A Collaborative Framework for Structure Identification over Print Documents

Maeda F. Hanafi
NYU Abu Dhabi
maeda.hanafi@nyu.edu

Miro Mannino
NYU Abu Dhabi
miro.mannino@nyu.edu

Azza Abouzied
NYU Abu Dhabi
azza@nyu.edu

ABSTRACT

We describe Texture, a framework for data extraction over print documents that allows end-users to construct data extraction rules over an inferred document structure. To effectively infer this structure, we enable developers to contribute multiple heuristics that identify different structures in English print documents, crowd-workers and annotators to manually label these structures, and end-users to search and decide which heuristics to apply and how to boost their performance with the help of ground-truth data collected from crowd-workers and annotators. Texture's design supports each of these different user groups through a suite of tools. We demonstrate that even with a handful of student-developed heuristics, we can achieve reasonable precision and recall when identifying structures across different document collections.

ACM Reference Format:
Maeda F. Hanafi, Miro Mannino, and Azza Abouzied. 2019. A Collaborative Framework for Structure Identification over Print Documents. In *Workshop on Human-In-the-Loop Data Analytics (HILDA'19), July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3328519.3329131

1 INTRODUCTION

Sealed in print is a vast amount of inscrutable information waiting to be surfaced, identified, searched and linked. Extracting this information is not easy; While there are many tools and techniques for automatically extracting information from documents that are largely structured or at least have a well-defined mark-up such as web documents and web tables [5, 20], we are yet to construct methods that can scalably handle the large variety and absence of a marked-up structure in print documents. Existing approaches either (i) give up on structure entirely, utilizing for example natural language processing to infer the semantics of the text within the document [24] (ii) or examine structure in a siloed fashion, developing complex and sophisticated techniques to determine a specific structural element such as a figure, a table, or other within a given document collection [11, 27].

In Texture, we adopt a two-staged approach to extracting information from collections of documents such as books, magazines, articles, receipts, etc. that are intended for print and are stored and shared digitally as PDF documents. Our first stage involves identifying structural elements from the collection such as title, section titles, tables, figures, captions, headers, footers, lists, etc. with the help of multiple, independently authored and boosted weak heuristics, and structure annotations provided by the crowd or expert annotators. Our second stage allows users to construct, in a domain specific language, simple extraction rules over the identified structures: these rules can even be automatically generated from examples with the help of known wrapper induction techniques.

```
extract (l/listitem i).content as ingredient
from CookBooks/document d/list l
where l below[1] d/sectiontitle s
and lowercase(s.content) like '%ingredient%'
and lowercase((d/title  t).content) like '%stir-fry%'
```

Listing 1: A rule in Texture's Structure-Based Extraction Language (SBEL) for extracting ingredients from lists in cookbooks.

```
extract (d/ title t).content as name,
   entity(p.content, 'organization') as institution[2]
from Resumes/document d
where (d/* p) below d/sectiontitle s[3]
and lowercase(s.content) like '%education%';
```

Listing 2: A SBEL rule that extracts educational institutions where an applicant studied at but did not work at.

1.1 Motivating Examples

We illustrate the benefit of our two-staged approach with the following motivating examples:

Example 1.1. Chef Remy wishes to create a new stir-fry recipe using a data-driven approach. He has a collection of 500 cookbooks and wishes to investigate what are the commonly used ingredients when making stir-fry. With Texture, he simply specifies the rule in Listing 1 to extract most of the information he needs.

Example 1.2. Layla, an HR administrator, is compiling a table of applicant names and the educational institutions they attended from a collection of 50 resumes. With Texture, she specifies the rule in Listing 2 to extract the relevant information.

Without Texture, these end-users can still extract their data using any of the following approaches:

(1) Train an ML model to automatically label the relevant data. Such ML solutions, however, require large training data sets, which may not be readily available. Creating a training data set may not be a justifiable endeavor given the personal, and one-off nature of the use-cases Texture is designed for.

HILDA'19, July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6791-2/19/07…$15.00
https://doi.org/10.1145/3328519.3329131

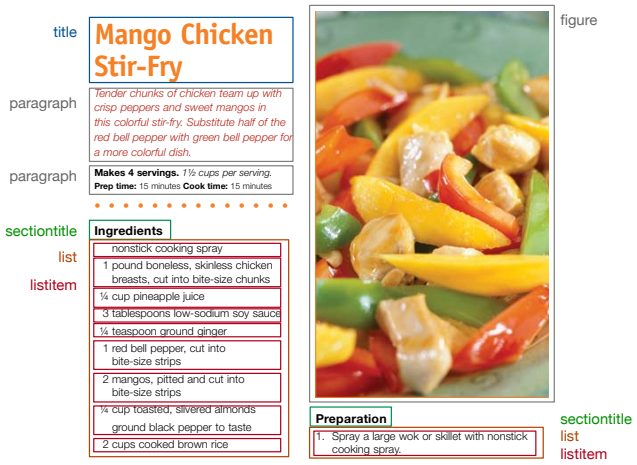

Figure 1: A recipe from Chef Remy's collection of 500 cookbooks. Texture **identifies different structural elements such as document title, section titles, lists and list items to allow him to only extract the ingredients of stir-fry recipes using Listing 1.**

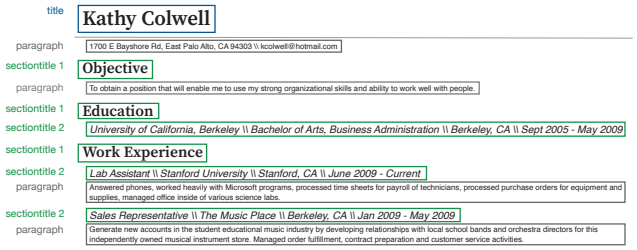

Figure 2: A resume from which Listing 2 can extract educational institutions attended.

