Populating Ontologies with Data from Lists in Family History Books

Thomas L. Packer
Brigham Young University
Provo, Utah, USA
Email: tpacker@byu.net

David W. Embley
Brigham Young University
Provo, Utah, USA
Email: embley@cs.byu.edu

***Abstract*—A flexible, accurate, and cost-effective method of automatically extracting facts from lists in OCRed documents and inserting them into an ontology would help make those facts machine searchable, queryable, and linkable and expose their rich ontological interrelationships. To work well, such a process must be adaptable to variations in list format, tolerant of OCR errors, and careful in its selection of human guidance. We propose a wrapper-induction solution for information extraction that is specialized for lists in OCRed documents. In this approach, we induce a regular-expression grammar that can infer list structure and field labels from OCR text. We decrease the cost and improve the accuracy of this induction process using semi-supervised machine learning and active learning, allowing induction of a wrapper from a single hand-labeled instance per field per list. To further reduce cost, we use the wrappers learned from the semi-supervised process to bootstrap an automatic (self-supervised) wrapper induction process for additional lists in the same domain. In both induction scenarios, we automatically map labeled text to a rich variety of ontologically structured facts. We evaluate our implementation in terms of annotation cost and extraction quality for lists in family history books.**

1555. Elias Mather, b. 1750, d. 1788, son of Deborah Ely and Richard Mather; m. 1771, Lucinda Lee, who was b. 1752, dau. of Abner Lee and Elizabeth Lee. Their children:—

1. Andrew, b. 1772.
2. Clarissa, b. 1774.
3. Elias, b. 1776.
4. William Lee, b. 1779, d. 1802.
5. Sylvester, b. 1782.
6. Nathaniel Griswold, b. 1784, d. 1785.
7. Charles, b. 1787.

1556. Deborah Mather, b. 1752, d. 1826, dau. of Deborah Ely and Richard Mather; m. 1771, Ezra Lee, who was b. 1749 and d. 1821, son of Abner Lee and Elizabeth Lee. Their children:—

1. Samuel Holden Parsons, b. 1772, d. 1870, m. Elizabeth Sullivan.
2. Elizabeth, b. 1774, d. 1851, m. 1801 Edward Hill.
3. Lucia, b. 1777, d. 1778.
4. Lucia Mather, b. 1779, d. 1870, m. John Marvin.
5. Polly, b. 1782.
6. Phebe, b. 1783, d. 1805.
7. William Richard Henry, b. 1787, d. 1796.
8. Margaret Stoutenburgh, b. 1794.

Fig. 1. Lists in *The Ely Ancestry*, Page 154

I. Introduction

Family history books and other machine-printed documents present much of their valuable content in data-rich lists. The 50,000+ family history books held by FamilySearch.org are full of lists containing hundreds of millions of fact assertions about people, places, and events. Figure 1 shows examples of lists found on Page 154 of *The Ely Ancestry* [2]. These lists make many assertions about family relationships and life events. Our goal is to develop a means to extract the diverse kinds of facts from lists in OCRed documents that is robust to OCR errors and relies on as little human effort as possible.

To be most useful to downstream search, query, and data-linking applications, the knowledge extracted from text must be expressive and well structured. Ontologies are machine-readable, mathematically specified conceptualizations of a collection of facts. They are expressive enough to provide a framework for storing more of the kinds of assertions found in lists than the typical output of named entity recognition and most other information extraction work. If we could populate user-specified ontologies with predicates representing the facts in OCRed lists, this more expressive and versatile information could better contribute to a number of applications in historical research, database querying, record linkage, automatic construction of family trees, and question answering.

In this paper we propose ListReader, a robust, general, and cost-effective solution to the challenge of extracting diverse types of facts from lists in OCRed documents. ListReader populates a user-defined ontology with assertions found and labeled automatically. A ListReader user constructs an ontology for a list by building a data-entry form in a custom web interface and fills in the form with the information from the first record of a list. ListReader induces a regular-expression wrapper and automatically generalizes it to extract asserted information from the remaining records of the list. Only when ListReader encounters a new field in a later record must it ask the user to update the form to accommodate the new field and insert the field value to provide additional training data. This is the minimum amount of training data conceivable as the user begins to train ListReader to recognize information in a new domain and document type. After ListReader has begun inducing grammars and extracting information from a document, it can switch into a self-supervised mode in which it uses its store of knowledge to effectively label its own training data for other lists, potentially removing the human user from the process.

