HTML Structure Meets Content

William W. Cohen
Read The Web, April 2006
Outline

• Motivation: finding even simple structures like lists is useful, and seems like it should be easy.

• Cohen & Fan, 1999a: List-finding as classification.

• Cohen 1999b, 2000: List-finding as matching structure to content.

Observation: Recognizing Structure is Useful

LISTEXTRACTOR(seedExamples)
   documents = searchForDocuments(seedExamples)
   For each document in documents
      parseTree = ParseHTML(document)
      For each subtree in parseTree
         keyWords = findAllSeedsInTree(subtree)
         prefix = findBestPrefix(keyWords, subtree)
         suffix = findBestSuffix(keyWords, subtree)
         Add to wrapperTree from createWrapper(prefix, suffix))
   For each goodWrapper in wrapperTree
      Find extractions using goodWrapper
   Return list of extractions

Figure 14: High-level pseudocode for List Extractor
Observation: Recognizing Structure is Useful

Experiment 10: Number of correct instances of Film at precision .90 and .80. List Extractor gives a 7-fold increase at precision .90 and an 8-fold increase at precision .80.
Observation: HTML Structure is Meaningful and (Easily?) Recognizable

“Colorless green ideas sleep furiously.”

Exploding porpoises, over four score and seven, well before configuration.

- Department of Computer and Information Sciences, University of New Jersey. Citrus flavorings: green, marine, clean and under lien.
- Computer Engineering Center, Lough Polytechnical Institute. This, that page extensionally left to rights of manatees.
- Electrical Engineering and Computer Science Dept, Bismark State College. Tertiary; where cola substitutes are frequently underutilized.

This page under construction. (Last update: 9/23/98.)

Figure 1: Nonsense text with a meaningful structure.
List-finding as Classification

[Cohen & Fan, WWW 1999]

Learning to extract “simple lists” and “simple hotlists”.

HTML source for a simple list:
<html><head>...</head>
<body>
<h1>Editorial Board Members</h1>
<table>
<tr>
<td>G. R. Emlin, Lucent</td>
<td>Harry Q. Bovik, Cranberry U</td>
</tr>
<tr>
<td>Bat Gangley, UC/Bovine</td>
<td>Pheobe L. Mind, Lough Tech</td>
</tr>
...

Extracted data:

<table>
<thead>
<tr>
<th>G. R. Emlin, Lucent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harry Q. Bovik, Cranberry U</td>
</tr>
</tbody>
</table>
...

HTML source for a simple hotlist:
<html><head>...</head>
<body>
<h1>Publications for Pheobe Mind</h1>
<ul>
</ul>
<li>A linear-time version of GSAT</li>
...

Extracted data:

<table>
<thead>
<tr>
<th>Optimization ...(PDF)</th>
<th>buzz.pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>A linear-time version of ...</td>
<td>peqnp.ps</td>
</tr>
</tbody>
</table>
...

Figure 2: A simple list, a simple hotlist, and the data that would be extracted from each.
List-finding as classification

In a page containing a *simple list*, the structure extracted is a one-column relation containing a set of strings $s_1,...,s_N$, and each $s_i$ is all the text that falls below some node $n_i$ in the parse tree. In a *simple hotlist*, the extracted structure is a two-column relation, containing a set of pairs $<s_1,u_1>,...,<s_N,u_N>$; each $s_i$ is all the text that falls below some node $n_i$ in the parse tree; and each $u_i$ is a URL that is associated with some HTML anchor element $a_i$ that appears somewhere inside $n_i$.

Technique: classify each node in the HTML tree as “extract” or “don’t extract”; then reconstruct the list or hotlist.

Evaluation: on a set of 84 pre-wrapped simple lists and simple hotlists (about 75% of a larger collection of wrapped pages).
List-finding as classification

- **Simple Features:**
  - Tag Name ("a", "p", "td")
  - Text Length, Non-white Text Length
  - Recursive Text Length, ...
  - Depth, NumChildren, NumSiblings
  - Parent tag, Ancestor tags, Child tags, Descendent tags

- **Complex features:**
  - tagSeqPosition \( TSP(n) \) = sequence of tags encountered walking from root to node \( n \)
  - NodePrefixCount\( (n) \) = \(|\{ \text{leaf } n' | TSP(n) \text{ prefix of } TSP(n')\}|\)
  - NodeSuffixCount\( (n) \) = ...
List-finding as classification
List-finding as classification

