be denoted using strategies including indentation, cells that span multiple rows or columns, or textual annotations (such as "Total"), among others. The *matrix* table in Figure 2a includes a row hierarchy: the number of roadkill incidents for each location is a subset of the "Total" included in the second row.

Headers define the labels for the values of an entity. For example, in the *vertical listing* in Figure 2b, the header is the first row and the labels are: "Location", "Species", and "Roadkill Incidents"; for the *horizontal listing*, the headers appear in the first column, whereas for the *matrix* and *attribute-value* table, the headers are implicit. A **key** is an attribute

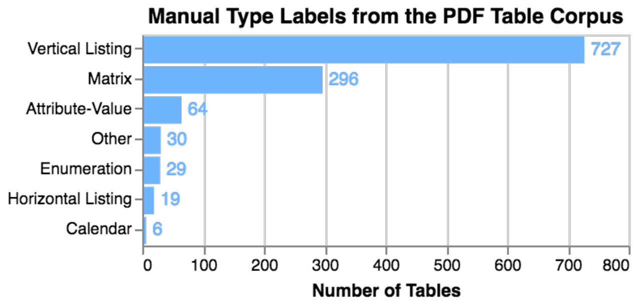


Figure 3: We labeled the type of all 1,171 data tables from our PDF table corpus. The vast majority of tables are *vertical listings*, with *matrices* being the second most common.

that uniquely identifies an entity. In some tables, multiple attributes are required to uniquely identify an entity, forming a composite key [11]. For example, Figure 2b has a composite key comprising the "Location" and "Species" attributes.

4 PDF TABLE CORPUS: MANUAL ANALYSIS

To gain a better understanding of PDF tables, we sampled and labeled a corpus of 447 PDF documents from an internal collection of 1.5M documents gathered for research and development. The corpus includes scientific publications, legal proceedings, presentation slides, and personal correspondences, of varying lengths ($max = 302, \mu = 18.6, \sigma = 32.9$ pages), and includes both scanned and text-based documents.

For each document, we manually identify the data tables and label the table *type*, the table *orientation*, whether the table included *hierarchies*, and the *headers* and *keys*. We consider tables that span multiple pages as a single table if they share the same title or number; if a multi-page table does not have a shared title or number, we count the table on each page separately. To complete the labeling, we split the documents and tables amongst the paper authors and discussed exceptional examples to reach agreement on the labels.

Number of Tables. Of the 447 documents in our corpus, 199 contained data tables, which resulted in a corpus of 1,171 PDF tables ($max = 73, \mu = 5.88, \sigma = 9.84$ tables per document).

Table Type. Figure 3 shows our labeled results. We found that the vast majority of tables in our corpus are of the *vertical listing* type (62.1%), with *matrices* as the second most common type (25.3%). The frequency of table types in our corpus is notably different than those examined by Crestan and Pantel [8]: Crestan and Pantel focus on labeling the type of all table structures found in a large-scale web crawl (many of which are purely for page layout), whereas we focus on classifying the types of data tables from PDF documents.

Orientation. We found that the majority of our tables have a *vertical* orientation (70.0%) and only a very small subset are arranged *horizontally* (2.14%). The orientation of the table is closely related to the table type, so this distribution of orientations reasonably reflects the table type results.

Hierarchies. Hierarchies occur in 44.9% of the tables in our corpus. Hierarchies were most common in the row structure of the table (32.0% of tables), but also occurred in the columns (20.2% of tables). Hierarchies in the data cells of the table were uncommon (only 3.07% of tables) and often appeared in tables with an unusual structure (e.g., *other*).

Headers and Keys. 10.2% of tables did not include any headers, and 4.70% did not include any keys. As one might expect, headers and keys frequently occurred in the first row or column of the table. 85.7% of tables included a header in the first row or column and only 49 tables (4.18%) had headers that did *not* include the first row or column. 26.9% of tables had headers in multiple rows or columns of the table. 86.4% of tables included keys in the first row or column of the table. Tables rarely had keys in multiple columns (only 7.86%), which generally occurred when tables repeated columns or when composite keys are necessary.

5 PDF TABLE ANALYSIS PIPELINE

Based on the above discussion, we propose and implement a processing pipeline that includes: (1) *detecting* and (2) *extracting* tables from PDF documents; (3) *classifying* the table to determine the table type and (4) *analyzing* the underlying structure; (5) *parsing* the table based on the structure to produce clean and usable data; and finally (6) *reusing* the extracted data for other interactive applications. The analysis pipeline is shown in Figure 4. In this section, we present the technical details and results of our pipeline implementation.

5.1 Detecting Tables in PDF Documents

We utilized the table detection and extraction process presented by He et al. [16], which uses a fully convolutional neural network to segment pages and then heuristically extract tables; a verification network helps to remove false positives, thus providing an effective and generalizable algorithm for table detection in generic PDF documents. After the detection phase of our analysis pipeline, the system found that 246 of the original 447 documents did not include any tables; The remaining 201 documents contained an average of 4.75 tables per document, thus producing a corpus of 954 extracted tables. Comparing these results to our manually labeled tables, we found that the detection phase correctly identifies 551 of the original 1,171 tables (47.1%).

5.2 Extracting Tables from PDF Documents

At the end of the extraction phase, each document is represented using a JSON structure. Each part of the JSON structure is labeled with a "tag" property describing the purpose of the component (e.g., "Table" or "Caption"). We search through the JSON to identify all the subcomponents that are tagged as a "Table" for analysis in the next phases of our pipeline. The extraction phase introduces many errors into

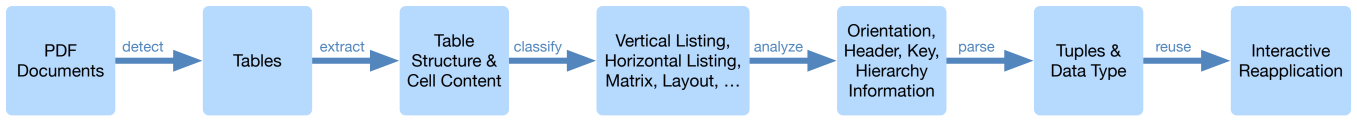

PDF Documents
detect
Tables
extract
Table Structure & Cell Content
classify
Vertical Listing, Horizontal Listing, Matrix, Layout, …
analyze
Orientation, Header, Key, Hierarchy Information
parse
Tuples & Data Type
reuse
Interactive Reapplication

Figure 4: The analysis pipeline includes several steps: (1) *detecting* tables in PDF documents and (2) extracting the tables. We analyze the automatically extracted tables first to (3) *classify* the table type, and then to (4) *analyze* additional structural properties of the predicted data tables: orientation, hierarchy, headers, and keys. Using these properties, we (5) *extract* the data entities and can finally (6) *reuse* the extracted results for new interactive applications.

the process; 126 of the 954 detected tables are successfully extracted without any errors (13.2%). The errors that are introduced during this phase are described in Section 6.

