Cost-Effective Information Extraction from Lists in OCRed Historical Documents

Thomas L. Packer
Department of Computer Science
Brigham Young University
Provo, Utah 84602
thomaspacker@gmail.com

David W. Embley
Department of Computer Science
Brigham Young University
Provo, Utah 84602
embley@cs.byu.edu

ABSTRACT

To work well, machine-learning-based approaches to information extraction and ontology population often require a large number of manually selected and annotated examples. In this paper, we propose ListReader which provides a way to train the structure and parameters of a Hidden Markov Model (HMM) without requiring any labeled training data. The induced HMM is a wrapper—a function that hides within it the complexities of low-level processing—in ListReader's case the complexities of information extraction from OCRed historical documents. The HMM wrapper is capable of recognizing lists of records in text documents and associating subsets of identical fields across related record templates. The algorithmic training method we employ is based on a novel unsupervised active grammar-induction framework. The training produces an HMM wrapper and uses an efficient active sampling process to complete the mapping from wrapper to ontology by requesting annotations from a user for automatically-selected examples. We measure performance of the final HMM in terms of F-measure of extracted information and manual annotation cost and show that ListReader learns faster and better than a state-of-the-art baseline and an alternate version of ListReader that induces a regular-expression wrapper.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Language parsing and understanding*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

Keywords

information extraction, wrapper induction, list, unsupervised learning, active learning, grammar induction, OCR, ontology population, HMM, Hidden Markov Model

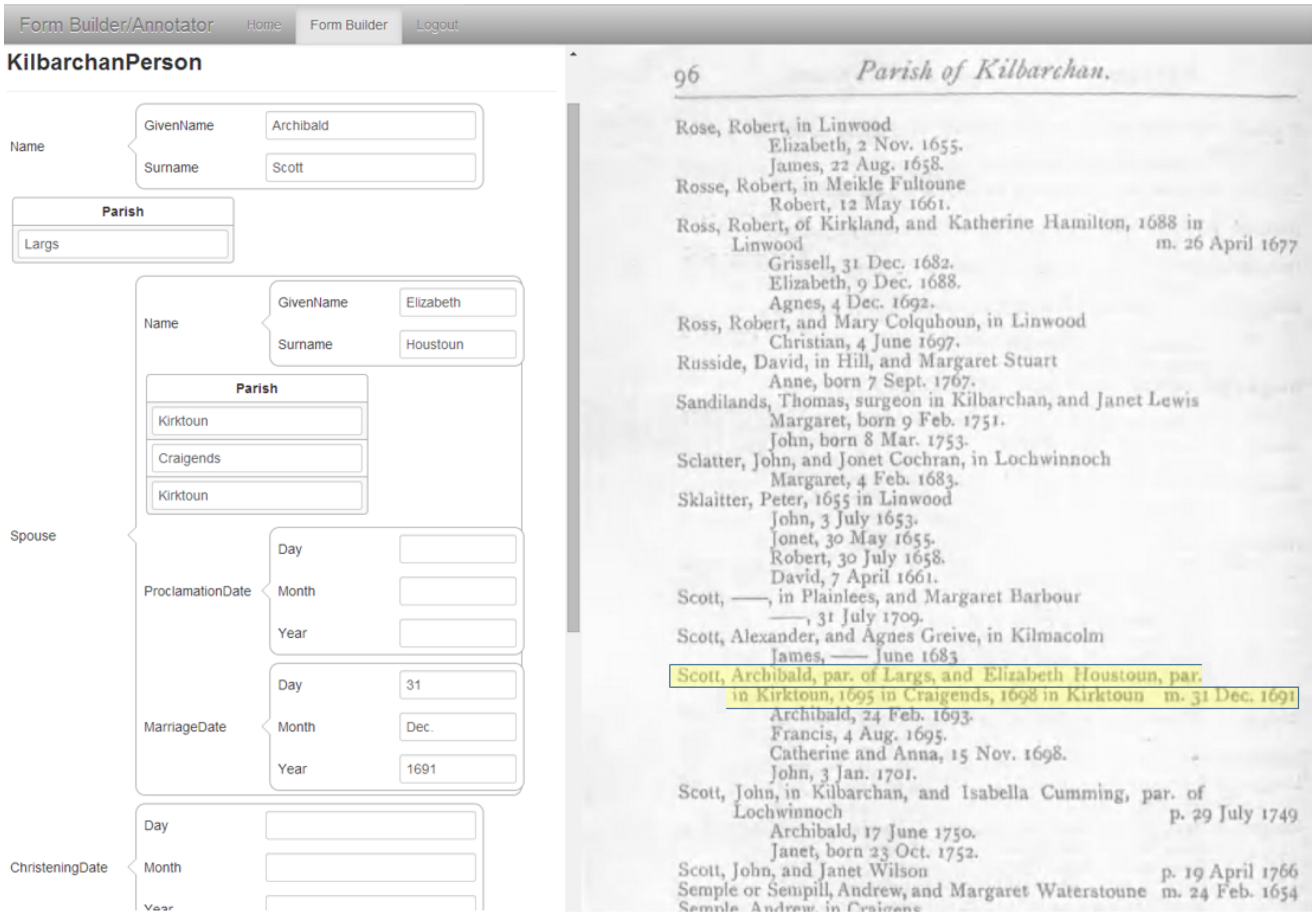
HIP Š15 August 22, 2015, Nancy, France
© 2015 ACM. ISBN 978-1-4503-3602-4/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2809544.2809547

1. INTRODUCTION

FamilySearch [7] has scanned, OCRed, and placed on line more than 150,000 historical documents, mostly family history books, and continues to add to its every-growing collection at the rate of about 25,000 per year. Automatically extracting information from these historical documents is a major challenge. Much of the information in these documents is semi-structured in list-like structures, for which we propose *ListReader*, an unsupervised active wrapper induction tool that learns Hidden Markov Models (HMMs) capable of extracting detailed information from OCRed historical documents and populating richly-structured ontologies.