(2) Crowd-source the data extraction task. Crowd-sourcing data extraction is not trivial even if we provide a simple tool that creates HITs for every page in the end-user's collection of documents, automatically stores the labeled text in a backend database and provides agreement measures for each label. This is because specifying the task non-ambiguously and constructing quality assurance and validation checks are task-specific and often require several trial and error runs. Moreover, with private or copyrighted collections, sharing the documents with the crowd may not even be an option. Finally, as the size of the collection grows, this solution does not scale well as we are bottle-necked by the number of crowd workers available and the time and cost required to label a page.

(3) Hire freelance developers to write data extraction code. Unfortunately, our experience with this approach has been disappointing: Handling PDF documents and data extraction tasks requires expertise that most freelance developers do not have[4]. Recruiting and hiring an experienced developer not only takes time but can be quite expensive. Moreover, most of the data extraction code developed does not generalize to other collections with different styles and is difficult to adapt to new data extraction tasks. Thus, this may not be a viable option for our end-users that want quick solutions for their adhoc data extraction requirements.

[4]We engaged with multiple developers on different freelance platforms like Freelancer and Upwork and hired five developers to complete different data extraction tasks.

1.2 Design Considerations

If we can accurately identify structure, then we can enable end-users like Chef Remy and Layla to construct powerful and meaningful data extraction rules from the imputed structure over print documents. In this work, we address only the first stage of our two-staged approach, i.e. the challenging problem of accurately identifying structure in print documents. We describe the key factors that influenced the design of Texture's interface and implementation:

(1) Multi-role Support & Division of Labor: End-users like Remy and Layla are interested in data extraction over their uploaded data collection; they may be incapable of writing heuristics to extract structure but they would like to search, apply and evaluate different structure identification heuristics and may even be willing to provide a few structure annotations themselves or with the help of the crowd to boost the performance of the heuristics. Developers can contribute open-source heuristics to Texture's shared code repository to enable such end-users. Texture provides support for each of these roles through (i) the heuristics interface which allows one to search and apply different heuristics, (ii) the framework's code repository, where developers can upload and unit-test their Java-coded heuristics and (iii) the CrowdCollect and SelfLabel tools that allow for crowd sourcing structure identification and manually self-labeling structural elements within the collection, respectively.

(2) Tolerance for Imperfection: Creators of print documents often strive to create visually appealing and distinctive documents, which results in a plethora of document styles and layouts within and across different document collections. In a single CACM publication, there are at least four different layouts and styles for document titles alone (See Figure 5). Even though many templates for CVs exist, job applicants often add their own flare to stand out. Despite such variations, English print documents often follow certain conventions: text flows from left to right, paragraphs are contiguous blocks of words separated by white space, section titles are often stylized differently from other text, lists are often numbered or bulleted, etc. Thus, we expect that simple heuristics can identify structures but that no single one is comprehensive enough to accurately identify a structure across multiple documents and that the performance of heuristics varies greatly across different collections. This means that we need to tolerate noise and boost the performance of these heuristics for a given document collection utilizing either ground-truth data or heuristics agreements and disagreements.

(3) Independent & Graphical Structural Annotations: As multiple user-groups collaborate on structure identification, it is crucial to maintain a single unifying representation of identified structural elements. In Texture, all heuristics and hand-labeled annotations are represented as graphical bounding-boxes over regions on a page. We chose this representation instead of content-based representations (such as text-highlights) to create a unifying representation of different structural elements (both figures and text are covered

by boxes), and to ease the manual annotation process — drawing bounding boxes is much easier and faster than trying to highlight text. Our bounding box representation also allow us to use spatial indexing techniques to efficiently determine (a) if multiple annotations overlap, and (b) the spatial relationships between structures and hence the logical layout of a document.

(4) Flexbile Workflow System: To enable developers to focus on the semantics and behavior of heuristic, rather than low-level implementation details, we need to provide a library of basic primitives and manage heuristic execution and data management. Texture provides a library of basic primitives such as methods for identifying words and lines in a document, and basic statistics such as distributions of font size, line height, line spacing, etc. Texture handles heuristic execution within a workflow to handle order constraints (e.g. to identify figures before captions to support a caption heuristic that identifies text immediately below or above a figure as a caption), prioritizes the order of documents fed into the heuristic workflow, and handles the storage of structures identified.

Our two key contributions in this paper are, first, a description of our collaborative framework and how it supports different users performing their roles from writing structure identification heuristics, to searching, applying and evaluating them and, second, a qualitative evaluation of the heuristics written in our framework by five undergraduate students: we discuss the process as well as the complexity and the quality of the heuristics.