5.3 Classifying the Table Type

Once tables have been extracted from the document, we build a machine learning model to predict the table *type*. Following the procedures from related work [4, 8, 10], we compute numerous features based on the table contents and style (e.g., the frequency of numeric cells in a column or the maximum length of the cell contents). For a full list of features used in our classification, please refer the supplemental material. Following the advice of Eberius et al. [10], we used the computed features and our gold standard labels (described in Section 4) to train a random forest model to predict the type using Python's machine learning library *scikit-learn* [26].

Using all 954 extracted tables from our corpus, we start by splitting the corpus into quarters: we use three quarters of the dataset to train a random forest model and produce predictions for the remaining quarter. We repeat the training and prediction process four times with our subdivided dataset, each time using a different quarter of the dataset for testing. This procedure produces one prediction for each table in our corpus. We then repeat this procedure 10 times to produce 10 predictions for each table. We label the table with the table type chosen by the majority of trials. 78.6% of table types are correctly classified during the analysis. Figure 5 shows a comparison of the manually labeled table type with the predicted table type; the number of correctly predicted types are shown along the diagonal.

5.4 Analyzing the Table Structure

After completing the type classification, we filter out all tables that are predicted as a layout type to exclude them from further analysis; based on the predictions, filtering leaves us with 727 tables to be analyzed. We then use machine learning models and heuristic analysis techniques to compute additional structural properties of the table: the *orientation*, the presence of *hierarchies*, and the *headers* and *keys*.

Orientation. Using the same features and analysis procedure as the type, we use a random forest model to predict the orientation of each table based on 10 trials. Our prediction strategy for the orientation has an accuracy of 83.2%.

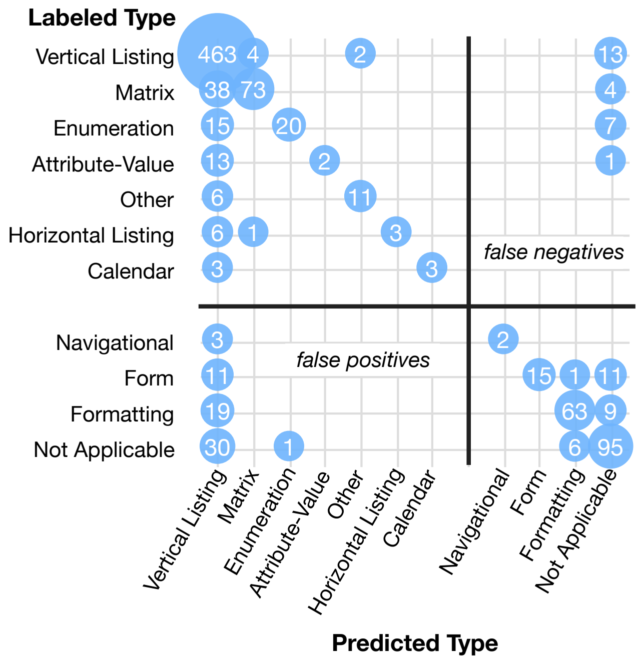


Figure 5: 78.6% **of table types are classified correctly (those along the diagonal). However,** 2.62% **of tables are misclassified as layout tables and will be incorrectly excluded from further analysis (e.g., *false negatives*) and** 6.71% **of tables are misclassified as data tables (e.g., *false positives*).**

Hierarchies. Using the same features and analysis procedure as the table type and orientation, we use a random forest model to predict whether or not tables include hierarchies in the rows, columns, or data based on 3 trials for each type of hierarchy. Even with a smaller number of trials, the hierarchy prediction for all three types of hierarchies is more accurate than the other structural properties. In the predicted data tables, 26.6% have row hierarchies, 15.4% have column hierarchies, and only 2.61% of tables included data hierarchies. We reach a prediction accuracy of 87.8% for labeling the presence or absence of row hierarchies, 90.0% for column hierarchies, and 98.2% for data hierarchies. Given that data hierarchies were so infrequent in our corpus, additional training data is needed to ensure the results are robust.

Headers. We explored several techniques from related work to identify the table headers. For our analysis pipeline, we use the Senbazuru framefinder [6] to label each row of the table with the row type (e.g., "Title", "Header", "Data", or "Footnote"). To do this, we first convert the table from the JSON structure to an Excel structure suitable for the Senbazuru framefinder. We then run the framefinder on the spreadsheets and identify any of the rows labeled with the type "Header" as the table headers. To determine the accuracy of our header detection, we compare the predicted results to the header labels for the *extracted* table rather than the original table. This comparison allows us to examine the accuracy of this analysis step, since some of the extracted tables may have erroneous removed headers, which thus cannot be identified by Senbazuru. We found that this strategy computes the correct headers for 40.0% of the extracted tables.

Keys. We leverage the technique described by the Web Data Commons [22] to calculate which columns are the keys for the table. This procedure looks at each column to identify a column that contains primarily unique, string values and has an average cell length between 3.5 and 200 characters. As with the headers, we compare the predicted results to the *extracted* key labels, since this comparison provides a more accurate representation of how the step performs on the data it was provided. Using this technique, we can correctly identify the table keys for 27.0% of the extracted tables.

