Extracting Names from Websites Containing Lists of People

William Gronim <wwg@andrew.cmu.edu>[1],

Edoardo Airoldi <eairoldi+@cs.cmu.edu>,

and Latanya Sweeney <latanya@privacy.cs.cmu.edu>

Data Privacy Laboratory, Institute for Software Research International

March 31st, 2006

School of Computer Science

Carnegie Mellon University

Pittsburgh, PA 15213

[1]Corresponding author.

Phone: (412) 268-4484

Fax: (412) 268-6708

1320 Wean Hall

5000 Forbes Ave

Pittsburgh, PA 15213

Abstract

In this paper we describe an automated system for extracting people's names from websites containing lists of people. The contents of these websites describe attributes common to the people listed. This public information has strategic value, such as demonstrating who tends to appear at similar events. Unlike traditional named entity recognition (NER) we are extracting names embedded in HTML without natural language context. We use a hidden markov model (HMM) to segment the document's HTML source in order to extract entire names. Engineering features for this classifier led us to several general types of features useful for segmenting text in structured documents. Rosters may order first and last names in many ways. A first/last classifier determines the ordering used by each document using dictionaries to provide partial knowledge of the distribution of names across token positions. The first/last classifier uses the two dimensional coordinates of text as it would appear when rendered by a browser in order to abstract away the HTML. The HMM segmenter was able to achieve 95% precision and 91% recall while the first/last classifier achieved 84% precision and 82% recall, on average in a corpus of 37 documents containing approximately 10,000 names.

Keywords: machine learning, information retrieval, hidden markov models, named entity recognition, text mining, rosters

1 Introduction

It is common practice to publish lists of names on the Web. Social clubs and internet communities publish lists of their members, universities publish lists of students and faculty, and corporations publish the names of their executives and board members. By combining information about interpersonal relationships from all these sources one can discover useful facts about an individual or group of individuals. For example, a university admissions office might want to compare a list of accepted student against a competing university's list of admitted students. In 2002, Yale University complained to the United States Federal Bureau of Investigation that someone in admissions at rival Princeton University had illegally gained access to Yale's admissions system in order to snoop on students who had applied to both schools (Ferdinand & Barbaro, 2002).

Universities could also mine their own sites for lists of names in order to evaluate their Family Educational Rights and Privacy Act (FERPA) (33 CFR Part 99) compliance. In an earlier experiment, MIT's stated FERPA policy prohibited web pages containing of lists of student names yet such lists were found on-line (Sweeney, 2004). In order to accomplish this automatically we must be able to locate web pages containing lists of names, extract the names, and then analyze the extracted information.

This work builds on previous work on the automatic location of lists of people on the web (Sweeney, 2004). In that work an algorithm named ”RosterFinder” was developed to locate these lists automatically. A ”roster” is defined in (Sweeney, 2004) as a web page consisting of one or more itemized lists of names appearing in some dictionary. Locating rosters by hand is tedious because the list structure required cannot be specified in a traditional keyword search. RosterFinder first uses one or more keyword searches to retrieve a set of candidate documents. Candidate documents are classified as rosters if a fraction of the words in a document which also appear in a dictionary is greater than a threshold. RosterFinder only locates rosters, it does not extract any information from them.

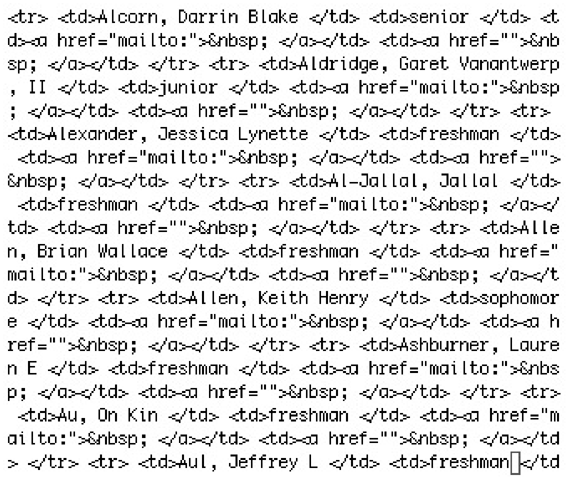
Extracting names from rosters presents several challenges. A dictionary alone cannot reliably classify names. Different rosters order the components of names differently, and some include components others do not. For example, a roster might present names in ”Last, First”, ”First Last”, or ”Last, First Middle” order. We must also extract names from many cultures. All the rosters located by RosterFinder in a previous project (Malin et al., 2003) used either HTML tables or lists to present names, but the details of the HTML differ significantly between documents. Many of the existing techniques for name extraction rely on the grammatical structure of the surrounding text to identify names. These techniques are not directly applicable here, because these names are surrounded only by HTML and have no grammatical context.