**Figure**: Performance of RIPPER in leave-one-page out experiments.

<table>
<thead>
<tr>
<th>Performance Level</th>
<th># pages reached</th>
</tr>
</thead>
<tbody>
<tr>
<td>perfect</td>
<td>26/84 31%</td>
</tr>
<tr>
<td>good (ε =1%)</td>
<td>33/84 39%</td>
</tr>
<tr>
<td>good (ε =3%)</td>
<td>35/84 39%</td>
</tr>
<tr>
<td>good (ε =5%)</td>
<td>41/84 49%</td>
</tr>
<tr>
<td>good (ε =10%)</td>
<td>45/84 54%</td>
</tr>
<tr>
<td>good (ε =15%)</td>
<td>47/84 56%</td>
</tr>
<tr>
<td>good (ε =20%)</td>
<td>48/84 57%</td>
</tr>
<tr>
<td>good (ε =25%)</td>
<td>48/84 57%</td>
</tr>
</tbody>
</table>
Another Experiment:
Wrapper Induction: User Labels $K$ Positive Examples
(Hybrid: After Accepting/Rejecting Default Wrapper)

**Figure**: Performance of the intra-page learning method and the hybrid intra-page and page-independent learning method as the number of positive examples $K$ is increased.
Outline

• Motivation: finding even simple structures like lists is useful, and seems like it should be easy.

• Cohen & Fan, 1999: List-finding as *classification*: kind of *disappointing*, only 30-50% of the pages were wrapped well.

• Cohen 2000a, 2000b: List-finding as *matching* structure to content.

Matching Structure To Content

To encode an HTML page in WHIRL, the page is first parsed. The HTML parse tree is then represented with the following EDB predicates.

- \( \text{elt}(Id, Tag, Text, Position) \) is true if \( Id \) is the identifier for a parse tree node, \( n \), \( Tag \) is the HTML tag associated with \( n \), \( Text \) is all of the text appearing in the subtree rooted at \( n \), and \( Position \) is the sequence of tags encountered in traversing the path from the root to \( n \). The value of \( Position \) is encoded as a a document containing a single term \( t_{pos} \), which represents the sequence, e.g., \( t_{pos} = \text{“html_body.ul.li”} \).

- \( \text{attr}(Id, AName, AValue) \) is true if \( Id \) is the identifier for node \( n \), \( AName \) is the name of an HTML attribute associated with \( n \), and \( AValue \) is the value of that attribute.

- \( \text{path}(FromId, ToId, Tags) \) is true if \( Tags \) is the sequence of HTML tags encountered on the path between nodes \( FromId \) and \( ToId \). This path includes both endpoints, and is defined if \( FromId=ToId \).

As an example, wrappers for the pages in Figure 2 can be written using these predicates as follows.

\[
\text{page1}(NameAffil) \leftarrow \\
\text{elt}(\_\_\_, \_\_\_, \text{NameAffil}, \text{“html_body_table_tr_td”}). \\
\text{page2}(Title, Url) \leftarrow \\
\text{elt}(ContextElt, \_\_\_, \text{Title}, \text{“html_body_ul_li”}) \\
\wedge \text{path}(ContextElt, AnchorElt, \text{“li_a”}) \\
\wedge \text{attr}(AnchorElt, \text{“href”}, \text{Url}).
\]

**HTML source for a simple list:**

\[
\text{<html><head>…</head>} \\
\text{<body>} \\
\text{<h1>Editorial Board Members</h1>} \\
\text{<table> <tr>} \\
\text{<td>G. R. Emlin, Lucent</td>} \\
\text{<td>Harry Q. Bovik, Cranberry U</td>} \\
\text{</tr>} \\
\text{<tr>} \\
\text{<td>Bat Ganglely, UC/Bovine</td>} \\
\text{<td>Pheobe L. Mind, Lough Tech</td>} \\
\text{</tr>} \\
\text{</table> </body>} \\
\text{</html>}
\]

**Extracted data:**

<table>
<thead>
<tr>
<th>G. R. Emlin, Lucent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harry Q. Bovik, Cranberry U</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Matching Structure To Content

To encode an HTML page in WHIRL, the page is first parsed. The HTML parse tree is then represented with the following EDB predicates.