5.5 Parsing the Table Data

For the data parsing stage of the analysis pipeline, as with the structural analysis, we only extract data entities for tables that were predicted to be data tables (727 of the original 954 tables). We have three different protocols for extracting the data entities based on the table orientation: *vertical*, *horizontal*, and *not applicable*. We currently do not attempt to extract data hierarchies, though this process has been explored in related work [4] and should be incorporated in future work.

To extract data from tables with a vertical orientation, we traverse each row that is not denoted as a header row. For each column, we lookup the property name in the matching header column. The procedure for extracting data from tables with a horizontal orientation is the same as that for a vertical orientation, but we traverse the columns instead of the rows. Matrix tables are neither vertical nor horizontal, so to extract data from these tables, we produce a unique data tuple for each cell in the table that is not denoted as a header or key; the value of the cell is thus stored as the "value" of the tuple. We then extract the cell contents for the headers and keys that correspond to the cell of interest, and store these in the data tuple as additional properties of the tuple.

6 ERRORS & COMPLICATIONS IN THE PIPELINE

Machine learning models commonly make inaccurate predictions. Even with correct analysis and parsing, the formatting of the table contents may constitute challenges for dynamic reuse of the data. In this section, we present a taxonomy of errors and complications that arise throughout the analysis process (Figure 6). We further discuss the downstream impact of errors introduced at each stage of the pipeline.

Class of Error	Error or Complication	Tables	Frequency	Affected Downstream Changes
Detecting Tables in PDF Documents				
Detection Errors	Table Not Detected	620	52.95%	Table unavailable for data reuse
	Detected Non-Table	200	20.96%	Result not appropriate for data reuse
Extracting Tables from PDF Documents				
Missing Table Data	Table Spans Multiple Pages	279	29.25%	Extracted data is incomplete
	Table Only Partially Extracted	116	12.16%	Extracted data is incomplete
Excessive Table Data	**Includes Non-Table Content**	179	18.76%	Incoherent extracted data values
	Merged Multiple Tables	13	1.36%	Incoherent extracted data values
Incorrect Table Structure	**Split Cell Into Multiple**	230	24.11%	Incoherent cell contents
	Merged Adjacent Cells	220	23.06%	Improper cell feature extraction
	Missing Whitespace Cells	32	3.35%	Structural mismatch between cells
	Cell Misalignment	5	0.52%	Uninterpretable table rows/columns
Incorrect Cell Content	Incorrect Symbol	67	7.02%	Does not accurately reflect original table
	Incorrect Number	22	2.31%	Misleading data values
	Incorrect Contents	22	2.31%	Misleading cell contents
Analyzing Table Type and Structure				
Type Prediction	Table Misclassified as Data Table	52	5.45%	Table not appropriate for data reuse
	Table Misclassified as Layout Table	22	2.31%	Data values not recognized as usable
Structural Identification	Incorrect Orientation	110	15.28%	Data entities improperly extracted
	Incorrect Hierarchies	154	21.39%	Structural relationships not recognized
	Incorrect Headers	440	61.11%	Data entities improperly labeled
	Incorrect Keys	521	72.36%	Data entities improperly identified
Parsing the Table Data for Downstream Data Reuse				
Parsing and Data Reuse Challenges	Repeated Table Structure	12	1.26%	Data entities incorrectly merged
	Cells Contain Units/Type	79	8.28%	Cells incorrectly typed
	Cells Annotate Missing Data	55	5.77%	Cells incorrectly typed
	Cells Contain Uncertainty	45	4.72%	Uncertainty influences data reuse
	Cells Contain Excessive Information	36	3.77%	Non-numeric values impact reuse
	Cells Contain Footnote	21	2.20%	Cells incorrectly typed

Figure 6: We identify eight classes of errors that occur during our analysis pipeline for *detecting*, *extracting*, *classifying*, *analyzing*, and *parsing* tables from PDF documents. For each type of error, we show the number of tables and the frequency of the source of error in our table corpus.

Only 11 tables from our original corpus of 1,171 are correctly extracted, analyzed, and parsed (0.94%). 551 tables were correctly detected (47.1%) and 111 were extracted without errors (20.1%). Of the remaining 111 tables, 88 have the type correctly predicted (79.3%) and 81 of those have the orientation correctly predicted (92.0%). From the remaining tables, 69 have the presence of hierarchies correctly annotated (85.2%). Identifying the headers and keys is one of the biggest hurdles in the analysis pipeline, with only 11 tables labeled correctly at the end of the pipeline (15.9%).

6.1 Detecting Tables in PDF Documents

We identified two sources of error when detecting tables in PDF documents (Figure 6, *Detection Errors*): TABLE NOT DETECTED (e.g., false negatives) and DETECTED NON-TABLE (e.g., false positives). Tables that are not detected at the start of our pipeline often exhibit deficient structural information in the extracted JSON. For example, the table contents may be collapsed into a single paragraph such that the structural information is not encoded. We therefore cannot easily reincorporate the table into the analysis pipeline, and thus need additional segmentation to extract the structure.

6.2 Extracting Tables from PDF Documents

Most of the tables in our corpus are plagued with extraction errors that impact the downstream effectiveness of our pipeline. From our corpus of 954 extracted tables, only 126 tables are extracted without any errors (13.2%). The errors labeled during the extraction phase fall into four high-level categories (shown in Figure 6): *missing table data*, *excessive table data*, *incorrect table structure*, and *incorrect cell content*.

Missing Table Data. The extraction sometimes fails to fully extract or recognize the coherence of a particular table, which leads to two distinct sources of error: TABLE SPANS MULTIPLE PAGES and TABLE ONLY PARTIALLY EXTRACTED. These errors are particularly problematic because they may mislead readers who believe that the extraction is complete. For example, visualizations of the partial results will produce an incomplete picture of the data and therefore cannot be trusted outright. In some of the tables, each page of the table would be individually extracted but not joined together in which case manual comparison across representations would be necessary. For some tables however, the page break would displace only a small portion of the table (e.g., a single row) that would then be overlooked by the detection process.