Once names are extracted the semantic information encoded by the rosters must be analyzed. Beyond merely listing student names, FERPA (33 CFR Part 99) concerns the publication of directory information such as phone numbers and mailing addresses, so this additional information would have to be extracted from rosters in a compliance application. Simple social networks can be built by considering all names appearing together on a roster to be related. The content and link structure of homepages was analyzed in (Adamic & Adar, 2001) to predict personal interests and friendships using techniques which could be adapted to rosters. The ReferralWeb (Kautz, Selman, & Shah, 1997) system was developed in order to use existing social networks to automatically locate experts on a given subject, and its sources of names could be expanded using automated extraction from rosters. The goal of this work is to develop an automated tool for name extraction in order to support research into these and other applications.

The rest of this paper is organized as follows. In section 2 we discuss related work in content extraction. In section 3 we present the design of our HMM segmenter and first/last classifier. In section 4 we describe the materials such as dictionaries and sample documents used in our experiments. In section 5 we describe the experimental performance of our system. In section 6 we discuss our results and possibilities for future research.

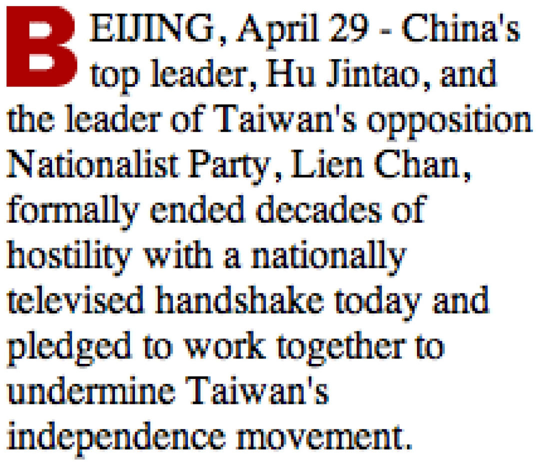
2 Related Work

The most closely related problem domains are protein name extraction from medical text and person, place, and organization recognition in news article text. Previous successful named entity recognition systems such as (Bikel, Schwartz, & Weischedel, 1999), (Chei & Ng, 2002), (Klein, Smarr, Nguyen, & Manning, 2003), (Isozaki & Kazawa, 2002), and (Kazuhiro & Mostafa, 2003) all apply to natural language text. These systems are therefor inapplicable to structured documents containing lists of names. Some of the features and classifiers developed for natural language text are immediately useful for structured documents, particularly dictionaries of names and HMMs. However, structured documents lack the natural language grammar frequently used in natural language named entity recognition. Figure 1 shows names presented in a roster versus names appearing within text. Prior work has addressed extracting names from text by using syntax and grammar appearing in the text. This work addresses extracting names from rosters which lack grammatical context.



```
<tr> <td>Alcorn, Darrin Blake </td> <td>senior </td> <t
d><a href="mailto:">&nbsp; </a></td> <td><a href="">&nb
sp; </a></td> </tr> <tr> <td>Aldridge, Garet Vanantwerp
, II </td> <td>junior </td> <td><a href="mailto:">&nbsp
; </a></td> <td><a href="">&nbsp; </a></td> </tr> <tr>
<td>Alexander, Jessica Lynette </td> <td>freshman </td>
 <td><a href="mailto:">&nbsp; </a></td> <td><a href="">
&nbsp; </a></td> </tr> <tr> <td>Al-Jallal, Jallal </td>
 <td>freshman </td> <td><a href="mailto:">&nbsp; </a></
td> <td><a href="">&nbsp; </a></td> </tr> <tr> <td>Alle
n, Brian Wallace </td> <td>freshman </td> <td><a href="
mailto:">&nbsp; </a></td> <td><a href="">&nbsp; </a></t
d> </tr> <tr> <td>Allen, Keith Henry </td> <td>sophomor
e </td> <td><a href="mailto:">&nbsp; </a></td> <td><a h
ref="">&nbsp; </a></td> </tr> <tr> <td>Ashburner, Laure
n E </td> <td>freshman </td> <td><a href="mailto:">&nbs
p; </a></td> <td><a href="">&nbsp; </a></td> </tr> <tr>
 <td>Au, On Kin </td> <td>freshman </td> <td><a href="m
ailto:">&nbsp; </a></td> <td><a href="">&nbsp; </a></td
> </tr> <tr> <td>Aul, Jeffrey L </td> <td>freshman </td
```

(a) HTML excerpt from a roster.