2 OVERVIEW

Texture supports collaboration across multiple user groups for structure identification. Figure 3 shows an overview of the different components involved in the process of structure identification, which we now describe:

2.1 Human-driven Structure Identification

Texture allows both crowd workers and annotators, who may be the end-users themselves, to label the different structures within a document with the help of bounding boxes using CrowdCollect and SelfLabel respectively. End-users can use these hand-labeled structures to create a ground-truth data set against which heuristics within the shared code repository are evaluated and later boosted.

Texture simplifies the process of collecting structure labels from the crowd through CrowdCollect. Unlike generic data extraction tasks, it is much easier to generate HITs for structure identification because these structures are not task-specific and we have a general understanding of what they are within English print documents. Thus, we can easily create HITs that include tutorials for guiding crowd-workers on how to label structural elements, training materials as well as validation and quality assurance checks as seen in Figure 4.

To improve worker efficiency and accuracy, we limit each HIT to a specific structural element and we require at least five workers to perform the same HIT. Workers cannot proceed with a HIT if they fail the training phase. We require workers to draw bounding boxes that (i) completely enclose a structural element such as a paragraph, (ii) do not contain other elements and (iii) have tight margins (i.e. have as little surrounding white space as possible). If we detect any overlapping boxes, we ask the worker to re-draw the boxes. We drop a worker's entire HIT if they failed our hidden validation task by not correctly labeling all relevant structures on a page that we have previously labeled.

End-users can provide a budget and CrowdCollect determines the sample of documents to issue to the crowd for labeling. End-users may also modify the content of the HITs, including providing their own labeled validation pages. Currently Texture generates an Amazon Mechanical Turk kit, which end-users have to manually upload to the platform and launch themselves. In the future, we hope to seamlessly integrate with the platform.

After the HITs are launched and completed, Texture stores the labeled structures and ensures that a minimum level of agreement across multiple workers for each stored labeled structure (see Section 2.3).

2.2 Shared Heuristic Repository

Developers can contribute heuristics to the code repository as long as they are written in Java and they adhere to our minimalist `Heuristic` class interface. There are no restrictions on how a heuristic identifies a structure, or its performance on a hidden dataset. This is to encourage developers to contribute any number of heuristics that may be tailored to a specific domain. For each heuristic contribution, developers are required to provide a unit-test and a corresponding test document set[5]. To support the development of heuristics, our framework provides basic primitive methods such as `getContent(<box> B)`, which returns the text or image within a bounding box, `getFontFeatures(<box> B)`, which returns the different font types and styles used within a box, `getFontTypes(<document> d)`, which returns a map of the font types used in a document and their frequency, etc. The framework automatically controls execution order if a heuristic relies on the identification of other structures before its execution. In the future, we hope to support heuristic development in Python.

2.3 Data model

For a given document collection, Texture stores all identified structures in a Postgres database. We represent structures as bounding boxes within a document defined by a tuple (k, d, p, B, F, l, h), where

- k is a unique box identifier,
- d is the document identifier,
- B is a rectangular bounding box defined by its width, height, and 3-D coordinates (page, x, y),
- F is a set of properties that describe the box's content (e.g. 'fonttype: Times New Roman; fontsize: 12 pt;...' or 'nested-in: k''),
- l is the label assigned to the bounding box (e.g. 'title' or 'table'), and
- h identifies the heuristic, crowd-worker or annotator that generated the bounding box.

Our flexible data model allows custom structures with different properties in F. For example, table rows and columns may have an index property and section titles may have a depth level. Note that this data model allows a multiplicity of labels: different heuristics

[5]In the future we intend to measure how similar this test document set is to an end-user's document collection, to allow us to automatically suggest heuristics to apply

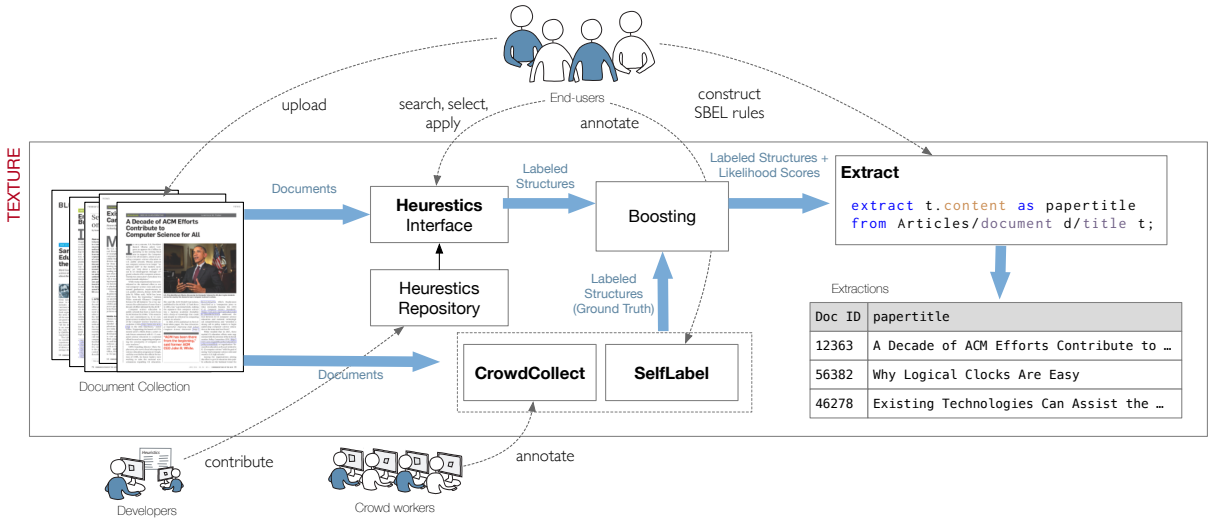

Figure 3: An overview of the framework, its components and the collaboration across different user groups that it supports.