Excessive Table Data. In contrast to tables that are missing data, some tables include information that does not belong in the table. These extracted tables have often absorbed content from the surrounding area, such as the page header, table title, or values from a nearby table. Resolving these errors for proper using in new interactive applications requires the erroneousness data to be identified and shifted to the proper location in the document. When the extraction has MERGED MULTIPLE TABLES, the incorrect data may need to be properly added to a different table structure or extracted separately.

Incorrect Table Structure. Incorrectly extracting the table structure (or identifying structures that are not present) can lead to downstream errors when attempting to parse the table data. Each error in this class alters the original table structure in a different way, and therefore requires different actions to fix the error. While the CELL MISALIGNMENT error is uncommon in our corpus (only 5 tables), it often produced extreme alterations to the table structure and resulted from tables that were poorly structured in the original document.

Figure 1 exhibits the three most common *Incorrect Table Structure* errors. The header row of the table (which includes the cell "South of Broad Branch Road"), has been incorrectly split into two rows in the extracted table, due to the third line of the original header being incorrectly identified as a separate row. When the split on the header occurred, the extraction failed to add a whitespace cell to the first extracted row, which causes the text "Total for" to incorrectly span the first two columns. Finally, several rows of the table have been merged together: the rows corresponding to the entities "Squirrel" and "Deer" have had all of their contents merged.

Incorrect Cell Content. While the previous classes of errors have focused on the table structure, some tables exhibited errors in the cell contents themselves. These errors can have varying degrees of impact on the analysis and reuse of the table data. We found that mathematical symbols were often the hardest for the original document recognizer to identify and correctly reproduce in the extracted table. However, the error is often more problematic when the extraction incorrectly recognizes the cell number or contents. Alterations to the numbers can drastically change the meaning of a table, while not having a noticeable impact on the extracted result. The extraction also sometimes misplaces the cell contents, thus producing a result that can be analyzed and parse, but with the value misattributed to the wrong property or entity.

6.3 Classifying the Table Type

Erroneously labeled data tables (e.g., TABLE MISCLASSIFIED AS DATA TABLE) are *false positives*: tables which will be included in the analysis when they should have been removed. A small fraction of the type predictions are *false negatives* (e.g., TABLE MISCLASSIFIED AS LAYOUT TABLE): tables which will be incorrectly excluded from subsequent analysis. Figure 5 labels the quadrants corresponding to these errors.

6.4 Analyzing the Table Structure

To correctly parse the data, we must first understand the structural properties of the table. However, errors often arise in our prediction of the table structure for the data tables (Figure 6, *Structural Identification*). Each of these properties parameterizes our procedure for extracting the data into coherent data entities, so errors in the classification can lead to improperly parsed data, and thus incorrect data reuse.

6.5 Parsing the Table Data

Unlike the errors described previously, this class of errors does not lead to a misrepresentation of the table itself, but rather introduces downstream complications when trying to parse or reuse the table data (e.g., for dynamic visualization).

The REPEATED TABLE STRUCTURE error occurs when the table layout is reused in different parts of the table; for example, a table may include data for only two properties, but arrange the entities into four columns. In this case, multiple entities may occur on a single row of the table but only one will be recognized by the parsing behavior, thus leading to potential data loss in the parsed data. Recognizing this complication is essential for reflowing the table layout and fully extracting the underlying data.

The other four complications each require additional data cleaning before the underlying data can be reused appropriately (e.g., for further analysis or visualization). When cells contain units/type, it could be beneficial to display this information on the axis or legend of a visualization or use the information to correct the automatically interpreted data type or labels; similarly, when cells contain footnote, properly linking the extracted data with the table footnote can ensure that readers do not overlook information pertinent to their interpretation of the results. When cells contain uncertainty, it may be appropriate to design customized visualizations or to otherwise modify the way the data is reused in new interactive applications; however, different tables encode uncertainty information differently, thus making it harder for a system to understand and reproduce the data without additional support from the reader.

Figure 1 exhibits the cells contain units/type and cells contain excessive information complications. For example, the row that describes the "Percent of Road Length" includes the symbol "%" in each cell of the row; this annotation causes the cell contents to be interpreted as a string rather than a number. Similarly, the row "Average annual roadkill per mile" includes excessive information because the cells show the process of producing the average, in addition to the final value. These complications cause the cell data to generally be interpreted as strings rather than numbers, which impacts the computed features used in our analysis pipeline and may complicate downstream reuse of the extracted data.

6.6 Cascading Errors

Each stage of our analysis pipeline builds on the results of the previous stage, so errors at any point can lead to even more errors downstream. The impact of these cascading errors is shown in Figure 1: when data cells are merged in the extracted table, the individual cells are labeled with the incorrect data type; the data types are used in the classification procedure, which impacts the prediction of the table type and structural properties. The incorrect cell types also contributes to improper data parsing. In order to produce usable data that can be appropriately analyzed or reused in interactive applications, it is important to mitigate upstream errors to limit their subsequent impact.

7 ERROR CORRECTION ON MOBILE DEVICES

In order to mitigate errors introduce during the analysis pipeline, we contribute a system for fixing extraction errors on the fly. We propose two sets of lightweight table editing interactions on mobile devices: a set of gestures for directly manipulating the table contents and a toolbar based approach. While the gesture technique supports rapid editing interactions directly on the table, it requires readers to remember the mapping between operations and gestures. In contrast, the toolbar provides a context-sensitive list of operations that apply to the current selection, but displaces the reader's focus from the table to other parts of the device.

For our implementation, we focus on mobile interfaces, but also note that our proposed techniques also work out of the box on desktops. With mobile devices becoming pervasive for a variety of different tasks, they present an interesting challenge for both viewing next-generation PDF documents and for addressing the challenges that may arise from an automatic analysis pipeline. The smaller form factor and informal setting of a mobile device emphasize the need for interactive repair operations to be composable and lightweight, unlike more robust data cleaning systems.