BEIJING, April 29 - China's top leader, Hu Jintao, and the leader of Taiwan's opposition Nationalist Party, Lien Chan, formally ended decades of hostility with a nationally televised handshake today and pledged to work together to undermine Taiwan's independence movement.

(b) English language text exercpt from the New York Times.

Figure 1: Comparison of a roster and some natural language text.

Others have studied the problem of retrieving structured knowledge from a document designed for visual presentation. In (Chung, Gertz, & Sundaresan, 2002) one type of information (such as a list of names) is extracted from a corpus of documents which all present this information using different markup. The user inputs a XML schema describing the desired structure, and then the system operates on the tree structure of each document in order to remove frivolous markup and group tags related by the target schema. The authors of (Adelberg & Denny, 1999) developed a similar system for extracting structured information from text documents, only to see their extractions silently begin producing garbage when minor changes to the document formats were introduced. They added modules which keep running statistics on the performance of each text to structure wrapper in order to detect anomalies. Our work is similar in so far as we are also using statistical methods to automatically extract structures from markup documents. Our work differs in that we are targeting only one sort of structure, and we are semantically interpreting the extracted results in our first/last classifier.

There has been extensive work on segmenting text documents to isolate certain structures. One such structure is a sequence of text concerning one subject or topic. TextTiling (Hearst, 1997) segments a document into multi-paragraph subtopics by identifying changes in word frequency. Cosine similarity between word frequency vectors is used to segment around points of low inter-paragraph similarity. This idea was extended by (Brants, Chen, & Tsochantaridis, 2002) to include semantic information. The similarity measure was extended to use Probabilistic Latent Semantic Analysis (PLSA) (Hofmann, 1999). A PLSA model is parameterized with a fixed number of latent semantic classes roughly representing topics and trained on labeled data. The segmentation algorithm of (Brants, Chen, & Tsochantaridis, 2002) improves on TextTiling's performance by creating multiple PLSA models with different parameters and using the average of their similarity measures to evaluate similarity between paragraphs. In (Choi, 2000) topic segmentation occurs along sentence boundaries, and so cosine similarity is not as accurate given the smaller units of text. Cosine similarity is used to generate relative ranks for sentences, and these ranks are used as input to a divisive clustering algorithm which locates the boundaries between topics. Text segmentation can also be used to identify separate documents in a continuous stream of text, such as transcribed news reports. The methods for accomplishing this developed in (Beeferman, Berger, & Lafferty, 1999) are similar to our use of HMMs in that they both use probabilities conditioned on the sequence of tokens observed in a sliding window. The system from (Beeferman, Berger, & Lafferty, 1999) exploits certain regularities of news casts to develop prior probabilities for topic switches conditioned on trigrams and pairs of cue words.

Another area of related work concerns retrieving only the most relevant part of a structured document. This work applies to documents which are already broken into explicit chapters, sections, subsections, etc. and do not need to be automatically segmented. In (Wilkins, 1994) it is shown that applying relevance criteria to subsections of a document improves the performance of information retrieval on a corpus of long documents. In this scenario a user does not want her query to return a hundred page document when only one chapter is relevant. This work has been extended in (Lalmas, 1999) to cope with corpuses containing documents which are heterogeneous in their media types and languages by using more than one indexing vocabulary.

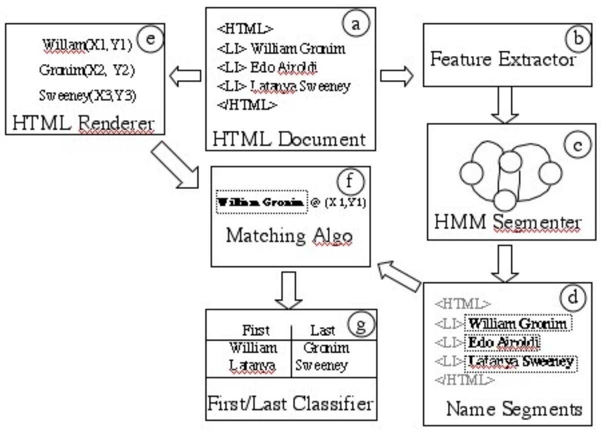


Figure 2: System architecture. (a) Raw HTML documents are input, (b) the feature extractor converts text to binary feature vectors, (c,d) the HMM segmenter extracts full names, (e) HTML is rendered as if in a browser, (f) name segments are matched to (X,Y) coordinates, (g) name tokens are classified as first or last names.