- \textit{elt}(\textit{Id}, \textit{Tag}, \textit{Text}, \textit{Position}) is true if \textit{Id} is the identifier for a parse tree node, \textit{n}, \textit{Tag} is the HTML tag associated with \textit{n}, \textit{Text} is all of the text appearing in the subtree rooted at \textit{n}, and \textit{Position} is the sequence of tags encountered in traversing the path from the root to \textit{n}. The value of \textit{Position} is encoded as a a document containing a single term \( t_{pos} \), which represents the sequence, e.g., \( t_{pos} = \text{“html\_body\_ul\_li”} \).

- \textit{attr}(\textit{Id}, \textit{AName}, \textit{AValue}) is true if \textit{Id} is the identifier for node \textit{n}, \textit{AName} is the name of an HTML attribute associated with \textit{n}, and \textit{AValue} is the value of that attribute.

- \textit{path}(\textit{FromId}, \textit{ToId}, \textit{Tags}) is true if \textit{Tags} is the sequence of HTML tags encountered on the path between nodes \textit{FromId} and \textit{ToId}. This path includes both endpoints, and is defined if \textit{FromId} = \textit{ToId}.

As an example, wrappers for the pages in Figure 2 can be written using these predicates as follows.

\begin{verbatim}
page1(NameAffil) ←
ell(\_, \_, NameAffil, “html\_body\_table\_tr\_td”).
page2(Title, Url) ←
ell(ContextElt, \_, Title, “html\_body\_ul\_li”)
∧ path(ContextElt, AnchorElt, “li\_a”)
\end{verbatim}

\textbf{HTML source for a simple hotlist:}
\begin{verbatim}
<html>
<head>…</head>
<body>
<h1>Publications for Pheobe Mind</h1>
<ul>
<li>A linear-time version of GSAT</li>
…
</ul>
</body>
</html>
\end{verbatim}

\textbf{Extracted data:}
\begin{tabular}{|l|l|}
\hline
Optimization … (PDF) & buzz.pdf \\
A linear-time version of … & peqnp.ps \\
\hline
\end{tabular}
Matching Structure To Content

Key point: every simple list or hotlist can be written *just like this*: you just need to fill in

- one root-node path for simple lists;
- one root-node path and one node-node path for simple hotlists.

⇒ Given a web page with $N$ nodes, there are $O(N^2)$ possible wrappers, which can be enumerated and scored.

As an example, wrappers for the pages in Figure 2 can be written using these predicates as follows.

page1(NameAffil) ←
elt(_, _, NameAffil, “html_body_table_tr_td”).

page2(Title,Url) ←
elt(ContextElk, _, Title, “html_body_ul_li”) ∧
path(ContextElk, AnchorElk, “li_a”) ∧

**HTML source for a simple hotlist:**
<html><head>...</head>
<body><h1>Publications for Pheobe Mind</h1><ul>
<li>A linear-time version of GSAT</li>
...;

<table>
<thead>
<tr>
<th>Extracted data:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization ...(PDF)</td>
<td>buzz.pdf</td>
<td></td>
</tr>
<tr>
<td>A linear-time version of ...</td>
<td>peqnp.ps</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Enumerating and Scoring Wrappers

Enumerating and Scoring Wrappers

<table>
<thead>
<tr>
<th>Enumerating and Scoring Wrappers</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruitful_piece(Path1, Path2) ←</td>
</tr>
<tr>
<td>possible_piece(Path1, Path2) ∧</td>
</tr>
<tr>
<td>many(extracted_by(Path1a, Path2a, -),</td>
</tr>
<tr>
<td>(Path1a = Path1 ∧ Path2a = Path2).</td>
</tr>
<tr>
<td>possible_piece(Path1, Path2) ←</td>
</tr>
<tr>
<td>elt(TextElt, -, -, Path1)</td>
</tr>
<tr>
<td>∧ elt(AnchorElt, -, “a”, -)</td>
</tr>
<tr>
<td>∧ attr(AnchorElt, “href”, -)</td>
</tr>
<tr>
<td>∧ path(TextElt, AnchorElt, Path2).</td>
</tr>
<tr>
<td>extracted_by(Path1, AnchorElt, Path2) ←</td>
</tr>
<tr>
<td>elt(TextElt, -, -, Path1)</td>
</tr>
<tr>
<td>∧ path(TextElt, AnchorElt, Path2).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enumerating and Scoring Wrappers</th>
</tr>
</thead>
<tbody>
<tr>
<td>anchorlike_piece(Path1, Path2) ←</td>
</tr>
<tr>
<td>possible_piece(Path1, Path2) ∧</td>
</tr>
<tr>
<td>many(extracted_by(Path1a, Path2a, TElt, AElt),</td>
</tr>
<tr>
<td>(Path1a = Path1 ∧ Path2a = Path2)</td>
</tr>
<tr>
<td>∧ elt(TElt, -, Text1, -) ∧ elt(AElt, -, Text2, -) ∧ Text1 ~ Text2).</td>
</tr>
<tr>
<td>R-like_piece(Path1, Path2) ←</td>
</tr>
<tr>
<td>possible_piece(Path1, Path2) ∧</td>
</tr>
<tr>
<td>many(R-extracted_by(Path1a, Path2a, -),</td>
</tr>
<tr>
<td>(Path1a = Path1 ∧ Path2a = Path2).</td>
</tr>
<tr>
<td>R-extracted_by(Path1, Path2, TextElt, AnchorElt) ←</td>
</tr>
<tr>
<td>elt(TextElt, -, Text, Path1)</td>
</tr>
<tr>
<td>∧ path(TextElt, AnchorElt, Path2)</td>
</tr>
<tr>
<td>∧ R(X) ∧ Text ~ X.</td>
</tr>
</tbody>
</table>