7.1 Mitigating Seven Common Analysis Errors

We do not address all the errors from our taxonomy in the scope of this paper; instead, we focus on addressing structural errors in the table that directly impact the extraction of coherent data tuples (Figure 6, **in blue**). Unlike errors or complications from the table content itself, structural errors are amenable to rapid, gestural modifications which can be more easily reapplied to similar structural errors in other tables. We designed seven interactions for modifying the extracted table structure. The goal of these interactions is *not* to support the full range of operations required by a robust data cleaning process (such as [9, 19, 30]), but to allow for rapid, lightweight modifications on the fly. With the interactions we support, we can fix 1,627 errors from those identified in the tables. In doing so we improve the number of fully correct tables from 11 to 108 (e.g., 9 times the number of tables). In future work, we would like to explore interactions to address more of the errors from our taxonomy.

7.2 Lightweight Table Editing Operations

To view the extracted table, readers tap directly on the table in the original document. The extracted table is then displayed below the original table; readers can toggle the visibility of the original and extracted table using the toolbar that appears at the bottom of the screen. As the reader selects parts of the extracted table, they can modify the structure using gestures or by selecting from the recommended actions in the toolbar at the bottom. Each action in the toolbar is associated with an icon [12].

Select. To make any changes to the table, readers must first select the rows, columns, or cells they want to change. Cells are selected by tapping on them. Swiping left or right from an unselected cell selects the row; swiping up or down from an unselected cell selects the column.

Insert. The errors missing whitespace cells and cell misalignment directly impact the structural layout of the table contents. Merging or splitting cells can also introduce new structural anomalies in the table that require the user to insert new cells to rearrange the structure. To insert cells relative to a selection, the reader drags the selection in the direction it should move; for example, in Figure 1, to insert

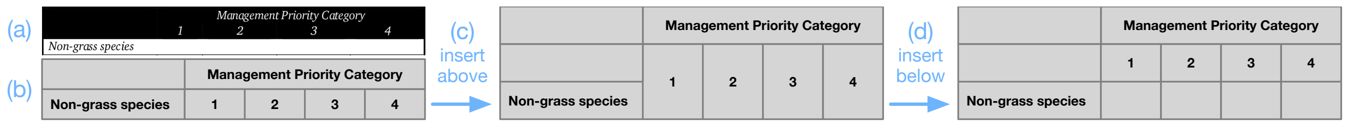


Figure 7: (a) The original table. (b) The extracted table. (c) The reader inserts a cell above "Non-grass species", which only inserts a cell in the first column. (d) Inserting cells below the numbers leaves "Non-grass species" in its own row.

a cell to the left of "Total for", the reader drags the cell to the right →. The reader may also select "insert" + on the toolbar and then choose the direction (e.g., "left" →). The icons indicate the direction the selection will move rather than where the new elements will be added.

Inserting a row or column produces an empty row or column with the same number of cells as the original selection. Inserting a cell adds a new cell in the specified direction. The insertion first reduces the span of the original cell if it spanned multiple rows or columns; in Figure 1c.1 inserting a cell to the left of "Total for" causes the "Total for" cell to align with only one column of the table rather than spanning the first two columns. If the selected cell does not span multiple columns, the insertion only adds a new cell to the original row or column of the selected cell. In Figure 7c, inserting a cell above "Non-grass species" only inserts a cell in the first column; to move "Non-grass species" to a row by itself, the reader can then insert cells below each number (Figure 7d).

Merge. To fix the SPLIT CELL INTO MULTIPLE error and merge cells back together, the reader uses a pinch gesture or selects "merge" from the toolbar. Merging adjacent cells merges the contents of the cells. When merging adjacent rows, the cells are merged along the table columns; for merging adjacent columns, the cells are merged based on the row. To merge adjacent rows or columns, the selections must have the same number of cells. For example, in Figure 1, the reader must first insert a cell into the first row (Figure 1c.1) to ensure that the two header rows each have five cells; the reader can then merge the header rows together, which merges the contents in each column (Figure 1c.2).

Split. To fix the MERGE ADJACENT CELLS error and split the contents back apart, the reader uses a zoom-out gesture or selects "split" from the toolbar. When applying the split operation to row selections, the row is split based on line breaks in the original document. For example, in Figure 1, several of the data rows were merged in the original extraction (e.g., "Squirrel" and "Deer"); using a split on these rows breaks up the contents based on the text positions in the original document. For column selections, the split occurs based on spaces in the cell contents. For cell selections, the split first looks for new lines in the cell contents; if no new lines are found, the split occurs based on spaces.

Delete. To address the complication that the table INCLUDES NON-TABLE CONTENT, we support a delete operation on the table cells. Delete also allows readers to remove parts of the original table to focus their analysis. To delete a selection, the reader double taps the selection or selects "delete" 🗑 in the toolbar. Deleting a selected row or column is straightforward. When a cell in the deleted selection spans multiple rows or columns, the cell still exists in rows or columns that were not deleted. For example, in Figure 7, deleting the column containing the cells "Management Priority Category", "4", and "" removes the cells "4" and "", but leaves "Management Priority Category" spanning the cells "1", "2", and "3". Deleting individual cells requires other cells in the table to expand into the vacated space. This expansion occurs based on the table type; for example, in vertical tables (in which the entities are represented as rows), the cells within the row expand into the vacated space.

Fix Header/Key. The analysis pipeline frequently encountered the error INCORRECT HEADERS or INCORRECT KEYS. Cells labeled as headers and keys are colored gray, with headers in bold and keys in italic. Readers can select rows or columns and use a long press to mark them as a header or key, or the reader can select "is header" or "is key" ↑ on the toolbar. The reader can use a double tap to remove the header or key status, or select "not header" or "not key" ↓ on the toolbar.

Other Operations. From the toolbar at the bottom of the screen, the user can also "undo" ↺ the previously performed action or change the visibility of the extracted table by modifying the "mode" ⚙. Finally, the reader can change the "type" ⊞ of the table (e.g., "vertical", "horizontal", or "matrix"), which changes parsing behavior for the table data. We represent the table type visually by varying the thickness of the table borders. In a *matrix*, we have a thick border around every cell to show that each cell is a unique entity. For a *vertical listing*, we place a thick border above and below the row, and thin borders between columns to indicate that each row is a unique entity; *horizontal listings* are similar, but with the opposite orientation.