ListReader processes text documents that comprise an OCRed collection of page images from a scanned book such as the *Kilbarchan Parish Register* [8], a partial page of which appears in the right side of Figure 1. To begin ListReader processing, a user constructs a data entry form for the desired information such as the form on the left side in Figure 1. ListReader translates the form into an ontology schema into which the extracted information will be stored. Then, without anything more than the given text document, ListReader applies an unsupervised process to automatically discover and align records and train the structure and parameters of an HMM. It then actively requests labels for selected strings of text from the user, highlighting the strings it selects for the user to label and allowing the user provides labels by filling in the data entry form. Figure 1 shows the filled-in form for the highlighted text. ListReader uses the structure of the form to generate specialized labels for the field strings in the text document that specify the mapping of the strings to ontology predicates.

The ListReader approach to wrapper induction, which is a combination of unsupervised learning and active learning, is unique in comparison with other wrapper-induction tools. Because we apply ListReader to OCRed historical documents, it cannot rely on consistent, error-free landmarks available in machine-generated HTML pages, assumed by previously developed wrapper-induction systems (e.g. [1, 3, 5, 6, 11, 14, 15]). Indeed, we show in our evaluation of ListReader (Section 5) that an adaptation of likely the best of these wrapper-induction prototypes does not perform as well as ListReader. Furthermore, to keep the cost of labeled training down, we must use unsupervised machine-learning techniques to the extent possible. ListReader takes a markedly different approach from other unsupervised learning systems [6, 9, 11] because it must work with input that is less-consistently formatted and OCR-error-prone and with output that is more richly-structured.



KilbarchanPerson

Name	GivenName	Archibald	
	Surname	Scott	
Parish	Largs		
Spouse	Name	GivenName	Elizabeth
		Surname	Houstoun
	Parish	Kirktoun	
		Craigends	
		Kirktoun	
	ProclamationDate	Day	
		Month	
		Year	
	MarriageDate	Day	31
		Month	Dec.
		Year	1691
ChristeningDate	Day		
	Month		
	Year		

96 *Parish of Kilbarchan.*

Rose, Robert, in Linwood
Elizabeth, 2 Nov. 1655.
James, 22 Aug. 1658.
Rosse, Robert, in Meikle Fultoune
Robert, 12 May 1661.
Ross, Robert, of Kirkland, and Katherine Hamilton, 1688 in Linwood m. 26 April 1677
Grissell, 31 Dec. 1682.
Elizabeth, 9 Dec. 1688.
Agnes, 4 Dec. 1692.
Ross, Robert, and Mary Colquhoun, in Linwood
Christian, 4 June 1697.
Russide, David, in Hill, and Margaret Stuart
Anne, born 7 Sept. 1767.
Sandilands, Thomas, surgeon in Kilbarchan, and Janet Lewis
Margaret, born 9 Feb. 1751.
John, born 8 Mar. 1753.
Sclatter, John, and Jonet Cochran, in Lochwinnoch
Margaret, 4 Feb. 1683.
Sklaitter, Peter, 1655 in Linwood
John, 3 July 1653.
Jonet, 30 May 1655.
Robert, 30 July 1658.
David, 7 April 1661.
Scott, ——, in Plainlees, and Margaret Barbour
——, 31 July 1709.
Scott, Alexander, and Agnes Greive, in Kilmacolm
James, —— June 1683
Scott, Archibald, par. of Largs, and Elizabeth Houstoun, par. in Kirktoun, 1695 in Craigends, 1698 in Kirktoun m. 31 Dec. 1691
Archibald, 24 Feb. 1693.
Francis, 4 Aug. 1695.
Catherine and Anna, 15 Nov. 1698.
John, 3 Jan. 1701.
Scott, John, in Kilbarchan, and Isabella Cumming, par. of Lochwinnoch p. 29 July 1749
Archibald, 17 June 1750.
Janet, born 23 Oct. 1752.
Scott, John, and Janet Wilson p. 19 April 1766
Semple or Sempill, Andrew, and Margaret Waterstoune m. 24 Feb. 1654
Semple, Andrew, in Craigens

Figure 1: *Kilbarchan Parish Register* Page and KilbarchanPerson Filled-in Form

Our ListReader research makes the following contributions. (1) We give an algorithm to train both the model structure and the parameters of an HMM for list information extraction that requires no hand-labeled examples (Sections 2 and 3). This algorithm is linear in time and space with respect to the input text length, the discovered pattern length, and output label alphabet. (2) We define an efficient active sampling process to complete the HMM as a data-extraction wrapper that can map the data in lists to an expressive ontology schema (Section 4). Active sampling is an active-learning-like process that requests labels of selected examples from the user without modifying the internal wrapper structure. (3) In an experimental evaluation, we show that a ListReader-induced wrapper outperforms two alternatives in terms of a metric that combines precision, recall, and annotation cost (Section 5).