Figure 3: WHIRL programs for recognizing plausible structures in an HTML page. (See text for explanation.)
Enumerating and Scoring Wrappers

Enumerates all Path1, Path2 such that Path1 goes from root to n, and Path2 goes from n to an anchor element.

---

Figure 3: WHIRL programs for recognizing plausible structures in an HTML page. (See text for explanation.)
Enumerating and Scoring Wrappers

Enumerates and scores Path1, Path2 according to how many things are extracted by that wrapper.

Figure 3: WHIRL programs for recognizing plausible structures in an HTML page. (See text for explanation.)
Enumerating and Scoring Wrappers

“Soft” predicate, true/false according to TFIDF similarity

Enumerates and scores Path1, Path2 according to how many things are extracted by that wrapper such that the text under node $n$ is close to the text under the anchor element.

e.g., discounts wrappers where $n$ is close to the root.

Figure 3: WHIRL programs for recognizing plausible structures in an HTML page. (See text for explanation.)
Enumerating and Scoring Wrappers

Enumerates and scores Path1, Path2 according to how many things are extracted by that wrapper such that the text under node $n$ is close to the text of some $X$ such that $R(X)$ is true.

$R(X)$ is “content” information, as a K-NN classifier.

Figure 3: WHIRL programs for recognizing plausible structures in an HTML page. (See text for explanation.)
Results

Figure 4: Performance of ranking heuristics that use little or no page-specific information.

<table>
<thead>
<tr>
<th>Oklahoma</th>
<th>dietitians</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yukon</td>
<td>Yukon codpiece</td>
</tr>
<tr>
<td>Vermont</td>
<td>Vermont</td>
</tr>
<tr>
<td>British Columbia</td>
<td>British Columbia Talmudizations</td>
</tr>
<tr>
<td>Oklahoma</td>
<td>Oklahoma</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>New Jersey</td>
<td>New Jersey incorrigible blubber</td>
</tr>
<tr>
<td>Alaska</td>
<td>Alaska</td>
</tr>
<tr>
<td>New Brunswick</td>
<td>New Mexico cryptogram</td>
</tr>
</tbody>
</table>

Table 2: Ten US States and Canadian Provinces, before and after corruption with $c = 1$.

Figure 5: Performance of ranking heuristics that use page-specific training examples.

Figure 6: Performance of ranking heuristics that use text extracted from an previous version of the page.
Using Extracted Lists

- Experiments above describe semi-automated techniques for wrapper learning: some user intervention is needed.
- Can you use the wrappers without letting a user check them?

**Idea [Cohen, ICML2000]:**
- Start with some textual examples for a classification problem (e.g., names of classical/rock musicians)
- Use these examples as “seeds” $R$ and find a bunch of simple lists $L_1, L_2, \ldots, L_m$
- Use each list as a *feature*: $F_i$ true for $x$ iff $x$ (approximately) matches something in list $L_i$.
- Also: used features for header words that seemed to modify the matching element of the list (kind of like anchor text).
Using Extracted Lists

Table 1. Benchmark problems used in the experiments.