3 Classifier Design

3.1 Overview

Our name extraction system consists of several modules. The segmenter module is responsible for extracting full names from HTML source using an HMM. Unlike natural language NER names may be represented in several orders, for example ”William Gronim” and ”Gronim, William”. It is critical that our system recognize instances of one name from multiple documents as identical, regardless of ordering. In order to do this we use a first/last classifier that classifies each full name token extracted by the segmenter as either a first or last name. Names in any order can then be re-arranged into a common order for purposes of inter-document comparison. The first/last classifier requires tokens to be augmented with their (X,Y) coordinates as if the tokens were being displayed in a browser. An open-source HTML renderer, Multivalent (Phelps, 2006), is used to associate every text token in a document with its (X,Y) location. The full names produced by the segmenter are reconciled with the HTML renderer’s output using a matching algorithm. These geomertry-augmented tokens are then passed to the first/last classifier for final processing. This process is outlined in Figure 2, and described in detail below.

3.2 Feature Extractor

The feature extractor is tasked with converting text tokens into binary feature vectors. The HMM segmenter uses these feature vectors to learn conditional probability relationships between features and the names class. Documents are split into tokens around white space and punctuation. Every individual token is then presented as input to the feature extractor. The resulting feature vectors are then presented to the classifier, which classifies each individual token as either a name or not a name as described below.

3.3 HMM Segmenter

We use a Hidden Markov Model (HMM) (Rabiner, 1989) classifier to segment HTML source into ”name” and ”not-name” portions. The specific classifier used is the ”Voted Perceptron Hidden Markov Model” classifier. This classifier is available as part of the Minorthird text learning software (Cohen, 2006). All of the tokens inside names were extracted by hand and placed in the full name class. These name tokens were also placed in one of the first or last name classes, as appropriate. The document corpus is described in more detail in section 4, materials. We include titles and punctuation in a full name, and consider middle names to be first names.

We developed the set of features used by this classifier gradually. Some features, such as membership in a dictionary of names, are obviously of value to the classifier. We attempted to find others by measuring the information gain of each token, though this failed to identify any significant features. The most fruitful features were found by observing the errors our early classifiers made.

We have identified several general types of features useful to segmenting structured documents. Here a ”structured document” is a document with formatting and other information accompanying the text, as opposed to a document consisting of only natural language text. The first type of features identified are internal structure features. These describe the form of the extraction targets, for example ”names are capitalized”. Then there are ”doppleganger-pruning” features intended to identify text similar in form to extraction targets that should not be extracted. Our segmenter uses one of these features to reject email addresses which contain proper names. Finally we have structural location features, which identify the regions of a document in which extraction targets may appear. Our segmenter uses one of these to recognize that names do not appear within HTML tags. A summary of the features used by the first stage classifier and their types are included in Table 1. In Table 2 examples of each feature are given, with the tokens having that feature italicized. Each feature is described in detail below.

Feature Name	Feature Description	Feature Type
DictLast	The token is in the last name dictionary.	Internal form
DictFirst	The token is in the first name dictionary.	Internal form
CharTypePattern	The pattern of capitalization of the token.	Internal form
EmailAddr	The token is part of an email address.	Doppleganger-pruning
CommonCaps	Non-name tokens which were frequently capitalized.	Doppleganger-pruning
SingleWordTag	Single word HTML tags.	Doppleganger-pruning
InsideTag0	Tokens outside of HTML tags	Structural location
InsideTag1	Tokens inside of HTML tags	Structural location
FourToksAfterLI	Tokens occuring after <LI> tags.	Structural location
UserURL	URLs of personal sites	Structural location

Table 1: Features and their purposes

Feature Name	Example
CommonCaps	*Junior, University, Students*
CharTypePattern	John ⇒ X+x+, faculty ⇒ x+
DictLast	<td>John *Smith* - J.*Smith*@cmu.edu</td>
DictFirst	<td>*John* Smith - J.Smith@cmu.edu</td>
EmailAddr	<td>John Smith - *J.Smith@cmu.edu*</td>
FourToksAfterLI	<LI>*John Smith 412 555* 0000 J.Smith@cmu.edu
InsideTag0	<td>*John Smith - J.Smith@cmu.edu*</td>
InsideTag1	<*td*>John Smith - J.Smith@cmu.edu</*td*>
SingleWordTag	$< TD >, < LI >, < P >$
UserURL	<a href="/people/jsmith/">*John Smith*</a>