2. PATTERN DISCOVERY

ListReader begins its discovery of patterns by abstracting text to tokenize and chunk the text which in turn improves tolerance of many common OCR errors and the natural variations among fields of the same type. Text abstraction rules consist of (1) joining tokens split by end-of-line hyphens; (2) abstracting dashes of varying lengths to single-length dashes; (3) replacing digits with a digit designator (“`Dg`”); (4) replacing white space with a newline (“`\n`”) or space symbol (“`[Sp]`”); (5) removing spaces that occur on the “wrong side” of certain punctuation characters because of an OCR or typesetting error, such as immediately before a period; (6) abstracting space/newline-delimited letter sequences of capitalized words (“`[UpLo]`”), upper-case words (“`[Up]`”), camel-case words (“`[LoUp]`”), and lower-case words (“`[Lo]`” or, as a user-selected option, not abstracted); and (7) punctuation, which is not abstracted. As an example, the text strings of child records for the Elizabeths, Agnes, a Margaret, and Francis in Figure 1 all generalize to:

```
[\n][UpLo],[Sp][Dg][Sp][UpLo].[Sp][DgDgDgDg].[\n]
```

To find record-like patterns, ListReader builds a suffix tree from the abstracted text and searches it for repeated patterns that satisfy our record-selection constraints: *records must begin and end with a valid record delimiter, must occur at least twice, and must contain at least one numeral or capitalized word.* The abstracted text is the single long string of symbols from the first abstract symbol for the first token on the first page of the book to the last abstract symbol on the last page. Each edge in the suffix tree is labeled by the substring of symbols it represents (e.g. the sequence of abstract symbols above for Elizabeth and James). ListReader constructs a suffix tree in linear time and space using Ukkonen’s algorithm [21]. It also finds and collects record-like patterns within the suffix tree in linear time and space by iterating over the edges of the suffix tree and checking for strings terminating in each edge that adhere to the properties specified for a record. In Figure 1 ListReader finds, for example, that the child record pattern for Elizabeth and James also pertains to Grissel, the second Elizabeth, and Agnes, but not the first Robert, since his christening date "May" is not abbreviated and thus is not followed by a period.

ListReader next discovers parts of records—*field groups*—that recur among different record clusters. These correspondences will be represented later in the HMM and used to reduce the number of necessary hand-labeled fields. The intuition is that a field value like a birth year or marriage year that follows a specific field group delimiter like "`born`" or "`m.`" can be identified and labeled the same even when found in different record clusters. In the Kilbarchan parish record, "`born`" designates birth dates like those for Margaret and John Sandilands in Figure 1, rather than christening dates, which are unmarked, and "`m.`" denotes marriage dates, like the marriage date for Robert and Katherine in Figure 1.

ListReader constructs *field group templates* from the text appearing between field group delimiters and associates a field group template with the delimiter on its left. Field group templates consist of a field group delimiter, whose text (like "`born`" *is not* abstracted in the template, followed by one or more variations of the field group itself, whose text *is* consists of abstract symbols. For example, the initial delimiter, "`\n`" has the following variations for the text comprising the first three families in Figure 1:

```
\n[UpLo],[Sp][UpLo]
\n[UpLo],[Sp][DgDg][Sp][UpLo].[Sp][DgDgDgDg]
\n[UpLo],[Sp][DgDg][Sp][UpLo][Sp][DgDgDgDg]
\n[UpLo],[Sp][Dg][Sp][UpLo].[Sp][DgDgDgDg]
```

A few more variations appear on the page and still more in the full book. A field group template for a final record delimiter is just the delimiter itself as no field follows it.

At this point ListReader almost has what it needs for HMM creation. With some additional adjustments, ListReader will have identified record and field group templates from which it can directly construct an HMM that will extract the fields in the records. The adjustments include discarding record patterns that do not resolve into a clean sequence of field group templates and grouping record clusters that satisfy the same sequence of field group templates and then splitting some of the individual field group templates into alternate template groups depending on whether there is enough variation to warrant a split. Among many others, ListReader finds the record and field group templates in Figure 2 in the Kilbarchan parish record.

```
[[\n-Segment][\n-End-Segment]]
  [\n-Segment]
    \n[UpLo],[Sp][DgDg][Sp][UpLo][Sp][DgDgDgDg]
      : \nRobert, 12 May 1661
    \n[UpLo],[Sp][UpLo] : \nAllasoun, Richard
    \n[UpLo] : \nLochwinnoch
  [\n-End-Segment]
    .\n : .\n
    \n : \n
[[\n-Segment][born-Segment][\n-End-Segment]]
  [\n-Segment]
    \n[UpLo] : \nJanet
  [born-Segment]
    ,[Sp][born][Sp][DgDg][Sp][UpLo].[Sp][DgDgDgDg]
      : , born 23 Oct. 1752
  [\n-End-Segment]
    .\n : .\n
    \n : \n
```

Figure 2: Record and Field Group Templates (sample text snippets have been added following the colon to aid in readability)

3. HMM CONSTRUCTION

An HMM is a probabilistic finite state machine consisting of a set of hidden states S, a set of possible observations W, an emission model $P(w|s)$ associating a state with a set of observable events, and a transition model $P(s_t|s_{t-1})$ associating one state with the next. States are initially "hidden" and must be inferred during application of the HMM from the observable events in the text. In our work, each event is a word-sized chunk of text (a token), including alphabetic words, numerals, spaces including newlines, and punctuation characters. Inferring the correct state associated with each word token is the main task done in extracting information from the text and is guided by the parameters of the HMM. Using the Viterbi algorithm, ListReader selects the most probable sequence of states given the words of the input text and the HMM's parameters. The emission model is a categorical distribution—a table of conditional probabilities indicating which observation w can be emitted from which hidden state s and with what probability given s. The transition model is also a categorical distribution—a table of conditional probabilities indicating which hidden state s_t at position t can follow which other hidden state s_{t-1} at position $t-1$ and with what probability given s_{t-1}. The two kinds of probabilities are the parameters of the HMM. The set of states and the transitions that have non-zero probabilities in the transition model determine the structure of the state machine of the HMM. The processing described in Section 2 provides what ListReader needs to produce the set of hidden states and both the transition and the emission model for our application.

To construct the HMM, ListReader transforms each field group template into a linear sequence of HMM states, one HMM state for each token in the field group segment. The HMM fragment for the `[born-Segment]` in Figure 2, for example, has ten states, one for each word token in the template. In addition to one state per word token in a field group template, ListReader generates an insertion state between every pair of consecutive word states. These insertion states allow for inconsistent punctuation and noise in pattern delimiters and for random comments that sometimes appear in otherwise structured text. The main transitions among the states form a straight line through the template. Transitions to and from insertion states allow for text-addition deviations from a typical record. Transitions that skip over one or more states are also added; these transitions allow for text-omission anomalies.