Other information extraction papers target lists [8], [9], [11], but very few target OCRed lists. Those that target non-OCRed lists target HTML lists and generally rely on consistent landmarks (e.g. HTML tags) that are not available in OCR text. Furthermore, and perhaps more importantly relative to our work, they do not target the diverse semantic distinctions in the rich ontological structures that we do. Most information-

extraction work targeting OCRed lists is specific to certain kind of lists. Belaïd [3] [4] and Besagni, et al. [5] [6] extract records and fields from lists of citations, but rely primarily on hand-crafted knowledge that is specific to bibliographies. A paper by Adelberg [1] and one by Heidorn and Wei [10] target lists in OCRed documents in a general sense. They, however, use supervised wrapper induction that we believe will not scale as well as our semi-supervised or self-supervised approaches when encountering the "long tail" of list formats. Also, the extracted information is limited in ontological expressiveness.

In this paper, we make the following contributions. (1) We establish a formal correspondence among list wrappers, ontologies, data-entry forms, and in-line annotated text. This correspondence provides the data flow for a processes in which a user can easily annotate OCRed text as training data for wrapper induction and create a new ontology schema. It also enables even simple induced wrappers that produce in-line or sequentially labeled text to extract rich facts from lists and insert them into an expressive ontological structure (Section II-A). This effectively reduces the ontology population problem to a sequential labeling problem. (2) We demonstrate that it is possible to perform wrapper induction for a list using only one human-provided label per field (Section II-B). (3) We show one way that automatic labeling can replace a human labeler in providing input to wrapper induction by using wrappers previously induced from other lists (Section II-C). (4) We evaluate extraction accuracy and show that ListReader outperforms a general, state-of-the-art information extraction system with high statistical significance (Section III). (5) We conclude that we can benefit from the ListReader line of research and identify opportunities for future research into cheaply inducing accurate wrappers for general OCRed lists (Section IV).

II. List Wrapper Induction

A. ListReader Overview

ListReader populates an ontology from lists in OCRed text as follows:

First, a user selects an OCRed image (a two-layer PDF file in our implementation) that contains a list (e.g. Figure 1). Initially, when ListReader has no information in its knowledge repository allowing it to find and process lists on its own, a user spots a list and, with ListReader's form interface, constructs a form for the data fields in the first record of the list and fills in the form with text from its first record. For example, supposing the spotted list is the second child list in Figure 1, the user would construct the form in Figure 2 and fill it in by clicking on the words in the PDF in the order specified by the form.

Second, from the empty form, ListReader creates the schema of an ontology (e.g. Figure 3).

Third, to induce a wrapper, ListReader uses the information obtained from the filled-in form to label the fields within the OCRed text as training data. Figure 4 shows the labeled text for our example. ListReader labels text strings within the OCRed document with path expressions. Each expression identifies a path in the ontology hyper-graph from the root node to a leaf (text) node which ListReader can use to map labeled