Table 2: Features and examples

The most obvious feature of significance to our segmenter is dictionary membership. We used two pairs of dictionaries, each pair consisting of a last name dictionary and a first name dictionary. The first pair was obtained from the Social Security Death Index (SSDI) as explained in the materials section. We also had smaller first and last name dictionaries developed during previous work on rosters, the ICU project (Malin et al., 2003). The dictionaries are stored in lower case, and each token is converted to lower case before being tested for membership in the dictionary. Every token in a dictionary was assigned the 'DictLast' or 'DictFirst' feature depending on which dictionary it appears in. Some names appear in both dictionaries and thus receive both features. Neither the SSDI nor the ICU dictionaries included every name in our input data, particularly names from Asian cultures. Both pairs of dictionaries also included common words which were frequently not names, such as "Good".

The second obvious feature is whether a token is inside or outside an HTML tag. Every token between a '<' and the first occurrence of '>' is assigned the "InsideTag1" feature, while all other tokens are assigned the "InsideTag0" feature. Though some names appeared inside HTML tags these are not the names which visually appear on the page. For example, the webmaster's name frequently appears in an HTML comment tag. Since none of the hand labeled names appear inside a tag, the "InsideTag1" feature has a very strong negative association with the name class.

The doppleganger-pruning features are included for the purpose of improving precision by discouraging the segmenter from extracting certain common name-like strings which are not in fact names. The first of these features is the "EmailAddr" feature, which is assigned to any token which is part of an email address. This prevents the classifier from classifying any part of "John.Smith@cmu.edu" as a name. Email addresses are detected in the HTML source using a straightforward regular expression. This regular expression matches tokens consisting of alphanumeric characters, ".", "+", "-", or "_" on either side of an "@" symbol. The second such feature is the "SingleWordTag" feature, which is assigned to any single all-alphabetic token between "<" and ">". For example, "Li" is a common last name, but we do not want to classify "<LI>" as a name.

During the design phase our classifier was found to perform much better on table structured rosters than on list structured rosters. When HTML tables are used to represent lists of names the names tended to be separated from accompanying information by HTML tags. However, when HTML lists are used, the names, phone numbers, emails, etc. are presented one after the other as free text. To improve performance on HTML list based rosters we added the "FourToksAfterLI" feature, which is assigned to the first four tokens after a '<LI>' tag which are not HTML comments. In all the list structured documents we encountered the individuals name was listed first, optionally followed by associated information.

Many rosters contain a link to an individual's home page. Every example of this in our document corpus included the individual's name inside the link text. In order to take advantage of this we use the UserURL feature. We only consider URLs containing one of the strings "∼" "people", "home", or "user" to be a UserURL. This is an exhaustive list of home page identifiers for our data, though others are possible and could be added to the feature extractor easily.

The last feature used by our segmenter is the capitalization pattern of a token. One or more capital letters is represented by "X+", and one or more lower case letters is represented by "x+". So, "John" would have

Input: NAMES: an array of extracted name tokens, sorted in increasing order of character offset in the HTML source; RENDERED: an array of tokens augmented with (X,Y) coordinates, sorted first by left to right parse tree traversal and then on X.
Output: NAMES′: NAMES augmented with (X,Y) coordinates.

```
1: $j \leftarrow 1$ //Point to the first element of NAMES
2: NAMES′ $\leftarrow$ copy(NAMES)
3: **for** $i = 1$ to the length of RENDERED **do**
4:    **if** RENDERED$[i]$ ==NAMES$[j]$ **then**
5:       augment NAMES′$[j]$ with coordinates from RENDERED$[i]$
6:       $j \leftarrow j + 1$
7:    **end if**
8:    $i \leftarrow i + 1$
9: **end for**
```

Figure 3: Pseudo-code for the matching algorithm responsible for augmenting extracted names with their (X,Y) coordinates.

the capitalization pattern ”X+x+”. Applying this to every token causes recurring false positives, because there are some common words in our input which share the capitalization pattern of names. To correct this we compiled a list of the 100 most frequent tokens of the ”X+x+” capitalization pattern and removed those from consideration for this feature. The makeup of this list is certainly related to the subject matter of our rosters, in this case university students. The common capitals list would probably need to be adapted to rosters listing different sorts of people (for example, military personnel might need ”Major”).