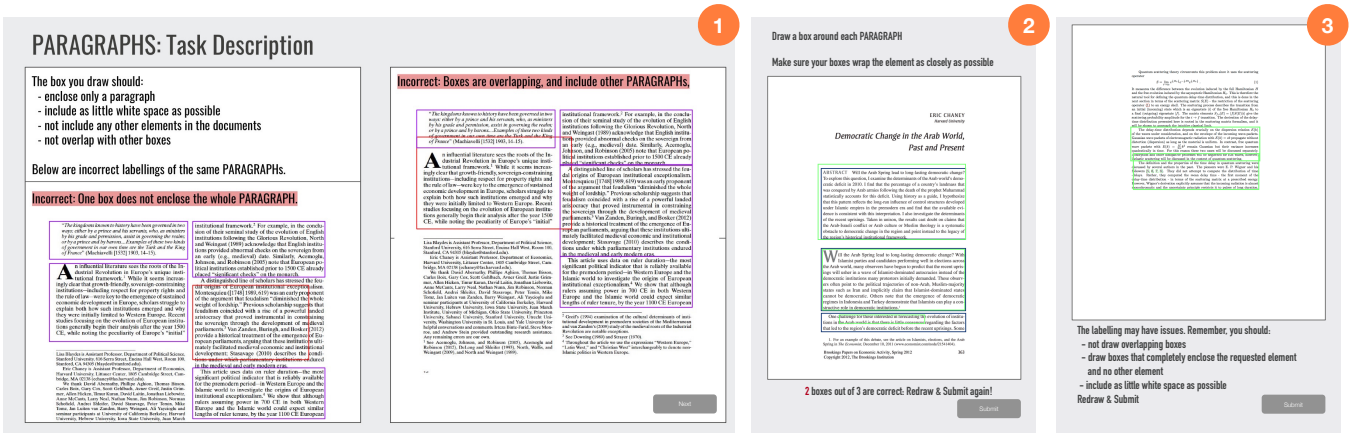

Figure 4: Texture's CrowdCollect **tool automatically generates an Amazon Mechanical Turk kit that includes (1) a tutorial for guiding crowd workers on how to label a structural element (2) a post-tutorial training phase that ensures crowd workers label elements as accurately as possible and (3) a secret validation task that provides quality control.** CrowdCollect **also suggests a grouping of pages into a HIT that optimizes for labeling quality, number of HITs generated and price per HIT. After the tasks are completed,** CrowdCollect **filters out poor quality labels.**

can label the same document region with the same or different labels.

Our framework provides primitive methods over bounding boxes such as `same(A, B)` which determines if two boxes identify roughly the same region, accounting for white space margins or `overlap(A, B)` which determines if two boxes have overlapping regions.

2.4 Heuristics Interface

Texture allows end-users to search and apply different heuristics. End-users can choose heuristics based on recency, description, authorship, etc. On selecting one or more heuristics, the document preview shows users the different structures labeled by these heuristics on the selected document. If a ground truth data set is available for their document collection, end-users can optionally boost the performance of their selected heuristics or default to Texture's majority vote. As seen in Figure 5, Texture displays the following performance statistics:

(1) *Execution Time:* Texture provides information on the execution time of different heuristics per page and how much of the document collection has been analyzed. With a large document collection and a selection of long-running heuristics that perform complex image analysis for example, structure identification can take hours. Thus, to ensure interactivity, we prioritize structure identification on documents currently under preview and we provide progress information.