<table>
<thead>
<tr>
<th></th>
<th>#example</th>
<th>#class</th>
<th>#terms</th>
<th>#pages</th>
<th>(Mb)</th>
<th>#features added</th>
<th>%examples expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>music</td>
<td>1010</td>
<td>20</td>
<td>1600</td>
<td>217</td>
<td>(11.7)</td>
<td>1890</td>
<td>68.7</td>
</tr>
<tr>
<td>games</td>
<td>791</td>
<td>6</td>
<td>1133</td>
<td>177</td>
<td>(2.5)</td>
<td>1169</td>
<td>53.0</td>
</tr>
<tr>
<td>birdcom</td>
<td>915</td>
<td>22</td>
<td>674</td>
<td>83</td>
<td>(2.2)</td>
<td>918</td>
<td>99.9</td>
</tr>
<tr>
<td>birdsci</td>
<td>915</td>
<td>22</td>
<td>1738</td>
<td>83</td>
<td>(2.2)</td>
<td>533</td>
<td>99.9</td>
</tr>
</tbody>
</table>
Using Extracted Lists
Using Extracted Lists

Table 1. Benchmark problems used in the experiments.

<table>
<thead>
<tr>
<th></th>
<th>#example</th>
<th>#class</th>
<th>#terms</th>
<th>#pages (Mb)</th>
<th>#features added</th>
<th>%examples expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>music</td>
<td>1010</td>
<td>20</td>
<td>1600</td>
<td>217 (11.7)</td>
<td>1890</td>
<td>68.7</td>
</tr>
<tr>
<td>games</td>
<td>791</td>
<td>6</td>
<td>1133</td>
<td>177 (2.5)</td>
<td>1169</td>
<td>53.0</td>
</tr>
<tr>
<td>birdcom</td>
<td>915</td>
<td>22</td>
<td>674</td>
<td>83 (2.2)</td>
<td>918</td>
<td>99.9</td>
</tr>
<tr>
<td>birdsci</td>
<td>915</td>
<td>22</td>
<td>1738</td>
<td>83 (2.2)</td>
<td>533</td>
<td>99.9</td>
</tr>
</tbody>
</table>

Table 5. Error rate of C4.5 and BoosTexter on the benchmark problems.

<table>
<thead>
<tr>
<th></th>
<th>RIPPER</th>
<th>C4.5</th>
<th>BoosTexter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W-L-T</td>
<td>W-L-T</td>
<td>W-L-T</td>
</tr>
<tr>
<td></td>
<td>expand</td>
<td>expand</td>
<td>expand</td>
</tr>
<tr>
<td></td>
<td>text</td>
<td>text</td>
<td>text</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>music</td>
<td>86-0-14</td>
<td>100-0-0</td>
<td>100-0-0</td>
</tr>
<tr>
<td></td>
<td>51.5</td>
<td>49.3</td>
<td>43.4</td>
</tr>
<tr>
<td></td>
<td>58.3</td>
<td>59.6</td>
<td>58.1</td>
</tr>
<tr>
<td>games</td>
<td>29-7-64</td>
<td>13-6-81</td>
<td>16-1-83</td>
</tr>
<tr>
<td></td>
<td>65.8</td>
<td>68.2</td>
<td>61.8</td>
</tr>
<tr>
<td></td>
<td>67.2</td>
<td>68.6</td>
<td>63.1</td>
</tr>
<tr>
<td>birdcom</td>
<td>77-2-21</td>
<td>97-0-3</td>
<td>65-1-34</td>
</tr>
<tr>
<td></td>
<td>21.2</td>
<td>31.8</td>
<td>21.3</td>
</tr>
<tr>
<td></td>
<td>27.7</td>
<td>40.7</td>
<td>25.3</td>
</tr>
<tr>
<td>birdsci</td>
<td>35-8-57</td>
<td>46-4-50</td>
<td>35-6-59</td>
</tr>
<tr>
<td></td>
<td>23.6</td>
<td>37.3</td>
<td>25.4</td>
</tr>
<tr>
<td></td>
<td>26.4</td>
<td>39.0</td>
<td>26.7</td>
</tr>
</tbody>
</table>
Outline

- **Motivation:** finding even simple structures like lists is useful, and seems like it should be easy.
- **Cohen & Fan, 1999a:** List-finding as *classification*: kind of disappointing, only 30-50% of the pages were wrapped well.
- **Cohen 1999b, 2000:** List-finding as *matching* structure to content: seems to be effective even with moderately good models of content.
• Previous work in page classification using links:
  • Exploit hyperlinks (Slattery&Mitchell 2000; Cohn&Hofmann, 2001; Joachims 2001): Documents pointed to by the same “hub” should have the same class.