8 EVALUATION OF TABLE EDITING TECHNIQUES

To evaluate the design and utility of our two table editing approaches for resolving errors in automatically extracted PDF tables, we conducted a user study with 17 participants. In this section, we discuss our study methods and experimental results. We have included the session script and post-task questionnaire in the supplemental material.

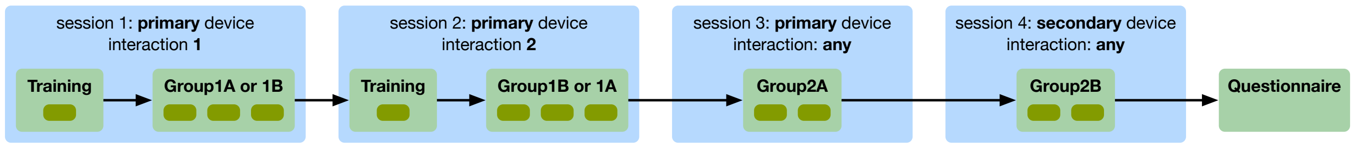


Figure 8: The evaluation procedure: participants were asked to identify and fix errors in ten tables using different interaction techniques and devices; we counterbalanced the devices (tablet and phone), techniques (toolbar and gesture), and table groups across participants, as described in the text. Participants then completed a post-study questionnaire about their experience.

Participants. We recruited 17 participants (8 female, 9 male) that had prior experience with data tables and with reading PDFs on their mobile devices. Participant ages ranged from 21 to 36 ($\mu = 27.1, \sigma = 3.86$). Participants were all college graduates – Bachelors (4), Masters (12), or P.h.D (1) – and most were still pursuing additional degrees (14). Participants received a $25 gift card for completing a one hour session.

Tasks. We selected 10 tables from our corpus of extracted PDF tables. We separated the tables into four groups that correspond to the different parts of our evaluation procedure. GROUP1A and GROUP1B each include three tables that are ordered based on their difficulty, where we define difficulty as the number and variety of errors exhibited in the table. The groups are defined such that each pair of tables exhibit similar errors. For example, the first table in each group has two errors: the table requires one row to be marked as a header, and one header row to be marked as data, but the tables themselves are different. The second table in each group has four errors, and requires a combination of insert and merge operations to repair. Finally, the last table in each group also contains four errors, but requires at least three distinct operations to fix. GROUP2A and GROUP2B each have two tables; the first table in each group is relatively simple (3 errors) and the second is more complex (7+ errors).

Procedure. Every participant performed three table repair sessions on a primary device (phone or tablet) and one additional session on a secondary device (Figure 8). We counterbalanced the primary and secondary devices across participants. Devices were provided by the authors; the tablet was a 10.5 inch iPad Pro and the phone was a 4.7 inch iPhone 7.

Using the primary device, each of the first two sessions focused on one interaction approach (GESTURE or TOOLBAR), and involved training for the interactions and one task consisting of three tables (GROUP1A or GROUP1B). We counterbalanced the order of the interaction approaches and tasks across participants. During the training in each session, participants were shown a sample table in our table editing interface. We described the errors in the table and the strategy for fixing the errors; the participant followed along by performing the table editing operations on the table as they were described. For the task in each session, participants were shown three tables (either GROUP1A or GROUP1B); for each table, the participant first inspected the original and extracted tables, and described the errors they saw and their plan for fixing them. Participants then fixed the errors using the table editing techniques they had just learned.

In the third session with the primary device, the participants were shown two tables in GROUP2A and were asked to identify and fix errors using any of the two interaction approaches they preferred. Participants could switch between the interaction techniques at any time. Finally, the participants switched to the secondary device and used any interaction approach to fix the two tables in GROUP2B.

We logged the table editing interactions that the participant performed, the time of each operation, and the technique (TOOLBAR or GESTURE) that was used. After completing all the tasks, participants filled out a post-study questionnaire with their general impressions about the table editing experience and demographic information. Participants also rated the ease of use and their preferences for the various interaction techniques and devices on a 5-point scale.

8.1 Quantitative Results & Analysis

We processed the participant results to only count the core actions performed; we filtered out the selections and clear actions since participants often performed these actions accidentally when navigating on the page, or without using a follow-up action when examining the table. For each table, we define the completion time as the amount of time between the first and last core action performed on the table. Participants performed 18.6 core actions ($\sigma = 21.8$) on average per table, with an average completion time of 2.22 minutes ($\sigma = 2.27$) per table, across all 10 tables in the study.

We first analyzed the results of the first two study sessions in which participants were limited to one interaction technique (GESTURE or TOOLBAR) per session. We used linear mixed-effects models for the number of actions performed to repair the table and the log task time. Each model includes fixed effects for the device (TABLET and PHONE), condition (GESTURE and TOOLBAR), and log task order, plus random intercepts for the subject predicated on the condition, and the table. The task order is defined from 1 to 12, where 1 and 4 are the two training tasks. Our analysis script and data are included in the supplemental material.

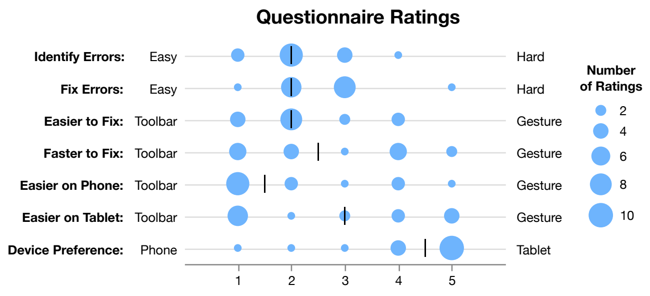


Figure 9: The median score (in black) and distribution (in blue) of participant ratings on the post-task questionnaire.

Across the 6 tables in the first two sessions, participants performed an average of 21.0 core actions ($\sigma = 24.9$) per table, with an average completion time of 2.39 minutes ($\sigma = 2.63$) per table. We found a significant effect of the condition on the number of actions performed ($\chi^2(2) = 9.38, p < 0.01$), with participants performing fewer actions when using the toolbar. We also found a significant effect of the condition ($\chi^2(2) = 21.1, p < 0.001$) and log task order ($\chi^2(2) = 20.5, p < 0.001$) on the log completion time for the task, with participants faster when using the toolbar than the gestures. However, we did not find a significant effect of the device on the log task time or number of actions performed.