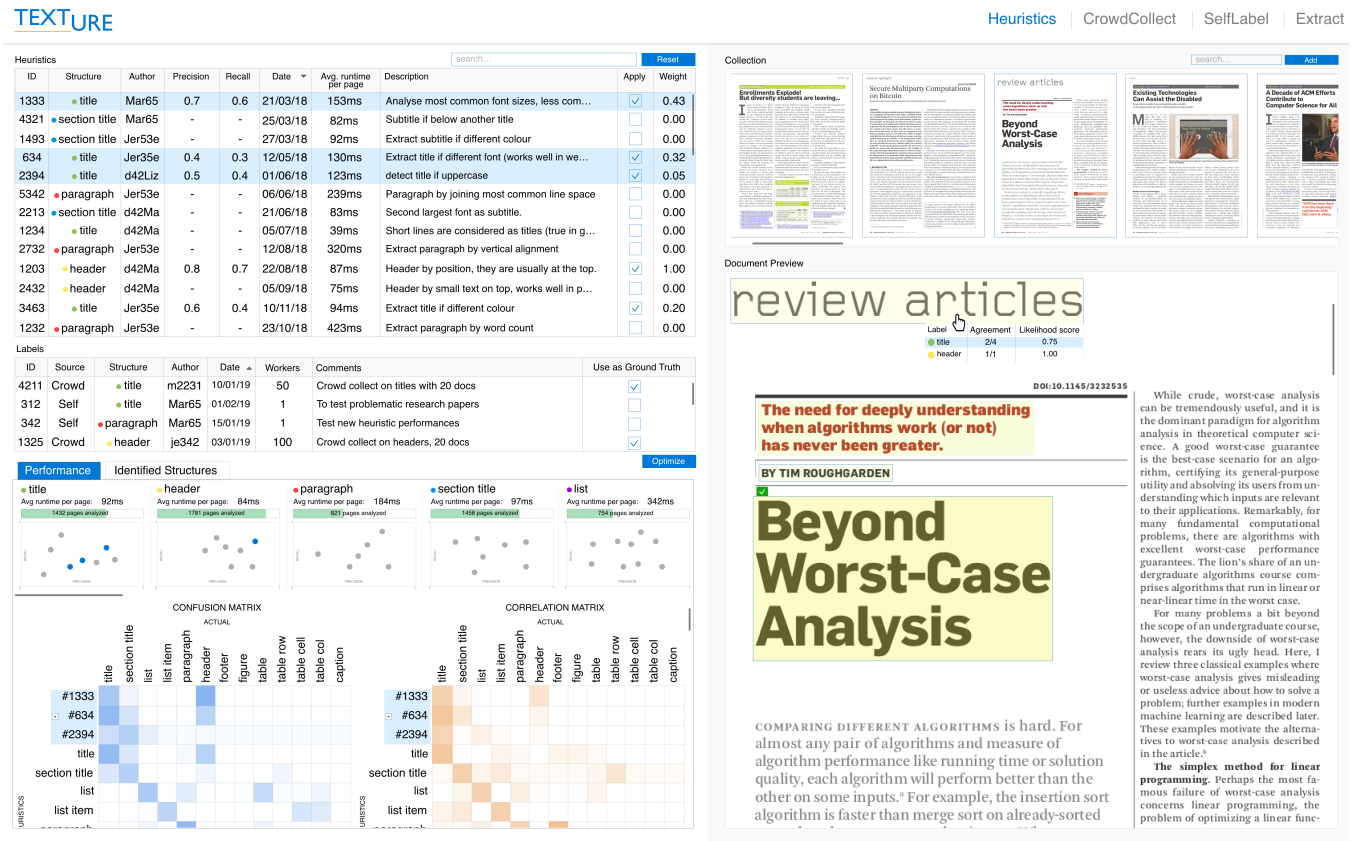

Figure 5: Texture's User Interface. Texture **provides (i) support for searching, applying and boosting multiple structure identification heuristics, (ii)** SelfLabel, **a tool for manually annotating documents quickly, (iii)** CrowdCollect, **a tool for quickly constructing crowd tasks for labeling different structural elements and (iii)** Extract, **a tool for writing data extraction rules in SBEL and, in the future, automatically learning SBEL rules from example extractions.**

(2) Precision and Recall Plots: If a ground-truth data set is provided, we display precision and recall measures with respect to this data set in the form of multiple PR plots for each structure. Since the heuristics search table is linked with these visualizations, we highlight the points representing each heuristic currently selected in blue.

(3) Confusion Matrix: If a ground-truth data set is available, we provide a confusion matrix that shows how often the heuristics labeled it correctly and what other structures it was mistaken for. Without ground truth, we simply present how often one class was labeled as another. See Figure 6 for an illustration of how the matrices are computed. These visualizations help users determine the best combination of heuristics to use given their time constraints and structures of interest. In the future, we hope to further automate the application of heuristics that provide the best performance taking into consideration not only the characteristics of the end-user's document collection but also their data extraction rules.

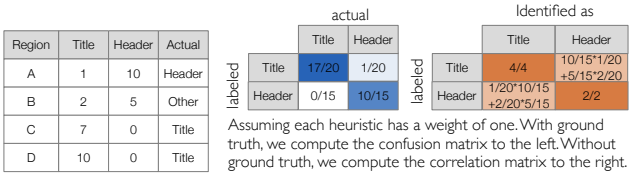

Figure 6: The Computation of Confusion and Correlation Matrices and Heat Map Visualizations

3 EVALUATION

We studied how five senior CS undergraduates developed heuristics within the Texture framework. Each student was assigned one or two structures for a total of eight structures. While some students re-implemented complex heuristics from research papers [8], others wrote simple ones. Some utilized existing libraries such as the Stanford CoreNLP library [1]. Across the structures, we found that heuristics often shared similar strategies. For instance, heuristics for headers, footers, and titles analyzed the distribution of font size. Some heuristics for tables and figures analyzed lines that started with "Fig" or "Table". Table 1 shows the number of heuristics per element and provides the average lines of code as a measure of code complexity: most heuristics were less than a 100 lines of code. Table

2 provides for each heuristic a brief description of the strategy employed. These observations affirm our second design consideration; Since heuristics can encode common conventions of how English print documents are structured with varying degrees of complexity and accuracy across document collections, Texture needs to support multiple weak heuristics and tolerate their individual imperfections.

Data Set and Ground Truth Collection. For each structure, we sampled 100 pages from four document collections: academic and white papers, course syllabi, cookbooks and processor specification sheets. The total cost for the structure identification was 448 USD. We utilized a total of 800 crowd workers to complete 80 HITs consisting of 10 pages each. Each HIT was completed by 10 different workers.