• What’s new in this paper (Cohen NIPS 2002):
  • Use structure of hub pages (as well as structure of site graph) to find better “hubs”
  • Adapt an existing “wrapper learning” system to find structure, on the task of classifying “executive bio pages”.
Intuition: links from this “hub page” are informative...

...especially these links
Background: “wrapper” learning

• System is based on a number of “builders”:
  – Infer a “structure” (e.g. a list, table column, etc) from few positive examples.
  – A “structure” extracts all its members
    • $f(page) = \{ x: x \text{ is a “structure element” on } page \}$

• A master algorithm co-ordinates the “builders”
• Add/remove “builders” to optimize performance on a domain (Cohen, Hurst & Jensen WWW-2002)
• Some builder usually obtains a good generalization from only 2-3 positive examples
Search Results

Click on the title for a detailed description of the position:

- Marketing Research Assistant
- Product Developer - Confections
- Oral Care Products Researcher
- Line Mechanic
- Research and Development Senior Process Engineer
- Sensory Specialist
- Territory Manager (Michigan)
- Territory Manager - Colorado
- General Maintenance Mechanic B
- Global Standards Manager
- Corporate Manager - Occupational Health
- Program Management Analyst, Supply Chain
- International Sales Representative
- General Supervisor - Mechanical Services
- Internet - Intranet Coordinator
- Assistant to Claims Support Manager
Experimental results:
2-3 examples leads to high average accuracy

F1

#examples
Background: “co-training” (Mitchell&Blum, ‘98)

• Suppose examples are of the form \((x_1,x_2,y)\) where \(x_1, x_2\) are **independent** (given \(y\)), and where each \(x_i\) is sufficient for classification, and **unlabeled** examples are cheap.
  - (E.g., \(x_1 = \) bag of words, \(x_2 = \) bag of links).

• Co-training algorithm:
  1. Use \(x_1\)’s (on labeled data \(D\)) to train \(f_1(x)=y\)
  2. Use \(f_1\) to label additional **unlabeled** examples \(U\).
  3. Use \(x_2\)’s (on labeled part of \(U+D\)) to train \(f_1(x)=y\)
  4. Repeat . . .
Simple 1-step co-training for web pages

\( f_1 \) is a bag-of-words page classifier, and \( S \) is a web site containing unlabeled pages.

- **Feature construction.** Represent a page \( x \) in \( S \) as a bag of pages that link to \( x \) ("bag of hubs").
- **Learning.** Learn \( f_2 \) from the bag-of-hubs examples, labeled with \( f_1 \).
- **Labeling.** Use \( f_2(x) \) to label pages from \( S \).

Idea. use one round of co-training to bootstrap the bag-of-words classifier to one that uses site-specific features. \( x_2 \sim f_2 \)
Improved 1-step co-training for web pages

Feature construction.

- Label an anchor \( a \) in \( S \) as **positive** iff it points to a positive page \( x \) (according to \( f_1 \)). Let \( D = - (x', a): a \) is a positive anchor on \( x' \)

- Generate many small training sets \( D_i \) from \( D \), by sliding small windows over \( D \).
- Let \( P \) be the set of all “structures” found by any builder from any subset \( D_i \)
- Say that \( p \) **links to** \( x \) if \( p \) extracts an anchor that points to \( x \). Represent a page \( x \) as the bag of **structures** in \( P \) that link to \( x \).

Learning and Labeling. As before.
List2

builder

extractor
Storage Access has assembled a world-class management team with the vision and understanding that makes them pioneers in the Storage Service Provider (SSP) industry. Our mission is clear: to become the leading global provider of managed storage solutions.

List3

builder

extractor
BOH representation:

- \{ \text{List1, List3,\ldots} \}, \text{PR}
- \{ \text{List1, List2, List3,\ldots} \}, \text{PR}
- \{ \text{List2, List3,\ldots} \}, \text{Other}
- \{ \text{List2, List3,\ldots} \}, \text{PR}