ListReader gives every state both a syntactic and a semantic ID. The syntactic ID ensures that each state is unique within the complete HMM. The semantic ID of an HMM state identifies the field group template to which the states of the field group apply and also the position of the token within the field group template. The semantic ID, therefore, represents both a type of field group segment and a token's position within that field group segment and is purposefully *not* unique within an HMM. States that share a semantic ID (and in turn the words they match) should be labeled the same because they have the same relationship with the primary object of their respective records. For example, all "m.

<date>" constructs in the entire *Kilbarchan Perish Record* are marriage dates and should be labeled as such regardless of the record template in which they are found.

ListReader sets the emission and transition parameters using maximum likelihood estimation (MLE). That is, they are set by normalizing the sums of counts of discovered text-phrase patterns. These parameters must allow for flexible alignment of an induced HMM with text containing natural differences from the text on which the HMM is trained, such as word substitutions, insertions, and deletions. Beyond MLE, we also smooth these parameters using pseudo-counts (Dirichlet priors, which we chose after thoughtful consideration) to allow for combinations of events not present in the training data. For example, a word with abstract text "`[DgDgDgDg]`" receives a count of 1.0 for "`[DgDgDgDg]`", a pseudo-count of 0.01 for "`[Dg]`", "`[DgDg]`", "`[DgDgDg]`", and "`[DgDgDgDgDg]`" and a pseudo-count of 0.001 for "`[UpLo]`", "`[LoUp]`", "`[Up]`" and "`[Lo]`". These counts and pseudo-counts are then summed and normalized to obtain the emission probabilities. In general, ListReader's construction of emission models promotes better alignment of similar words, especially words of the same character class (e.g. character and digit sequences like those just listed), despite the small amount of training data provided and despite possible OCR errors and other variations.

Figure 3 shows a schematic diagram of the HMM ListReader builds for the *Kilbarchen Parish Record.* As shown in the figure, ListReader generates page-level states for the beginning (*PageBeginning*) and ending (*PageEnding*) of each page and connects them to states for non-list text (*NonList*) and for list-record text (*RecordDelimiter*), the beginning state for all record-template HMMs. Also shown is how ListReader connects its record-delimiter state to every HMM record template—all 47 of them for our example run of the *Kilbarchen Parish Record.* One of the record-template HMMs is open, showing the interconnections of the HMM fragments for the "`born`" record template in Figure 2. Notice that the field group template "`[\n-End-Segment]`" for the "`born`" record template in Figure 2 has two templates, one for "`.\n`" and one for "`\n`". Whenever a field group template has multiple component templates, ListReader generates parallel HMM fragments, one for each component template. A field group template has an HMM fragment of the form of the field-template HMM for "`[born-Segment]`" described earlier as having ten word-token states with an insertion state between each, and transitions between and around these states. Figure 3 shows this HMM fragment with only its first and last state. The "..." in Figure 3 stands for the remaining eight states of the "`[born-Segment]`" HMM fragment and its nine insertion states, 27 transitions associated with these insertion states, and 36 transitions modeling the possibility of anomalous deletions. Each HMM fragment requires connections to all prior and subsequent HMM fragments as Figure 3 shows, and also connections from the *RecordDelimiter* state to the first HMM fragment for every record template and from the end of the HMM fragments back to the *RecordDelimiter* state. ListReader determines the transition probabilities for the transitions to HMM record templates based on the size of the cluster of records identified in the document for each record template.

The emission model of *PageBeginning* and *PageEnding* are fixed to contain only the special character that ListReader artificially inserts into the text sequence at the beginning and ending of each page to represent page breaks. The emission model of *RecordDelimiter* is fixed to contain the set of allowable record delimiters, which for the *Kilbarchan Parish Record* contains only the newline character. For these fixed emission models, the probability of the allowable character is 1.0 and all other probabilities are 0.0. The emission model of *NonList* state is not fixed. Rather, it is set as the MLE estimate of all word tokens in the input text that were not covered by any candidate records during unsupervised wrapper induction. The emission model for the *NonList* state in Figure 3 lists several of these word tokens and their probabilities based on actual occurrence counts in our run of ListReader on the *Kilbarchan Parish Record.* The emission model for states in the record templates are as described earlier (and are not shown in Figure 3).

4. LABELING AND FINAL EXTRACTION

After ListReader constructs an HMM, it is fully capable of extracting the information in the discovered patterns. It does not know, however, what the information means. To give the information meaning, a user creates an ontology, a conceptualized knowledge structure, and shows ListReader how to match the information with the ontology. Cost-effectiveness—the reduction of time and effort expended by a user—is paramount. Conceptual effort is reduced because ontology creation consists of and only of creating a form such as the form in Figure 1, and labeling consists of and only of filling in the form for ListReader-selected text snippets. User time is reduced because ListReader effectively minimizes the number of labelings required to maximize the amount of information mapped to the ontology via its innovative *active sampling* process.

ListReader's active sampling consists of a cycle of repeated interaction with the user. On each iteration of the active-sampling loop, ListReader selects and highlights text that matches part of the HMM, and the user labels the fields in highlighted text. In the labeling tool we have constructed for ListReader (see Figure 1), labeling consists of merely clicking on a text token in the pdf image of a document page on the right to transfer the underlying OCRed text to the form field in focus on the left. ListReader accepts the labeled text via its web form interface and assigns labels to the corresponding HMM states, which completes that part of the HMM and enables it to become a "wrapper" that extracts information from the text and maps it to the ontology. For example, placing the text token "`1691`" in the marriage-year field in Figure 1 yields the path *KilbarchanPerson.Spouse.MarriageDate.Year*, which provides an immediate mapping of the text token to the internal ontological structure.