Heuristics Performance. Table 1 shows the average precision and recall for heuristics for each structural element as well as precision and recall when taking the union of all heuristics for a particular structure and when taking the majority vote. We find that we can generally improve precision and recall by such straightforward schemes even when dealing with documents from multiple domains.

Structure	n	Avg LOC	Avg Prec.	Avg Recall	Union Prec.	Union Recall	MV Prec.	MV Recall
Title	4	81	0.95	0.59	0.94	0.81	0.95	0.68
Paragraph	3	88	0.78	0.74	0.87	1	0.71	0.6
List	6	55	0.87	0.68	0.89	0.93	0.93	0.84
Header	6	57	0.85	0.57	0.78	0.83	0.98	0.7
Footer	6	50	0.95	0.34	0.91	0.72	0.97	0.41
Figure	5	89	0.95	0.37	0.9	0.75	0.92	0.65
Caption	3	89	0.92	0.36	0.94	0.78	1	0.28
Table	3	174	0.81	0.49	0.86	0.65	0.93	0.45

Table 1: For each structure, students developed n heuristics. We show here the average lines of code (LOC) as a measure of code complexity. We also show average precision and recall, precision and recall when considering the union of each heuristic group and when considering the majority vote (MV) $\geq \lfloor \frac{n}{2} \rfloor$. Note that while union improves recall at the cost of precision, the MV generally improves precision and with a few exceptions slightly improves recall.

4 RELATED WORKS

Texture's goal is to assist end-users in structure identification for the purposes of more accurate data extraction. There are many existing works in text extraction, [6, 14, 19, 21], and existing techniques such as Wrapper Induction[20] that are focused on text extraction over documents with varying levels of structured-ness from tabulated data to free-flowing text. However, existing systems tailored towards text extraction do not necessarily support structure identification, which is an important step in reducing errors especially in print documents.

There are existing works that do focus on identifying structure, specifically document understanding [3, 30]. Such works involve geometric and document structure analysis and identifying the labels of detected box regions. Texture is similar but also aims to support the usage of such existing algorithms in a collaborative setting. In the Appendix we briefly describe a variety of techniques studied to identify specific document structures: we view this literature as complementary to our work and we encourage the integration of these techniques as heuristics within Texture.

Similar to CrowdCollect are crowd data collection tools like Shreddr [4]. Shreddr is a system for digitizing papers in low-resource organizations. Because data entry workers do not necessarily think about the meaning behind what they are transcribing, Shreddr limits the task for each worker to transcribe a single field at a time in order to limit the risk of confusion. CrowdCollect limits each worker's identification task to only a single structure at a time and introduces a variety of error checking strategies (see Figure 4).

Our work is influenced by the data programming paradigm [25] and its first end-to-end implementation, Snorkel [28]. Snorkel, allows domain experts to label machine learning training data sets with labeling functions or heuristics. Without any ground-truth data, Snorkel learns, from the agreements and disagreements of the labeling functions, a generative model that allows it to estimate their correlations and accuracies. Snorkel then uses a discriminative model to generalize beyond its labeling functions. Snorkel's implementation, however, makes it difficult to construct labeling functions over specific document regions and stylistic features as it only supports text extraction.

Like Texture, Fonduer [29] extracts data with the help of labeling functions from richly formatted documents after representing documents in a unified structured data model. However, it is not clear how Fonduer identifies such structures in print documents that do not have a markup. Moreover, Fonduer's implementation requires users to abide to the DAG representation of the document structure. Texture's data model allows the same region to be labeled as different structures and does not restrict users to the system's interpretation of regions, allowing the user to prefer one label over another depending on his preferred interpretation of the region and the data extraction task hand.

Our work complements data-programming systems like Snorkel and Fonduer as we provide structure identification over print documents: this allows higher-level functions for data extraction or knowledge base construction to be developed over now-structured print documents. Data programming tools and Texture both encourage the use of heuristics to encode domain knowledge, and distinguishes itself from other systems by providing human-in-the-loop and collaborative tools at different stages through the data extraction pipeline, including collection of ground truth data.

5 CONCLUSION

In this paper, we presented Texture, a system that enables collaboration for structure identification and text extraction. Texture comes with a suite of tools: an interface for searching, applying, and boosting multiple structure identification heuristics, SelfLabel and CrowdCollect to collect ground truth data, and Extract to write extraction rules over the identified structures. We qualitatively evaluated Texture and showed that student developers can write structure identification heuristics with high precision and recall.

REFERENCES

[1] [n. d.]. Stanford CoreNLP. https://stanfordnlp.github.io/CoreNLP/.
[2] Evgeniy Bart and Prateek Sarkar. 2010. Information Extraction by Finding Repeated Structure. In *Proceedings of the 9th IAPR International Workshop on*

Document Analysis Systems (DAS '10). ACM, New York, NY, USA, 175–182. https://doi.org/10.1145/1815330.1815353

[3] Roldano Cattoni, Tarcisio Coianiz, Stefano Messelodi, and Carla Maria Modena. 1998. Geometric layout analysis techniques for document image understanding: a review. *ITC-irst Technical Report* 9703, 09 (1998).

[4] Kuang Chen, Akshay Kannan, Yoriyasu Yano, Joseph M Hellerstein, and Tapan S Parikh. 2012. Shreddr: pipelined paper digitization for low-resource organizations. In *Proceedings of the 2nd ACM Symposium on Computing for Development*. ACM, 3.