\ldots
Experimental results

Co-training hurts

No improvement
## Experimental results

<table>
<thead>
<tr>
<th>Site</th>
<th>Classifier $f_1$</th>
<th>1-CT (dtree)</th>
<th>1-CT (Winnow)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc (SE)</td>
<td>Acc (SE)</td>
<td>Acc (SE)</td>
</tr>
<tr>
<td>1</td>
<td>1.000 (0.000)</td>
<td>0.960 (0.028)</td>
<td>0.960 (0.028)</td>
</tr>
<tr>
<td>2</td>
<td>0.932 (0.027)</td>
<td>0.955 (0.022)</td>
<td>0.955 (0.022)</td>
</tr>
<tr>
<td>3</td>
<td>0.813 (0.028)</td>
<td>0.934 (0.018)</td>
<td>0.939 (0.017)</td>
</tr>
<tr>
<td>4</td>
<td>0.904 (0.029)</td>
<td>0.962 (0.019)</td>
<td>0.962 (0.019)</td>
</tr>
<tr>
<td>5</td>
<td>0.939 (0.024)</td>
<td>0.960 (0.020)</td>
<td>0.960 (0.020)</td>
</tr>
<tr>
<td>6</td>
<td>1.000 (0.000)</td>
<td>1.000 (0.000)</td>
<td>1.000 (0.000)</td>
</tr>
<tr>
<td>7</td>
<td>0.918 (0.028)</td>
<td>0.990 (0.010)</td>
<td>0.990 (0.010)</td>
</tr>
<tr>
<td>8</td>
<td>0.788 (0.044)</td>
<td>0.882 (0.035)</td>
<td>0.929 (0.028)</td>
</tr>
<tr>
<td>9</td>
<td>0.948 (0.029)</td>
<td>0.948 (0.029)</td>
<td>0.983 (0.017)</td>
</tr>
</tbody>
</table>
Summary

- “Builders” (from a wrapper learning system) let one discover and use structure of web sites and index pages to smooth page classification results.

- Discovering good “hub structures” makes it possible to use 1-step co-training on small (50-200 example) unlabeled datasets.
  - Average error rate was reduced from 8.4% to 3.6%.
  - Difference is statistically significant with a 2-tailed paired sign test or t-test.
  - EM with probabilistic learners also works—see (Blei et al, UAI 2002)
Formatting is regular on each site, but there are too many different sites to wrap. Can we get the best of both worlds?
Scoped Learning Generative Model

1. For each of the D documents:
   a) Generate the multinomial formatting feature parameters $\phi$ from $p(\phi|\alpha)$

2. For each of the N words in the document:
   a) Generate the $n$th category $c_n$ from $p(c_n)$.
   b) Generate the $n$th word (global feature) from $p(w_n|c_n, \theta)$
   c) Generate the $n$th formatting feature (local feature) from $p(f_n|c_n, \phi)$

$$p(\phi, c, w, f) = p_\alpha(\phi) \prod_{n=1}^{N} p(c_n)p_\theta(w_n|c_n)p(f_n|c_n, \phi)$$
Inference

Given a new web page, we would like to classify each word resulting in \( c = \{c_1, c_2, ..., c_n\} \)

\[
p(c|w, f) = \frac{\int \prod_{n=1}^{N} p(w_n|c_n)p(f_n|c_n, \phi)p(c_n)p(\phi)d\phi}{\int \prod_{n=1}^{N} \sum_{c_n} p(w_n|c_n)p(f_n|c_n, \phi)p(c_n)p(\phi)d\phi}
\]

This is not feasible to compute because of the integral and sum in the denominator. We experimented with two approximations:

- MAP point estimate of \( \phi \)
- Variational inference
**MAP Point Estimate**

If we approximate $\phi$ with a point estimate, $\hat{\phi}$, then the integral disappears and $c$ decouples. We can then label each word with:

$$\hat{c}_n = \arg \max_{c_n} p(w_n | c_n)p(f_n | c_n, \hat{\phi})p(c_n)$$

A natural point estimate is the posterior mode: a maximum likelihood estimate for the local parameters given the document in question:

$$\hat{\phi} = \arg \max_{\phi} p(\phi | f, w)$$

**E-step:**

$$p^{(t+1)}(c_n | w_n, f_n; \hat{\phi}) \propto p^{(t)}(f_n | c_n; \hat{\phi})p(w_n | c_n)p(c_n)$$

**M-step:**

$$\hat{\phi}_{c,f} = p^{(t+1)}(f | c; \hat{\phi}) \propto \sum_{\{n: c_n = c, f_n = f\}} p^{(t)}(c_n | f_n, w_n)$$
Family Services Director: Creative, energetic and enjoy working with people? Seeking director for program development, implementation and administration. Must possess a Bachelor’s Degree in Recreation, Family Studies or related field. Strong interpersonal and organizational skills a must. Excellent benefits. Send resumes to Jane Kim, Dir of Camping and Family Services, North Suburban YMCA, 2735 Techny Road, Northbrook, IL 60062.