3.4 HTML Renderer and Matching Algorithm

The first/last classifier relies on text tokens being associated with their (x,y) coordinates in the rendered version of the document. The HTML rendering package used, Multivalent (Phelps, 2006), provides access to the text elements of a rendered document in terms of their (x,y) coordinates. On the other hand, Minorthird provides access to the text elements based on their character offset within the HTML source. We developed an algorithm to reconcile these alternate representations of the document. We observed that the relative ordering of two text tokens under these two representations is the same. When representing a list in HTML sequential elements of the list appear sequentially in the HTML source. Our algorithm for associating text tokens with (x,y) coordinates takes as input two arrays of text tokens, RENDERED and NAMES. Pseudo-code for this algorithm is presented in Figure 3. RENDERED is obtained from Multivalent and sorted first according to left to right traversal of the parse tree (to sort rows) and then the x coordinate in the rendered document (to sort within rows). We have found empirically that the parse tree preserves geometric ordering in the vertical dimension well. Intuitively, the sequential nature of the lists we are searching for makes for properly ordered parse tree elements. that NAMES is the output of our segmenter and is sorted by character offset in the HTML source. The array obtained from the HTML renderer contains all of the text which would appear in the rendered document, while the array sorted on character offset contains only the text we are interested in. Our algorithm iterates through the array obtained from the HTML renderer, comparing each element encountered to the next element of the array which is sorted on character offset. When a match is encountered the text token is labeled with its (x,y) coordinate and the next element of the array sorted by character offset is examined in turn. This algorithm terminates when it reaches the end of the array sorted by character offset.

3.5 First/Last Classifier

The output of the first-stage HMM is a set of name spans that are scattered across the rendered HTML page. A human can easily recognize the pattern of first and last names. How exactly does it work? We gather insight about the underlying human reasoning by pretending to observe a set of first and last names written in an alien dialect. With the help of a dictionary of common names, we could start by labeling all the first names that we find in the dictionary with ones, and the remaining tokens with zeros. Looking at the document again, we can see that there are more ones than zeros in some columns. Our best guess is to take all the tokens in columns with "significantly" more ones than zeros as first names, and all the remaining tokens as last names.

Following this intuition, in the second stage, we build a simple test that will suggest to us the final classification of name spans into first names and last names. The data we use to perform the test are: a dictionary of common names, the output of the first-stage HMM, and simple statistical reasoning—to formally quantify what it means to measure "significantly" more ones than zeros. The outcome is a decision on whether each token identified in the first stage HMM as part of the name span is a first name or a last name.

We rearrange the name spans identified by the HMM into a matrix X of n rows and m columns in order to abstract away the specifics of a given document's formatting. Here rows correspond to changes in the Y coordinate of the bottom left coordinate of a token's bounding box. The bounding box is a minimal rectangular region of pixels that would entirely contain the token if the document were rendered in a browser.

Bounding boxes are produced by Multivalent. The bottom left coordinate is chosen because different font sizes would create spurious rows if the top of the bounding box was used. Columns are token offsets within a row - column i contains the ith token on every row, regardless of the visual position on the row. We associate a variable F_{xy} with each position in this matrix, such that $F_{xy} = 1$ if the token it contains is found in the dictionary of common first names, and $F_{xy} = 0$ otherwise. We also create a variable L_{xy} for every element of the matrix, defined identically to F_{xy} except for its use of a dictionary of last names. In both sets of variables the first subscript (”x”) indexes columns (as if it were taken from the x-axis) and the second (”y”) indexes rows (as if it were taken from the y-axis).

A first way to go about building a test is to recognize that each $SF_x = \sum_y F_{xy}$ for $1 \leq x \leq m$ is distributed as a Binomial random number. In fact, under the hypothesis that the common names are scattered at random across the matrix with equal probability p, the F_{xy} are independent Bernoulli trials. Then we would use the observed $SF_{1:m}$ to estimate p, and use this Binomial model with parameters $\hat{p}$ and n to assess the statistical significance. There are several problems with this strategy; the main concerns we have are that there are typically few columns and the estimation of p is poor with few SF_x observations, and the Binomial model is clearly non justifiable to any rate.

Instead, we adopt a more empirically based strategy, which is non-parametric in spirit as we do not assume a parametric form for the distribution of the observed SF_x variables. Specifically, we build the histogram of $\{SF_x\}$, a widely used non-parametric density estimator, and we choose the median of the $\{SF_x\}$ as a threshold to decide when the amount of ones in a column is “significantly” more than what we would expect to observe if first names tokens were scattered at random across the rendered HTML page. The empirical version of the test we choose to perform is more robust for several reasons, in particular because of the good properties of the histogram as a density estimator in situations where a small sample of observations is available.