ListReader's active sampling aims to minimize the number of required user labelings. (That it is effective is shown in our evaluation in Section 5.) ListReader's active sampling cycle is a modified form of active learning, focusing on the "active sampling" step and performing practically none of the "model update" step [13]. The HMM training ListReader does is fully unsupervised—no HMM structure or parameter learning takes place under the supervision of a user either interactively or in advance. Label renaming is the only change ListReader makes to the HMM during active sampling. In each cycle, ListReader actively selects the text for labeling that maximizes the return for the labeling effort expended. To initialize the active sampling cycle, ListReader applies

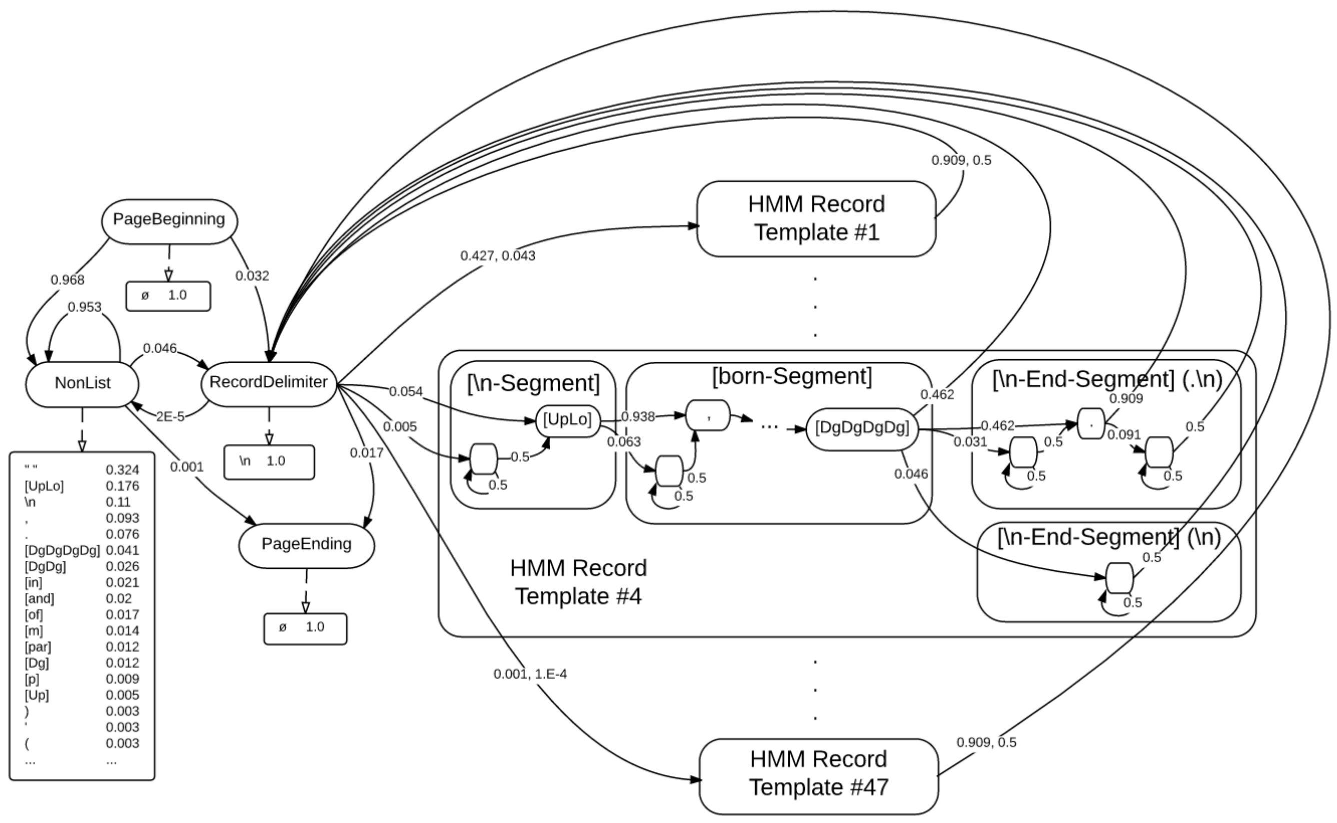


Figure 3: Schematic Diagram of ListReader-generated HMM

the HMM to the text of each page in the book. It labels the strings that match each state with the aforementioned semantic ID assigned during HMM construction. ListReader saves the count of matching strings for each semantic ID. It also records the page and character offsets of the matching strings throughout the book and their associated semantic IDs. ListReader uses the page and character offsets when highlighting a span of text in the user interface for the user to label. ListReader selects a span of text on each iteration of active sampling using a query policy (explained next) that is based on the counts of matching strings for each semantic ID.

The string ListReader selects as "best" is a string that matches the HMM fragment with the highest predicted return on investment (ROI). The ROI can be thought of as the slope of the learning curve: higher accuracy and lower cost produce higher ROI. The HMM fragments considered are HMM record templates or contiguous parts thereof (e.g. the HMM fragment for "`[born-Segment]`" illustrated in Figure 2). When more than one string matches the best HMM fragment, ListReader selects the first one on whichever page contains the most matches of that HMM fragment. ListReader computes the predicted ROI as the sum of the counts of the strings matching each state in the candidate HMM fragment divided by the number of states in the HMM fragment—that is, the average match-count per state. Querying the user to maximize the immediate ROI tends to maximize the slope of the learning curve and has proven effective in other active learning situations [12]. Once the user labels the selected text, ListReader removes the counts for all strings that match the corresponding states or that share the semantic IDs of labeled states, recomputes the ROI scores of remaining states, and issues another query to the user.