Massage Therapist - Male: The North Suburban YMCA is seeking a certified massage therapist to work part time in our men’s program center. Flexible hours, y membership, on-site child care available if needed. Please contact Harlan Shrickson by email or call at 847-272-7260.

Starbucks Server: Early day, evening and weekend shifts available for in-house cafe serving the Starbucks product line. An exciting opportunity and membership is included. Contact Sarah Tucker at 847-272-7260 x 213.

Teacher for ChildCare Center: Part-time 2-6 pm, Monday through Friday. Minimum requirements are 50 college credit hours in Early childhood or Education or similar subject. At least one year experience working with 2-5 year olds. Contact Helen at (847) 272-7260 x 222 and fax resume to (847) 272-7587.

Art Coordinator: Creative? Enjoy working with children? the North Suburban YMCA is looking for an art coordinator for the summer. Call Jane at (347) 727-7260 for more information.

Teachers: Seeking part-time early childhood teachers for summer or all year. 2-3 mornings per week from 9am-11:15am. Free child care on-site while you work. Free YMCA membership. College degree required in education or related field. Pickup an application at the front desk or call Caren Shulman, Child Development Coordinator at (847) 272-7260 x 252.

Group Exercise Personality Training: Interested individuals with proper certification may contact Mykion Saprovini at (847) 272-7260 x 217.

Customer Service Rep: OVERQUALIFIED APPLY HERE! Hone your skills by working in a friendly environment. The front desk is looking for part-time staff to work flexible shifts for early weekday mornings, day and evening shifts. Benefits include YMCA membership and babysitting during your shift. Please contact Sarah Tucker or Cheryl Stewart at (347) 272-7260 x 213.

Lifeguards and Swim Instructors: Love to swim? Love kids? Put the two together and make a difference. The North Suburban YMCA is looking for qualified and experienced swim instructors and lifeguards. Please contact Jane Kim at 847-272-7260 x 213.

Global Extractor: Precision = 46%, Recall = 75%
Family Services Director
Creative, energetic and enjoy working with people? Seeking director for program development, implementation and administration. Must possess a Bachelor’s Degree in Recreation, Family Studies or related field. Strong interpersonal and organizational skills a must. Excellent benefits. Send resumes to Jane Idin, Dir of Camping and Family Services, North Suburban YMCA, 2705 Techny Road, Northbrook, IL 60062.

Massage Therapist - Male
The North Suburban YMCA is seeking a certified massage therapist to work part time in our men’s program center. Flexible hours, y membership, on-site child care available if needed. Please contact Harlan Schaff in仅 by email or call at 847-272-7250.

Starbucks Server
Early day, evening and weekend shifts available for in-house cafe serving the Starbucks product line. An exciting opportunity and membership is included! Contact Sarah Tucker at 847-272-7250 x 213.

Teacher for ChildCare Center
Part-time 2-6 pm, Monday through Friday. Minimum requirements are 60 college credit hours in Early childhood or Education or similar subject. At least one year experience working with 2-5 year olds. Contact Helen at (847) 272-7250 x 222 and fax resume to (847) 272-7587.

Art Coordinator
Creative? Enjoy working with children? the North suburban Y is looking for an art coordinator for the summer. Call Jane at (847)272-7250 for more information.

Teachers
Seeking part-time early childhood teachers for summer or all year. 2-3 mornings per week from 9am-11:15am. Free child care on-site while you work. Free YMCA membership. College degree required in education or related field. Pick up an application at the front desk or call Caryn Shuman, Child Development Coordinator at (847) 272-7250 x 202.

Group Exercise
Personal Training
Interested individuals with proper certification may contact Myleen Siperstein at (847) 272-7250 x 217.

Customer Service
Rep
OVERQUALIFIED APPLY HERE! Hone your skills by working in a friendly environment. The front desk is looking for part time staff to work flexible shifts for early weekday mornings, day and evening shifts. Benefits include YMCA membership and babysitting during your shift. Please contact Sarah Tucker or Cheryl Stewart at (847) 272-7250 x 213.

Lifeguards and Swim Instructors
Love to swim? Love kids? Put the two together and make a difference. The North Suburban YMCA is looking for qualified and experienced swim instructors and

Scoped Learning Extractor:  Precision = 58%, Recall = 75% △ Error = -22%
Outline

• Motivation: finding even simple structures like lists is useful, and seems like it should be easy.

• Cohen & Fan, 1999a: List-finding as classification.

• Cohen 1999b, 2000: List-finding as matching structure to content.