We implement this test directly, as shown in the pseudo-code in Figure 4. We summarize the variables used in Table 3. We define $SL_x = \sum_y L_{xy}$ for $1 \leq x \leq m$, identically to the SF_x variables except that they are defined in terms of last names. On lines 1-6 we construct the variables described above and in Table 3. Line 7 calculates the column index with the highest occurrence of first names. This purely heuristic measure eliminates a problem encountered while designing this algorithm. We must choose either first or last name labels first, but we don’t want our first test to overwrite the column that is most probably of the second

Variable Name	Description
X	An n by m matrix of name tokens
F_{xy}	1 if the xth token on the yth row is in the first name dictionary, 0 otherwise.
L_{xy}	1 if the xth token on the yth row is in the last name dictionary, 0 otherwise.
SF_x/n	The mean of all F_{xy} - an estimate of the probability that a token in the xth column is a first name.
SL_x/n	The mean of all L_{xy} - an estimate of the probability that a token in the xth column is a last name.
M_F	The median of SF_x/n over all x - used as a threshold.
M_L	The median of SL_x/n over all x - used as a threshold.
MaxIdxF	The column having the greatest total number of tokens in the first name dictionary over all rows.

Table 3: Variables used by the first/last classifier pseudo-code and their descriptions.

Input: The n by m matrix X of extracted tokens; The variables F_{xy} and L_{xy}
Output: Vector D of length m, containing a label for every column of X.

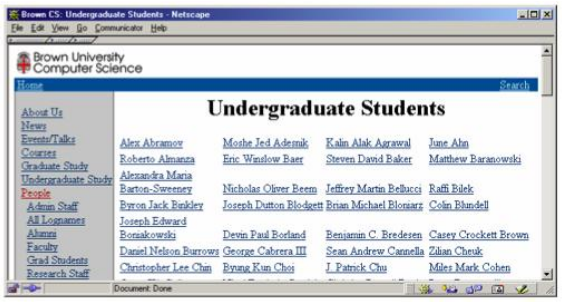
```
1: for $x = 1 : m$ do
2:     $SF_x \leftarrow sum_y(F_{xy})$
3:     $SL_x \leftarrow sum_y(L_{xy})$
4: end for
5: $M_F \leftarrow$ the median of all $SF_x/n$
6: $M_L \leftarrow$ the median of all $SL_x/n$
7: MaxIdxF $\leftarrow argmax_x(SF_x)$
8: for $x = 1 : m$ do
9:     if $((SL_x/n) > M_L$ and $x \neq$ MaxIdxF$)$ then
10:        $D_x$ = LAST
11:    else if $((SF_x/n) > M_F)$ then
12:        $D_x$ =FIRST
13:    else
14:        $D_x =$ NEITHER
15:    end if
16: end for
```

Figure 4: Pseudo-code for the first/last classifier.

type tested. On lines 9-14 we apply our test to every column, taking into account the heuristic variable MaxIdxF. Some examples of columns justifying a ”NEITHER” label are those containing ”,” in the ”Last, First” ordering and those containing titles.

4 Materials

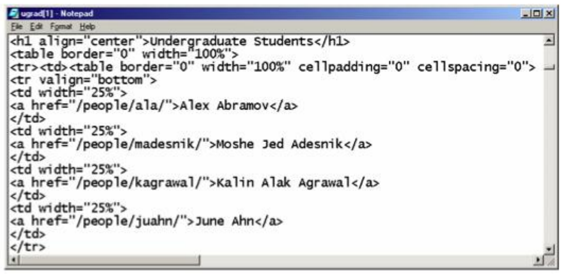
The document corpus used to develop out system consists of 37 HTML documents obtained from university websites, each of which primarily lists the names of undergraduate students. These documents were obtained during the ICU project (Malin et al., 2003) using the RosterFinder program. All of these docu-



Undergraduate Students

Alex Abramov	Moshe Jed Adesnik	Kalin Alak Agrawal	June Ahn
Roberto Almanza	Eric Winslow Baer	Steven David Baker	Matthew Baranowski
Alexandra Maria Barton-Sweeney	Nicholas Oliver Beem	Jeffrey Martin Bellucci	Raffi Bilek
Byron Jack Binkley	Joseph Dutton Blodgett	Brian Michael Bloniarz	Colin Blundell
Joseph Edward Boniakowski	Devin Paul Borland	Benjamin C. Bredesen	Casey Crockett Brown
Daniel Nelson Burrows	George Cabrera III	Sean Andrew Cannella	Zilian Cheuk
Christopher Lee Chin	Byung Kun Choi	J. Patrick Chu	Miles Mark Cohen