In the last two sessions when participants had the option to choose between using the gestures or toolbar, participants choose the toolbar for 57.4% of the repair actions. The preference for the toolbar was also the case for many of the actions individually, with one notable exception: when using the tablet, participants used the insert gesture more frequently than using insert on the toolbar. Whereas on the toolbar, participants had to select insert and then the direction, the gesture manipulation was more fluid with a simple drag. We ran a linear mixed-effects model on the percent of gesture interactions used, with fixed effects for the device (TABLET and PHONE), condition for Session 2 (GESTURE or TOOLBAR), plus random intercepts for the subject predicated on the condition, and the table. We found that the training condition for Session 2 had a significant effect on the percent of gestures used ($\chi^2(2) = 7.09, p < 0.01$), with participants using fewer gestures after having just completed the toolbar condition.

After completing all tasks, participants rated their impressions of the interaction techniques and devices in a questionnaire. We used 1-sample nonparametric Wilcoxon signed rank tests with a null hypothesis that the result is neutral (e.g., 3 on a 5-point scale) on the self-reported results for the post-task questionnaire. We found significant positive effects on how easy participants found it to identify ($median = 2, p < 0.01$) and fix ($median = 2, p < 0.1$) errors in the tables. Participants generally found the toolbar easier to use than the gestures ($median = 2, p < 0.05$) overall and particularly on the phone ($median = 1.5, p < 0.05$). Participants significantly preferred using the tablet over the phone ($median = 4.5, p < 0.01$). Figure 9 shows the median score and distribution of the participants' ratings.

8.2 Qualitative Results & Discussion

We summarize some of our main takeaways from participants' responses to the post-task questionnaire; participants provided free text responses, so more participants may have agreed with any given opinion if asked directly.

Thirteen participants mentioned that one of the largest challenges was in learning and remembering the gestures; P16 explained that *"It takes some time to get used to and remember the gesture but once I get used to it, it is quite intuitive."* The gestures were designed to mirror the underlying behavior and to enable more direct modification on the table itself. Eight participants felt the gestures were quite natural and eight participants explained that they were useful for rapidly editing the table. P2 explained that *"I liked that [the gestures are] intuitive and fast (and fun!)."*

The available gestures and corresponding actions were sometimes hard to remember, so eleven participants particularly liked that the toolbar provided a reminder of what could be done; P11 explained that *"I liked that it was easy to see the available options and [the toolbar] helped me figure out how to fix problems."* In addition to providing a reminder of what could be done, the toolbar helped users create a plan for how to approach the table editing process and suggestions of what actions would work well for their current selection.

The gestures and toolbar each provide a unique table editing experience, and three participants explicitly expressed a preference for using them both. When both options were available, P4 said that *"I used gestures for simpler actions, and the toolbar for more involved operations"* and P16 also noted that *"When I get more familiar with the gesture and the toolbar, I can combine both in a pretty efficient way and make the editing go faster."* While we primarily evaluated the techniques separately, the designs are complimentary within the table editing system and could be effectively used together in the last two sessions of the study.

While participants overwhelmingly preferred the tablet for the table editing interactions, they generally mentioned that in terms of actual use they would be more inclined to use a phone. To explain this preference, P7 noted that the *"tablet would be easier, but I rarely have one with me"*. When asked about their use of mobile devices for viewing PDF documents, only P17 reported using only a tablet. While several participants did like the utility for editing tables on the fly, three participants expressed an interest in the availability of these techniques for the desktop (*"my real preference"* - P7). For our study, we were interested in exploring techniques for supporting lightweight mobile interactions for table editing, but our techniques also work out of the box for desktops.

9 LIMITATIONS AND FUTURE WORK

The table editing approaches presented and evaluated in this paper support lightweight editing for common extraction errors. The goal of this work was not to produce a fully functional data wrangling platform (e.g., [9, 19, 30]), but to explore techniques for rapid editing on mobile devices. For more complex analysis of tabular data, customized data wrangling or visualization systems may be more effective.

Our system presents a proof of concept about how readers might interact with and benefit from a system that can recognize the document contents for eventual reuse in interactive applications. Future work should explore additional techniques for correcting a wider variety of errors from our taxonomy; in particular, we would like to support re-extraction of tables that were not originally detected, the ability to join data across multiple extracted tables, and how to address errors or complications in the table content itself.

In order to support more accurate extraction of table data also requires future work at other stages of the analysis pipeline. For example, we currently do not adequately handle hierarchies in the data tables, so these relationships may be lost in our extracted data. However, related work has examined techniques for this type of data extraction [4]. Our pipeline would also benefit from additional work on the identification of table headers and keys.

During the evaluation, many participants found it tedious to apply similar actions for correcting errors in multiple parts of the table; we would like to explore techniques to support reapplication of table editing interactions to similar errors in the table or across similar tables. An end-to-end system should be able to incorporate lessons learned from readers back into the original analysis pipeline to reduce the number of downstream errors that are presented to a reader.

10 CONCLUSION

In this paper we described a taxonomy of errors and complications from the table extraction and analysis process. We found that on a corpus of 1,171 PDF tables, only 11 tables were correctly processed at the end of the analysis pipeline. To mitigate some of these cascading errors, we present two complimentary approaches for lightweight table editing on mobile devices. In an evaluation with 17 participants, we show that readers can repair tables with even complex extraction errors in about 2.2 minutes on average per table.

ACKNOWLEDGMENTS

We would like to thank the Adobe PDF Mobile Team and other members of Adobe Research for their interest and feedback on this work. We would also like to thank our reviewers, the UW Interactive Data Lab, and many others for their helpful comments on this paper.

REFERENCES

[1] Manuel Aristarán, Mike Tigas, Jeremy B. Merrill, Jason Das, David Frackman, and Travis Swicegood. 2018. Tabula. https://tabula.technology/.