In our example run of the *Kilbarchan Parish Record*, ListReader selects the highlighted text in Figure 4. Its HMM record template is composed of the first pattern for the first "`[\n-Segment]`" field group template and the first pattern for the first "`[\n-End-Segment]`" field group template in Figure 2. There are nine matching states in this HMM record template, one for each word-level, non-record-delimiter symbol, "`[UpLo],[Sp][DgDg][Sp][UpLo][Sp][DgDgDgDg].`".The hit count for the strings matching each state are:

`[UpLo]`	2680
`,`	2678
`[Sp]`	2691
`[DgDg]`	2680
`[Sp]`	2678
`[UpLo]`	2679
`[Sp]`	2682
`[DgDgDgDg]`	2683
`.`	3840

whose sum is 25,291 and whose ROI score is thus 25291/9 and is greater than the ROI score for any other HMM record template. Intuitively, this makes sense because the most often occurring fact assertion in the *Kilbarchan Parish Record* is statement of the form "<GivenName>, <Day> <Month> <Year>", which documents the christening of a child.

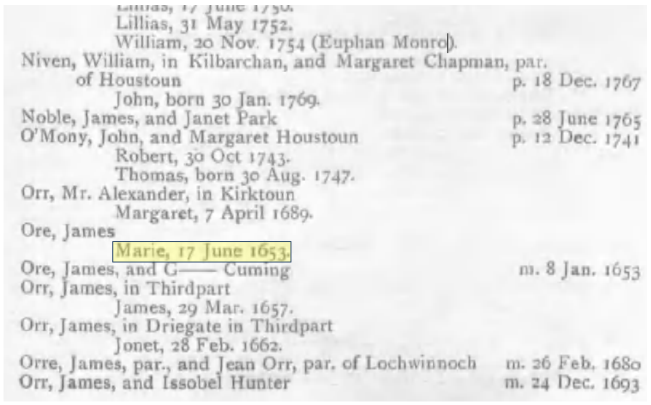

Lillias, 17 June 1750.
Lillias, 31 May 1752.
William, 20 Nov. 1754 (Euphan Monro).
Niven, William, in Kilbarchan, and Margaret Chapman, par. of Houstoun p. 18 Dec. 1767
John, born 30 Jan. 1769.
Noble, James, and Janet Park p. 28 June 1765
O'Mony, John, and Margaret Houstoun p. 12 Dec. 1741
Robert, 30 Oct 1743.
Thomas, born 30 Aug. 1747.
Orr, Mr. Alexander, in Kirktoun
Margaret, 7 April 1689.
Ore, James
Marie, 17 June 1653.
Ore, James, and G—— Cuming m. 8 Jan. 1653
Orr, James, in Thirdpart
James, 29 Mar. 1657.
Orr, James, in Driegate in Thirdpart
Jonet, 28 Feb. 1662.
Orre, James, par., and Jean Orr, par. of Lochwinnoch m. 26 Feb. 1680
Orr, James, and Issobel Hunter m. 24 Dec. 1693

Figure 4: First Active-Sampling User Query

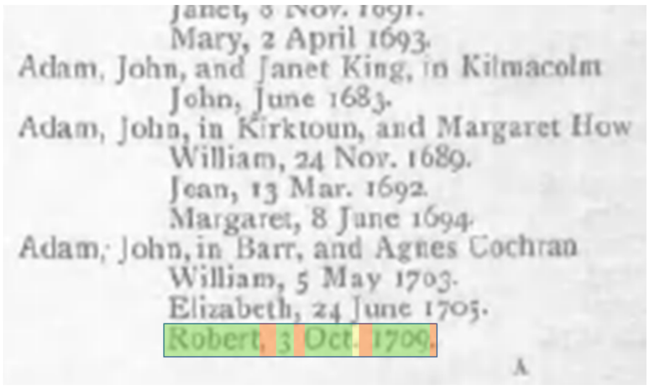

Janet, 8 Nov. 1691.
Mary, 2 April 1693.
Adam, John, and Janet King, in Kilmacolm
John, June 1683.
Adam, John, in Kirktoun, and Margaret How
William, 24 Nov. 1689.
Jean, 13 Mar. 1692.
Margaret, 8 June 1694.
Adam, John, in Barr, and Agnes Cochran
William, 5 May 1703.
Elizabeth, 24 June 1705.
Robert, 3 Oct. 1709.
A

Figure 5: First Active-Sampling User Query Requiring Only Partial Labeling

When one HMM state receives a user-supplied label, all states sharing the same semantic ID receive the same final label. In the example in Figure 4 the user would label "Marie" as *KilbarchanPerson.Name.GivenName*, "17" as *KilbarchanPerson.ChristeningDate.Day*, "June" as *KilbarchanPerson.ChristeningDate.Month*, and "1653" as *KilbarchanPerson.ChristeningDate.Year*. And, since given-name and date fields in other christening record-templates have the same semantic IDs, states for these fields are also labeled—thousands of them due to the date variations (abbreviated/non-abbreviated months and single-digit/double-digit days) that appear in the *Kilbarchan Perish Record*. Furthermore, all delimiters are implicitly labeled whenever a user labels the fields in a record as the text between labeled fields and preceding the first labeled field and following the last labeled field. In the example in Figure 4, the user implicitly labels four delimiters: the comma and space between the name and the day in the date, the two spaces within the date, and the period following the year. The states for delimiters also have semantic IDs, so ListReader propagates the labels to all other states with identical semantic IDs—those that have the same delimiter in the same position in the same field group template.

ListReader's label propagation across semantic IDs minimizes the user's labeling effort during active sampling. As an example, Figure 5 shows ListReader's second active-sampling query for our example run of the *Kilbarchan Parish Record*. The highlighting is multicolored: green for previously labeled fields (the *GivenName* "Robert", the *ChristeningDate.Day* "3", the *ChristeningDate.Month* "Oct", and the *ChristeningDate.Year* "1709"); red for previously labeled delimiters (", ", " ", " ", and "."); and yellow for unlabeled text (the period following "Oct" in Figure 5). ListReader does not know, by what it has so far learned, whether the period following "Oct" belongs to the *Month* field or to the delimiter between "Oct" and "1709". At this point, the user should direct the form-field focus to the *ChristeningDate.Month* field and click on the period in "Oct." to append it to the already labeled and thus already appearing text "Oct" in the *ChristeningDate.Month* field in the form.