[5] Zhe Chen and Michael Cafarella. 2013. Automatic Web Spreadsheet Data Extraction. In *Proceedings of the 3rd International Workshop on Semantic Search Over the Web (SSW '13)*. ACM, New York, NY, USA, Article 1, 8 pages. https://doi.org/10.1145/2509908.2509909

[6] Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick R. Reiss, and Shivakumar Vaithyanathan. 2010. SystemT: An Algebraic Approach to Declarative Information Extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL '10)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 128–137. http://dl.acm.org/citation.cfm?id=1858681.1858695

[7] Christopher Clark and Santosh Divvala. 2015. Looking Beyond Text: Extracting Figures, Tables and Captions from Computer Science Papers. https://aaai.org/ocs/index.php/WS/AAAIW15/paper/view/10092

[8] Christopher Clark and Santosh Divvala. 2016. PDFFigures 2.0: Mining Figures from Research Papers. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries (JCDL '16)*. ACM, New York, NY, USA, 143–152. https://doi.org/10.1145/2910896.2910904

[9] J. Fang, L. Gao, K. Bai, R. Qiu, X. Tao, and Z. Tang. 2011. A Table Detection Method for Multipage PDF Documents via Visual Seperators and Tabular Structures. In *2011 International Conference on Document Analysis and Recognition*. 779–783. https://doi.org/10.1109/ICDAR.2011.304

[10] Fabio Fumarola, Tim Weninger, Rick Barber, Donato Malerba, and Jiawei Han. 2011. Extracting General Lists from Web Documents: A Hybrid Approach. In *Proceedings of the 24th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems Conference on Modern Approaches in Applied Intelligence - Volume Part I (IEA/AIE'11)*. Springer-Verlag, Berlin, Heidelberg, 285–294. http://dl.acm.org/citation.cfm?id=2025756.2025792

[11] Robert P. Futrelle, Mingyan Shao, Chris Cieslik, and Andrea Elaina Grimes. 2003. Extraction, Layout Analysis and Classification of Diagrams in PDF Documents. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2 (ICDAR '03)*. IEEE Computer Society, Washington, DC, USA, 1007–. http://dl.acm.org/citation.cfm?id=938980.939530

[12] Liangcai Gao, Zhi Tang, Xiaofan Lin, Ying Liu, Ruiheng Qiu, and Yongtao Wang. 2011. Structure Extraction from PDF-based Book Documents. In *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries (JCDL '11)*. ACM, New York, NY, USA, 11–20. https://doi.org/10.1145/1998076.1998079

[13] Vidhya Govindaraju, Ce Zhang, and Christopher RÃŕ. [n. d.]. Understanding Tables in Context Using Standard NLP Toolkits.

[14] Maeda F. Hanafi, Azza Abouzied, Laura Chiticariu, and Yunyao Li. 2017. SEER: Auto-Generating Information Extraction Rules from User-Specified Examples. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 6672–6682. https://doi.org/10.1145/3025453.3025540

[15] D. He, S. Cohen, B. Price, D. Kifer, and C. L. Giles. 2017. Multi-Scale Multi-Task FCN for Semantic Page Segmentation and Table Detection. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 01. 254–261. https://doi.org/10.1109/ICDAR.2017.50

[16] Jane Hoffswell and Zhicheng Liu. 2019. Interactive Repair of Tables Extracted from PDF Documents on Mobile Devices. In *ACM Human Factors in Computing Systems (CHI)*. https://homes.cs.washington.edu/~jhoffs/papers/2019-InteractiveTableRepair-CHI.pdf

[17] Yunhua Hu, Hang Li, Yunbo Cao, Dmitriy Meyerzon, and Qinghua Zheng. 2005. Automatic Extraction of Titles from General Documents Using Machine Learning. In *Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL '05)*. ACM, New York, NY, USA, 145–154. https://doi.org/10.1145/1065385.1065418

[18] Matthew Hurst and Tetsuya Nasukawa. 2000. Layout and Language: Integrating Spatial and Linguistic Knowledge for Layout Understanding Tasks. In *Proceedings of the 18th Conference on Computational Linguistics - Volume 1 (COLING '00)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 334–338. https://doi.org/10.3115/990820.990869

[19] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transformation Scripts *(CHI '11)*. 3363–3372.

[20] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. 2002. A Brief Survey of Web Data Extraction Tools. *SIGMOD Rec.* 31, 2 (June 2002), 84–93. https://doi.org/10.1145/565117.565137

[21] Vu Le and Sumit Gulwani. 2014. FlashExtract: A Framework for Data Extraction by Examples *(PLDI '14)*. 542–553.

[22] Kristina Lerman, Craig Knoblock, and Steven Minton. 2001. Automatic Data Extraction from Lists and Tables in Web Sources. In *In Proceedings of the workshop on Advances in Text Extraction and Mining (IJCAI-2001), Menlo Park*. AAAI Press.

[23] Chun Chen Lin, Y. Niwa, and S. Narita. 1997. Logical structure analysis of book document images using contents information. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, Vol. 2. 1048–1054 vol.2. https://doi.org/10.1109/ICDAR.1997.620669

[24] Ion Muslea et al. 1999. Extraction patterns for information extraction tasks: A survey. In *The AAAI-99 workshop on machine learning for information extraction*, Vol. 2. Orlando Florida.