[2] PDF Association. 2017. About next-generation PDF. https://www.pdfa.org/next-generation-pdf/.

[3] Sriram Karthik Badam, Zhicheng Liu, and Niklas Elmqvist. 2018. Elastic Documents: Coupling Text and Tables through Contextual Visualizations for Enhanced Document Reading. In *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*. http://www.zcliu.org/elasticdoc/elastic-documents-InfoVis18.pdf

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

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

[6] Zhe Chen, Michael Cafarella, Jun Chen, Daniel Prevo, and Junfeng Zhuang. 2013. Senbazuru: a prototype spreadsheet database management system. *Proceedings of the VLDB Endowment* 6, 12 (2013), 1202–1205.

[7] Gennaro Costagliola, Mattia De Rosa, and Vittorio Fuccella. 2018. A technique for improving text editing on touchscreen devices. *Journal of Visual Languages & Computing* 47 (2018), 1–8.

[8] Eric Crestan and Patrick Pantel. 2011. Web-scale table census and classification. In *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 545–554.

[9] Datawatch. 2018. Datawatch: Data Intelligence to Fuel Your Business. https://www.datawatch.com/about-us/.

[10] Julian Eberius, Katrin Braunschweig, Markus Hentsch, Maik Thiele, Ahmad Ahmadov, and Wolfgang Lehner. 2015. Building the dresden web table corpus: A classification approach. In *Big Data Computing (BDC), 2015 IEEE/ACM 2nd International Symposium on*. IEEE, 41–50.

[11] Ramez Elmasri and Shamkant Navathe. 2010. *Fundamentals of Database Systems* (6th ed.). Addison-Wesley Publishing Company, USA.

[12] Inc. Fonticons. 2018. Font Awesome. https://fontawesome.com/.

[13] Vittorio Fuccella, Poika Isokoski, and Benoit Martin. 2013. Gestures and widgets: performance in text editing on multi-touch capable mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2785–2794.

[14] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G Karahalios. 2015. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, 489–500.

[15] Christine A Halverson, Daniel B Horn, Clare-Marie Karat, and John Karat. 1999. The beauty of errors: Patterns of error correction in desktop speech systems.. In *INTERACT*. 133–140.

[16] Dafang He, Scott Cohen, Brian Price, Daniel Kifer, and C Lee Giles. 2017. Multi-Scale Multi-Task FCN for Semantic Page Segmentation and Table Detection. In *Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on*, Vol. 1. IEEE, 254–261.

[17] Lilong Jiang, Michael Mandel, and Arnab Nandi. 2013. Gesturequery: A multitouch database query interface. *Proceedings of the VLDB Endowment* 6, 12 (2013), 1342–1345.

[18] Jaemin Jo, Sehi L'Yi, Bongshin Lee, and Jinwook Seo. 2017. TouchPivot: blending WIMP & post-WIMP interfaces for data exploration on tablet devices. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 2660–2671.

[19] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts. In *ACM Human Factors in Computing Systems (CHI)*. http://idl.cs.washington.edu/papers/wrangler

[20] Dae Hyun Kim, Enamul Hoque, Juho Kim, and Maneesh Agrawala. 2018. Facilitating Document Reading by Linking Text and Tables. In *ACM User Interface Software & Technology (UIST)*. https://juhokim.com/files/UIST2018-TextChartRef.pdf

[21] Shahid Latif, Diao Liu, and Fabian Beck. 2018. Exploring Interactive Linking Between Text and Visualization. *Computer Graphics Forum (Proc. EuroVis)* (2018). https://www.vis.wiwi.uni-due.de/uploads/tx_itochairt3/publications/091-094.pdf

[22] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A large public corpus of web tables containing time and context metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 75–76.

[23] Erietta Liarou and Stratos Idreos. 2014. dbTouch in action database kernels for touch-based data exploration. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*. IEEE, 1262–1265.

[24] Jennifer Mankoff, Scott E Hudson, and Gregory D Abowd. 2006. Interaction techniques for ambiguity resolution in recognition-based interfaces. In *ACM SIGGRAPH 2006 Courses*. ACM, 6.

[25] Feng Niu, Ce Zhang, Christopher Ré, and Jude W Shavlik. 2012. DeepDive: Web-scale Knowledge-base Construction using Statistical Learning and Inference. *VLDS* 12 (2012), 25–28.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[27] United States. National Park Service. 2005. *Rock Creek Park (N.P.), Rock Creek Park and the Rock Creek and Potomac Parkway Project General Management Plan: Environmental Impact Statement*. https://books.google.com/books?id=CDY3AQAAMAAJ

[28] Michael Shilman, Desney S Tan, and Patrice Simard. 2006. CueTIP: a mixed-initiative interface for correcting handwriting errors. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*. ACM, 323–332.

[29] Arjun Srinivasan, Bongshin Lee, and John Stasko. 2018. Facilitating Spreadsheet Manipulation on Mobile Devices Leveraging Speech. In *Data Visualization on Mobile Devices Workshop (MobileVis) at CHI'18*. ACM.

[30] Trifacta. 2018. Trifacta. https://www.trifacta.com/.

[31] Jacob O Wobbrock, Htet Htet Aung, Brandon Rothrock, and Brad A Myers. 2005. Maximizing the guessability of symbolic input. In *CHI'05 extended abstracts on Human Factors in Computing Systems*. ACM, 1869–1872.

[32] Jacob O Wobbrock, Meredith Ringel Morris, and Andrew D Wilson. 2009. User-defined gestures for surface computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1083–1092.

[33] Dongwook Yoon, Nicholas Chen, and François Guimbretière. 2013. TextTearing: opening white space for digital ink annotation. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 107–112.

[34] Polle T Zellweger, Susan Harkness Regli, Jock D Mackinlay, and Bay-Wei Chang. 2000. The impact of fluid documents on reading and browsing: An observational study. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 249–256.

[35] Emanuel Zgraggen, Robert Zeleznik, and Philipp Eichmann. 2016. Tableur: Handwritten spreadsheets. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2362–2368.