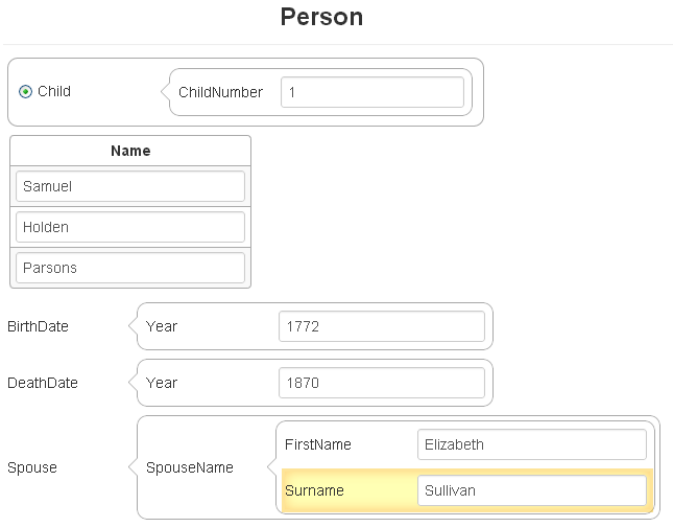

Fig. 2. Filled in Form for Samuel Holden Parsons Record

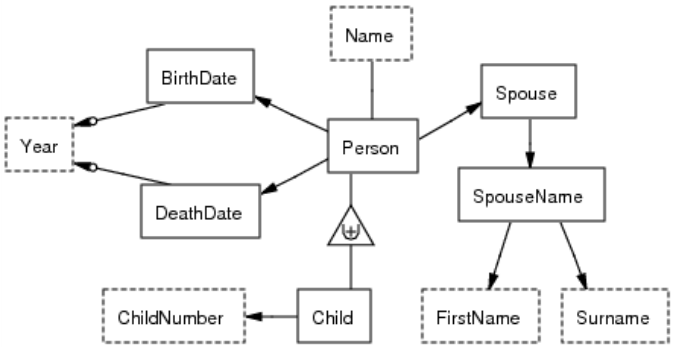

Fig. 3. List Ontology for Samuel Holden Parsons List

text to object and relationship predicates. The in-line labeled text provides enough information for ListReader to induce an information extraction wrapper for the whole list as we explain in Section II-B.

Fourth, the induced wrapper labels the remaining records in the list with labels like those provided in its training data.

Finally, ListReader saves the induced wrapper and ontology in its knowledge repository and uses it to find and process similar lists in other OCRed document images. In this case, the user needs neither to create a form to generate the ontology nor to fill in the form to label any of the fields of any of the records. We explain how this works in Section II-C.

```
<ChildNumber>1</ChildNumber>. <Name>Samuel</Name>
 <Name>Holden</Name> <Name>Parsons</Name>
, b. <BirthDate.Year>1772</BirthDate.Year>
, d. <DeathDate.Year>1870</DeathDate.Year>
, m. <FirstName>Elizabeth</FirstName>
 <Surname>Sullivan</Surname>.
```

Fig. 4. Labeled Samuel Holden Parsons Record

The metaphor of form fill-in for obtaining information is familiar to most users as is form creation from the set of primitives we provide. These form primitives, along with nesting, provide for a rich set of ontological structures. Each named form primitive corresponds to an object set, and each nesting

Label	Initial Regex	Final Regex RecordType1	Final Regex RecordType2
RecordDelimiter	(\n)	(\n)	(\n)
ChildNumber	(\d)	(\d)	(\d)
FieldDelimiter	(\.\s)	(\.\s)	(\.\s)
Name	(\w{6,6})	(\w{5,9})	(\w{5,9})
FieldDelimiter			(\s)
Name			(\w{3,8})
FieldDelimiter	(,\sb\.\s)	(,\sb\.\s)	(,\s[bh]\.\s)
BirthDate.Year	(\d{4,4})	(\d{4,4})	([i0-9]{4,4})
FieldDelimiter			([.,]\sd\.\s)
DeathDate.Year			(\d{4,4})
FieldDelimiter	(\.)	(\.)	(\.)
RecordDelimiter	(\n)	(\n)	(\n)

Fig. 5. Regex Induction for First Child List in Fig. 1

corresponds either to a relationship set or to a role specialization. ListReader can extract rich data with five types of expressiveness: (1) textual vs. abstract entities (e.g. *Name*("Elias") vs. *Person*(p_1)), (2) n-ary relationships among two or more entities instead of strictly unary and binary relationships (e.g. Husband-married-Wife-in-Year(p_1, p_2, "1771")), (3) ontology hyper-graphs with arbitrary path lengths from the root instead of just unit length as in named entity recognition or data slot filling (e.g. <Person.Spouse.SpouseName.Surname>), (4) functional and optional constraints on relationship sets (e.g. A person has one birth event vs. zero or more marriage events), (5) generalization-specialization hierarchies, including, in particular, role designations (e.g. *Child isa Person*).

B. Semi-supervised Wrapper Induction

In the semi-supervised wrapper induction setting, ListReader begins learning from nothing more than the text of an OCRed page image with the fields of the first record of a list labeled. ListReader initializes a new wrapper to model the text and labels of the first record using a sequence of capture groups corresponding to the sequence of fields and delimiters. Consider, for example, the following labeling of the first record of the first child list in Figure 1:

```
<ChildNumber>1</ChildNumber>. <Name>Andrew</Name>,
b. <BirthDate.Year>1772</BirthDate.Year>
```

From this labeling, ListReader generates the initial regular expression (regex) in Figure 5, a first level generalization of the field delimiters and content.