As our Kilbarchan example shows, active sampling is impactful from the first query. Furthermore, it improves recall monotonically as it does not back-track or reverse labeling decisions from one cycle to the next. Compared with typical active learning [19], it is not necessary for ListReader to induce an intermediate model from labeled data before it can become effective at issuing queries. This would be true even if ListReader did update the HMM during active learning cycles, although it would necessitate ListReader having to apply the HMM again on every cycle, which currently it avoids. Furthermore, ListReader need not know all the labels at the time of the first query. Indeed, it starts active sampling without knowing any labels. The query policy is similar to processes of novelty detection [16] in that it effectively identifies new structures for which a label is unknown and which most likely has the greatest effect on learning. Furthermore, the wrapper can be induced for complete records regardless of how much the user annotates or wants extracted, and ListReader is not dependent on the user to identify record- or field-delimiters nor to label any field the user does not want to be extracted.

5. EVALUATION

We evaluated ListReader with two historical books, the *Shaver-Dougherty Genealogy* [20] and the *Kilbarchan Parish Register* [8], and compared its performance to two baselines, an implementation of the Conditional Random Field (CRF) and a previous version of ListReader that induced regular-expression wrappers instead of HMM wrappers [18]. The regex version of ListReader is similar to the HMM version except that it creates separate regular-expression wrappers for every record pattern discovered during grammar induction whereas the HMM version is selective about which record and field group templates make it into the final HMM wrapper. The motivation for creating the HMM version is to overcome the brittleness of regular expressions, believing that the more malleable HMM wrappers would yield better recall results because of their ability to recognize variations in text patterns without requiring an exact match and would not hurt precision results too much because of ListReader's ability to create HMMs with a high degree of correlation to the observed text.

The CRF implementation we applied is from the Mallet library [17]. To ensure a strong baseline, we performed feature engineering work to select an appropriate set of word token features that allowed the CRF to perform well on development test data. We simulated active learning of a CRF

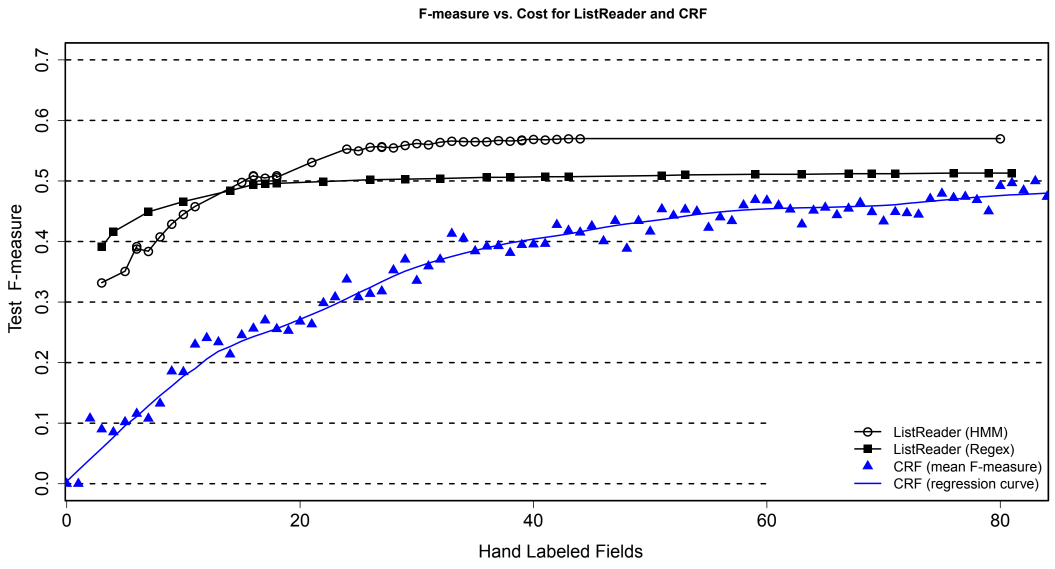


Figure 6: F-measure Learning Curves for the *Shaver-Dougherty Genealogy*

Table 1: Ground Truth Characteristics

Characteristic	*Shaver-Doughtery*	*Kilbarchan*
pages	498	143
labeled pages	68	3
labeled word tokens	14,314	852
labeled field instances	13,748	768
record instances	2,516	165
field types	46	12

using a random sampling strategy—considered to be a hard baseline to beat in active learning research, especially early in the learning process [2].

Since our aim is to develop a system that accurately extracts information at a low cost to the user, our evaluation centers on a standard metric in active learning research that combines both accuracy and cost into a single measurement: Area under the Learning Curve (ALC) [2]. The curve of interest for an extractor is the set of an extractor's accuracies plotted as a function of their respective costs. The ALC is the percentage of the area, between 0% and 100% accuracy and *min* and *max* cost, that is covered by the extractor's accuracy curve. ALC is equivalent to taking the mean of the accuracy metric at all points along the curve over the cost domain —an integral, which we computed for discrete values using the Trapezoidal Rule. A sample of one of the learning curves is in Figure 6.

The experimental results are based on a ground-truth, whose characteristics are described in Table 1. Visually, the learning curves in Figure 6 indicate that ListReader (Regex and HMM) both outperform the CRF fairly consistently over varying costs (varying numbers of human-labeled fields). Tables 2 and 3 succinctly summarizes the results and tell us that the difference among the three extraction wrappers are statistically significant for most pairwise comparisons.