[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] David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. 2003. Table Extraction Using Conditional Random Fields. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval (SIGIR '03)*. ACM, New York, NY, USA, 235–242. https://doi.org/10.1145/860435.860479

[27] Roya Rastan, Hye-Young Paik, and John Shepherd. 2015. TEXUS: A Task-based Approach for Table Extraction and Understanding. In *Proceedings of the 2015 ACM Symposium on Document Engineering (DocEng '15)*. ACM, New York, NY, USA, 25–34. https://doi.org/10.1145/2682571.2797069

[28] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid Training Data Creation with Weak Supervision. *arXiv preprint arXiv:1711.10160* (2017).

[29] Sen Wu, Luke Hsiao, Xiao Cheng, Braden Hancock, Theodoros Rekatsinas, Philip Levis, and Christopher Ré. 2017. Fonduer: Knowledge Base Construction from Richly Formatted Data. *arXiv preprint arXiv:1703.05028* (2017).

[30] Mao wusong, Azriel Rosenfeld, and Tapas Kanungo. 2003. Document structure analysis algorithms: A literature survey. *Proceedings of SPIE - The International Society for Optical Engineering* 5010, 197–207. https://doi.org/10.1117/12.476326

APPENDIX

5.1 Related Works: Specific Structure Identification Techniques

Texture aims to provide tools for extracting a variety of structural elements, all within a single framework. There are works that extract a variety of structural elements[9, 10, 12, 13, 17, 18, 22, 27], but some systems focus on documents from particular domains[2, 23], e.g. books that have a contents table for forming the logical structure and that have pages with consistent structure throughout[23].

Existing and specialized methods for extracting specific structures vary in methodology from machine learning (ML) [15, 26] to rule-based methods and may even use hybrid methods. In table extraction, some works use conditional random fields (CRFs) [26], while others use natural language processing (NLP) techniques [13]. Texture encourages the use of any heuristic.

In figure extraction, the focus is on distinguishing texts from images from text of other structures. PDFFigures [7] extracts figures from computer science papers, and can identify text-heavy figures and distinguish between mathematical symbols and the article's main text. PDFFigures depends on caption identification in order to identify figures. Texture supports the order of execution of heuristics over differing structures.

Many other works use structure identification in order to achieve other goals. Futrelle et al. [11] identifies vector-based diagrams in order to classify bar graphs and non-bar graphs. Texus [27] is a table extraction and table understanding system and can map the table's data values to its labeling cells. Hoffswell and Liu [16] identify tables to restructure them for mobile phone display.

Structure	ID	Description
Title	T1	Titles have the largest document font size and are less than 40 characters.
	T2	Titles have the largest document font size and are located in the first page.
	T3	Titles have the largest document font size and mostly contain capitalized words.
	T4	Titles have font sizes larger than the average font size and only contain alphanumeric characters or spaces.
Paragraph	P1	Paragraph have the most frequent font size. All lines with this font size are grouped if they are within a certain threshold distance, which is calculated by the most common distance between text lines in the document.
	P2	Similar to P1, except the paragraphs' font size is not necessarily unique throughout the document.
	P3	Paragraphs have more than 80% of its content in its normal case. It uses the case-based TrueCaseAnnotator in the Stanford CoreNLP library.
List	L1	Lists are paragraphs with alignments deeper to the right of the most common left alignment of paragraphs in the document.
	L2	Lists are paragraphs that start with a number or a special character.
	L3	Lists are sets of lines that begin with the same word and share the same left alignment.
	L4	Nested lists are previously identified lists that have deeper alignments from a nearby parent list.
	L5	Lists are identified by joining previously identified lists close to each other and if there is a repetition of words or a consecutive enumeration.
	L6	Lists are lines close to each other (similar to L3) and have more than 50% of the lines starting with a special character or number.
Header/Footer	HF1	Headers and footers have font sizes less than or equal to the most common font size in the document, and are in the top (header) or bottom (footer).
	HF2	Headers and footers have font sizes slightly larger than the main font size, but not the largest one.
	HF3	Headers and footers have similar text content throughout the document, which is measured using the Jaccard similarity measure.
	HF4	Headers (or footers) are top lines (or bottom lines) followed (or preceded) by a vertical whitespace that is taller than usual.
	HF5	Similar to HF4, but it does not analyze the first page of a document, which could have a different format.
	HF6	Headers (or footers) are the top (or bottom) lines of a page (or footer) that contains numbers.
Figure	F1	Figures are raster images.
	F2	Figures are regions with vector images, which are grouped paths close to each other.
	F3	Figures labeled by using F2 and F1 combined.
	F4	Figures are labeled by PDFFigures 2 [8].
	F5	Figures identified from F4 are removed regions if labeled as tables from other heuristics. This is because PDFFigures consider tables as figures.
Caption	C1	Captions contains specific keywords (e.g. "Table" or "Fig").
	C2	Captions are lines of text closest to a figure.
	C3	Captions are identified by PDFFigures 2 [8].
Table	TB1	Tables have horizontal paths but no vertical ruling line.
	TB2	Tables are regions where word density is at least 80% of the page's average word density.
	TB3	Combination of TB1 and TB2.

Table 2: Brief Description of Heuristics