ListReader generalizes the initial wrapper in four steps to produce a set of regexes, one for each record type. First, ListReader performs an A* graph search for each line of text below the first record. This is a search over a hypothesis space whose nodes are regexes and whose edges are edit operations transforming one regex into another. The goal of the search is the regex with the shortest edit distance from the initial regex that completely matches the new unlabeled text line. Using an admissible heuristic to estimate the remaining distance to the goal from each intermediate regex, the A* search is guaranteed to find our desired regex before any other regex that completely matches the text. The heuristic we use is the number of capture groups in the intermediate regex that do not match any substring of the unlabeled text.

To generate one regex from another, ListReader applies one of four operators to select capture group positions: insertion, deletion, character class substitution, and length substitution. The substitutions always result in a regex that will match a larger class of text. For example, the character class operator replaces each character in a delimiter with a predefined set of common OCR error substitutions (e.g. it replaces "[.]" with "[.,]"). ListReader assigns a field label of "Unknown" to inserted capture groups, used below during active learning.

Second, ListReader scores and ranks the new regexes. ListReader assigns a quality score to each regex which is the product of *similarity* and *match-frequency* (all values are between 0.0 and 1.0). *Similarity* is 1 minus the normalized edit distance from the initial regex. (Values less than 0.0 are set to 0.0.) We compute edit distance by summing an empirically determined cost value (between 1.0 and 2.0) assigned to each operator. *Match frequency* is the number of candidate records matched in unlabeled text divided by an empirically determined maximum number of records that a single regex is expected to match. (Values above 1.0 are set to 1.0.)

Third, ListReader determines whether it should query the user. ListReader executes candidate regexes against the unlabeled text in the order of their quality scores and removes segments of text (records) that match. If ListReader encounters a regex with an "Unknown" label while removing records, it alerts the user by highlighting the text matched by the "Unknown" capture group. The user may then modify the form, which provides a new field label and updates the ontology, and then copy the part of the highlighted text that constitutes the field value into the new form field. ListReader then regenerates that section of the matching regex.

Last, ListReader stores the induced wrapper in its knowledge repository for use in self-supervised wrapper induction (Section II-C). After inducing a wrapper and labeling the records of a list, ListReader uses the labels to instantiate the ontology for the list. For the labeled field values in a record, ListReader creates objects and relationships corresponding to each label's path and properly links the field values within the record according to the structure of the ontology.

C. Self-supervised Wrapper Induction

In the self-supervised setting, ListReader begins wrapper induction with a completely unlabeled page and a non-empty repository of induced wrappers. ListReader effectively labels its own training data by applying the "best" of the stored wrappers to the page. Here, "best" means the regex with the highest quality score as computed against the new text, drawing only from regexes containing no "Unknown" capture groups. In this case, we recompute the match frequency and the heuristic distance for each regex using the new text, adding the latter to the actual edit distance from the original (fully labeled) regex. ListReader then executes the semi-supervised wrapper induction process (Section II-B) for each line of text above and below the best matching record and saves the resulting wrappers in its repository for immediate use.

III. Experimental Evaluation

A main objective of developing ListReader is to find a way to reduce cost (human labeling) and increase accuracy (F-measure) of inducing wrappers for lists by taking advantage of list structure. Since we can view part of ListReader as a

TABLE I. SEMI-SUPERVISED RESULTS

	Accuracy			Cost
	P.	R.	F.	# Labels / List
ListReader	**95%**	**91%**	**93%**	4.5
CRF 1^{st} Rec.	86%	71%	78%	**4.4**
CRF Best Rec.	86%	70%	77%	4.7
CRF Best 2	87%	82%	84%	8.4
CRF Best 3	87%	86%	87%	11.0

TABLE II. SELF-SUPERVISED RESULTS

	Accuracy			Cost
	P.	R.	F.	# Labels / List
ListReader	**91%**	**75%**	**82%**	**0.24**
CRF	72%	68%	69%	0.66

machine-learned sequential labeler, we empirically compare it to a highly regarded Conditional Random Field (CRF) statistical sequence labeler [12]. To make our labeling task learnable by the CRF and to ensure a fair test, we tuned its hyper-parameters and selected an appropriate set of word features. As test data, we selected and isolated the text of 30 child lists from throughout *The Ely Ancestry* [2] containing a total of 137 records and an average of 10 fields per list. We compute F-measures for field labels over all word tokens not used as hand-labeled training data. All reported differences between CRF and ListReader F-measures are statistically significant at $p < 0.01$ using both McNemar's test [7] and a paired t-test.