Table 4 shows the space and time characteristics of the

Table 2: *Shaver-Doughterty* ALC Metrics (%) (All differences are statistically significant at $p<0.05$ using an unpaired t test except for the difference in Recall of ListReader-Regex and the CRF.)

	Prec.	Rec.	F_1
CRF	50.63	33.95	38.82
ListReader (Regex)	**97.60**	32.55	48.78
ListReader (HMM)	69.59	**42.84**	**52.54**

Table 3: *Kilbarchan* ALC Metrics (%) (All differences are statistically significant at $p<0.05$ using an unpaired t test except for the difference in Precision of the two ListReaders and the difference in Recall of ListReader-Regex and the CRF.)

	Prec.	Rec.	F_1
CRF	68.86	63.02	65.47
ListReader (Regex)	**96.34**	54.30	67.92
ListReader (HMM)	**91.38**	**72.74**	**79.19**

extraction wrappers. We ran all wrappers on a desktop computer with Java (JDK 1.7), a 2.39 GHz processor, and 3.25 GB of RAM. The smaller number of states of the CRF probably contributed to its faster running time and lower accuracy compared to ListReader. ListReader's time and space complexity is linear in terms of the size of the input text. Unlike the Regex version, which is also linear in its time complexity, the HMM version is quadratic in the average size of a record and the size of the label alphabet. The typical implementation of the training phase of a linear chain CRF is quadratic in both the sizes of the input text and the label set [4], [10].

6. CONCLUDING REMARKS

ListReader addresses the problem of extracting information from OCRed lists for ontology population. It requires

Table 4: Space and Time Characteristics

	ListReader		CRF
	HMM	Regex	
	Extractor Size		
	# states	# chars.	# states
Shaver-Doughtery	2,015	319,096	28
Kilbarchan	255	54,600	15
	Running Time		
Shaver-Doughtery	59m 18s	2m 47s	52s
Kilbarchan	2m 11s	26s	9s

little effort to apply to a new book, is specialized to recognize and model list structures, and is tolerant of OCR errors.

Our HMM implementation of ListReader demonstrates a novel way to set the structure and parameters of an HMM automatically for the task of populating an expressive ontological conceptualization with information from lists in OCRed text. It also demonstrates a way to minimize the work necessary for completing the HMM wrapper by manually associating automatically-selected HMM states with ontology predicates. ListReader performs well in terms of accuracy, user labeling cost, time and space complexity, and required knowledge engineering—outperforming the comparison systems in terms of most criteria including the most important measure: accuracy achieved relative to minimal manual annotation cost.

In future work, we expect to be able to further reduce the cost of manual labeling by a bootstrapping technique that makes use of previously learned extraction patterns to automatically propose labels.

7. ACKNOWLEDGMENTS

We would like to thank FamilySearch.org for supplying data from its scanned book collection and for their encouragement in this project. We would also like to thank the members of the BYU Data Extraction Research Group, and particularly Stephen W. Liddle and Christopher Almquist, for coding the user interface used for ground truthing and for interactively supplying labels for ListReader and for providing additional tools and resources for completing our ListReader project.

8. REFERENCES

[1] N. Ashish and C. A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *Proceedings of the Second IFCIS International Conference on Cooperative Information Systems, 1997. COOPIS '97*, pages 160–169, 1997.

[2] G. C. Cawley. Baseline methods for active learning. *Journal of Machine Learning Research-Proceedings Track*, 16:47–57, 2011.

[3] C.-H. Chang, C.-N. Hsu, and S.-C. Lui. Automatic information extraction from semi-structured web pages by pattern discovery. *Decision Support Systems*, 35:129–147, 2003.

[4] T. A. Cohn. *Scaling conditional random fields for natural language processing*. PhD thesis, Citeseer, 2007.

[5] N. Dalvi, R. Kumar, and M. Soliman. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4:219–230, 2010.

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

[7] FamilySearch. https://familysearch.org/.

[8] F. J. Grant, editor. *Index to the Register of Marriages and Baptisms in the Parish of Kilbarchan, 1649 - 1772*. Scottish Record Society. J. Skinner and Company, Ltd., Edinburgh, Scotland, 1912.

[9] T. Grenager, D. Klein, and C. D. Manning. Unsupervised learning of field segmentation models for information extraction. In *Proceedings of the Forty-third Annual Meeting on Association for Computational Linguistics*, pages 371–378, Ann Arbor, Michigan, USA, 2005.

[10] Y. Z. Guo, K. Ramamohanarao, and L. A. F. Park. Error correcting output coding-based conditional random fields for web page prediction. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 1, pages 743–746. IEEE, 2008.

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

[12] R. A. Haertel, E. K. Ringger, J. L. Carroll, and K. D. Seppi. Return on investment for active learning. In *Proceedings of the Neural Information Processing Systems Workshop on Cost Sensitive Learning*, 2008.

[13] W. Hu, W. Hu, N. Xie, and S. Maybank. Unsupervised active learning based on hierarchical graph-theoretic clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(5):1147–1161, Oct. 2009.

[14] N. Kushmerick. *Wrapper induction for information extraction*. PhD thesis, University of Washington, Seattle, Washington, USA, 1997.

[15] 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.

[16] S. Marsland. Novelty detection in learning systems. *Neural computing surveys*, 3(2):157–195, 2003.

[17] A. K. McCallum. MALLET: A machine learning for language toolkit, 2002.

[18] T. L. Packer and D. W. Embley. Scalable recognition, extraction, and structuring of data from lists in OCRed text using unsupervised active wrapper induction. Technical report, Department of Computer Science, Brigham Young University, Provo, Utah, 2014. (Submitted to TKDD).

[19] B. Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, June 2012.

[20] H. E. Shaffer. *Shaver/Shafer and Dougherty/Daughery Families also Kiser, Snider and Cottrell, Ferrell, Hively and Lowe Families*. Gateway Press, Inc., Baltimore, MD, 1997.

[21] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.