To test ListReader's semi-supervised wrapper induction, we hand-labeled the first record of each list and ran ListReader separately on it. We compare the results of ListReader to the CRF, also run separately on each list with varying amounts of training data. Table I shows the results. Hand-labeling just the first record has a lower cost for the CRF compared to ListReader (4.4 v. 4.5 labels per list), but the F-measure is lower. Not until trained with the "three best" records does the F-measure of the CRF approach that of ListReader. Even then it is still significantly less and at over double the number of labels plus the effort to select the "three best"—a combination of a longest (1^{st} Best), a least typical (2^{nd} Best), and a most typical (3^{rd} Best) record.

To evaluate self-supervised learning, we ran ListReader on one of the lists in semi-supervised mode and then executed it in self-supervised mode on each of the remaining 29 lists. We were thus able to see how well ListReader could use a wrapper generated for one list to begin wrapper induction for another list using no more human labeling than for new unique fields not identified in the first list. For the CRF, we hand-labeled all records in one list to train it and then executed it on each of the remaining 29 lists. For both ListReader and the CRF, we repeated the procedure 30 times, using each list in our test set as a starting list, to compute the averages in Table II. ListReader achieves a higher F-measure at almost a third the cost of the CRF.

IV. CONCLUSIONS AND FUTURE WORK

These encouraging early results suggest that ListReader is a viable new line of research for solving the general problem of populating ontologies with data from lists in OCRed documents. ListReader outperforms a CRF ($p < 0.01$) in all our tests showing that we can better leverage the characteristics of list structure with a more tailored machine learning approach. ListReader has the potential to reduce human effort, not only limiting user involvement to labeling each distinct field of a list only once, but for an entire collection of lists, like the child lists in *The Ely Ancestry*. Additionally, ListReader uniquely obtains rich ontological assertions that are more useful than simply labeling words with named entity categories.

To continue this line of research, we will investigate using a Hidden Markov Model instead of regular expressions as the wrapper formalism to improve robustness and speed. Our goal is to increase both precision and recall and to simultaneously decrease cost during self-supervised wrapper induction. We also intend to work with more complex lists and ontologies: (1) lists split by intervening text or page breaks (e.g. the lists in *The Ely Ancestry* that split across page boundaries), (2) lists nested within other lists (e.g. the child lists nested within the larger parent list in Figure 1), (3) lists with fields factored out of each record, (e.g. the surname of the children in a family factored out of the child lists in Figure 1), (4) lists whose records describe entities from distinct categories (e.g. child lists containing records with distinct structures for sons and daughters), and (5) lists that can be modeled by joining fragments of previously learned wrappers and ontologies (e.g. parish christening records learned from joining parts of family lists with parts of church administration lists).

REFERENCES

[1] B. Adelberg. NoDoSE — a tool for semi-automatically extracting structured and semistructured data from text documents. *ACM SIGMOD Record*, 27:283–294, 1998.

[2] M. S. Beach, W. Ely, and G. B. Vanderpoel. *The Ely Ancestry*. The Calumet Press, New York, New York, USA, 1902.

[3] A. Belaïd. Retrospective document conversion: application to the library domain. *International Journal on Document Analysis and Recognition*, 1:125–146, 1998.

[4] A. Belaïd. Recognition of table of contents for electronic library consulting. *International Journal on Document Analysis and Recognition*, 4:35–45, 2001.

[5] D. Besagni and A. Belaïd. Citation recognition for scientific publications in digital libraries. In *Proceedings of the First International Workshop on Document Image Analysis for Libraries*, pages 244–252, Palo Alto, California, USA, 2004.

[6] D. Besagni, A. Belaïd, and N. Benet. A segmentation method for bibliographic references by contextual tagging of fields. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, pages 384–388, Edinburgh, Scotland, 2003.

[7] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, Oct. 1998.

[8] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2:1078–1089, 2009.

[9] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. *Proceedings of the VLDB Endowment*, 2:289–300, 2009.

[10] P. B. Heidorn and Q. Wei. Automatic metadata extraction from museum specimen labels. In *Proceedings of the 2008 International Conference on Dublin Core and Metadata Applications*, pages 57–68, Berlin, Germany, 2008.

[11] K. Lerman, C. Knoblock, and S. Minton. Automatic data extraction from lists and tables in web sources. In *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, volume 98, 2001.

[12] A. K. McCallum. MALLET: a machine learning for language toolkit. http://mallet.cs.umass.edu/, 2002.