(a) Formatted roster listing Brown undergraduates in a browser. Taken from (Sweeney, 2004).



```
<h1 align="center">Undergraduate Students</h1>
<table border="0" width="100%">
<tr><td><table border="0" width="100%" cellpadding="0" cellspacing="0">
<tr valign="bottom">
<td width="25%">
<a href="/people/ala/">Alex Abramov</a>
</td>
<td width="25%">
<a href="/people/madesnik/">Moshe Jed Adesnik</a>
</td>
<td width="25%">
<a href="/people/kagrawal/">Kalin Alak Agrawal</a>
</td>
<td width="25%">
<a href="/people/juahn/">June Ahn</a>
</td>
</tr>
```

(b) Formatted roster listing Brown undergraduates in a browser. Taken from (Sweeney, 2004).

Figure 5: Formatted and unformatted views of a roster.

ments represented lists of names as either HTML tables or HTML lists. The documents ranged in size from 4K to 404K. Some contained less than 10 names, while others contained as many as 3,000 names. Some groups of documents were from the same school, for example UC Berkeley's student rosters are split into documents according to the first letter of the students' last names. There were a total of 21 schools represented in the data. Within each school the pattern of HTML tags used to represent the lists was frequently but not always similar. In designing our first/last classifier we made the assumption that each document would choose a consistent ordering for the first and last names. Only one document in our corpus violates this assumption. The Brigham Young University roster presents names with several different capitalization and ordering schemes. It is difficult for a human to decide which components of a given name are which. Nonetheless we include this roster in our accuracy experiments, since it is real world data that we wish to utilize. The classifier was not told which documents came from which schools. Figures 5(a) and 5(b) show the formatted and unformatted views of a representative roster.

We used the Social Security Death Index (SSDI) to build large but not comprehensive dictionaries of first and last names. The SSDI records the name and social security number of everyone who has ever died while holding a social security number in the United States, for the purposes of preventing fraud by identity theft. These dictionaries included approximately 420,000 first names and 1.27 million last names. The dictionaries used in the ICU project consisted of approximately 4800 first names and 18,600 last names. In both pairs of dictionaries the first and last name dictionaries overlapped. Our system was not affected by out choice of dictionary, probably because the smaller dictionaries contained a large portion of the names in our data. We use the larger, SSDI dictionaries in practice because the performance difference is undetectable and the larger dictionaries should be more useful for rosters drawn from U.S. residents.

5 Experiments

5.1 HMM Segmenter

The segmenters's performance on large documents was very poor. Using an already trained HMM to label all the tokens in a 400K document took approximately 5 hours on a 3GHz Pentium desktop. We determined the bottle neck to be the Viterbi dynamic programming algorithm used by the HMM to find the highest scoring name/not name classification for each token in a sequence of tokens. The Viterbi algorithm's time complexity is super-linear on the length of the input sequence, and our larger training documents were hundreds of thousands of tokens long. Attempts to optimize the Java implementation itself failed to produce significant results. Splitting the input documents into smaller, fixed-size chunks of 1,000 tokens each improved the run time from 5 hours to about 2 minutes. We found in preliminary experiments that splitting the input into 1,000 tokien chunks did not affect accuracy. The document chunks are reassembled into the original HTML before being presented to the first/last classifier. Since the annotations added to the HTML source by the segmenter are represented in terms of character offsets it is straightforward to map annotations over chunks back to annotations over whole documents.

We performed a 10-fold cross validation experiment in order to evaluate our HMM segmenter. Each fold produced a different classifier. All documents were split into chunks of 1,000 tokens after being placed in either the training or testing partition (no document had chunks present in both the training and testing partitions). Running a 10-fold cross validation allowed us to observe the segmenter's overall performance while still examining the effects of differently structured documents. For example, some of the documents in our corpus use HTML lists while others use tables. If the training partition contained only documents using tables and the testing partition uses only documents using lists then performance will suffer. This experimental design also allows us to pinpoint testing partitions containing documents with which our system has difficulty, as discussed in the first/last classifier experiments below.

The average precision and recall values across all 10 folds were 95% and 91%, respectively. The standard deviation for precision was 0.07, and the standard deviation for recall was 0.08. We also plot the Receiver Operating Characteristic (ROC) for the segmenter in Figure 6. This plot incorporates the low

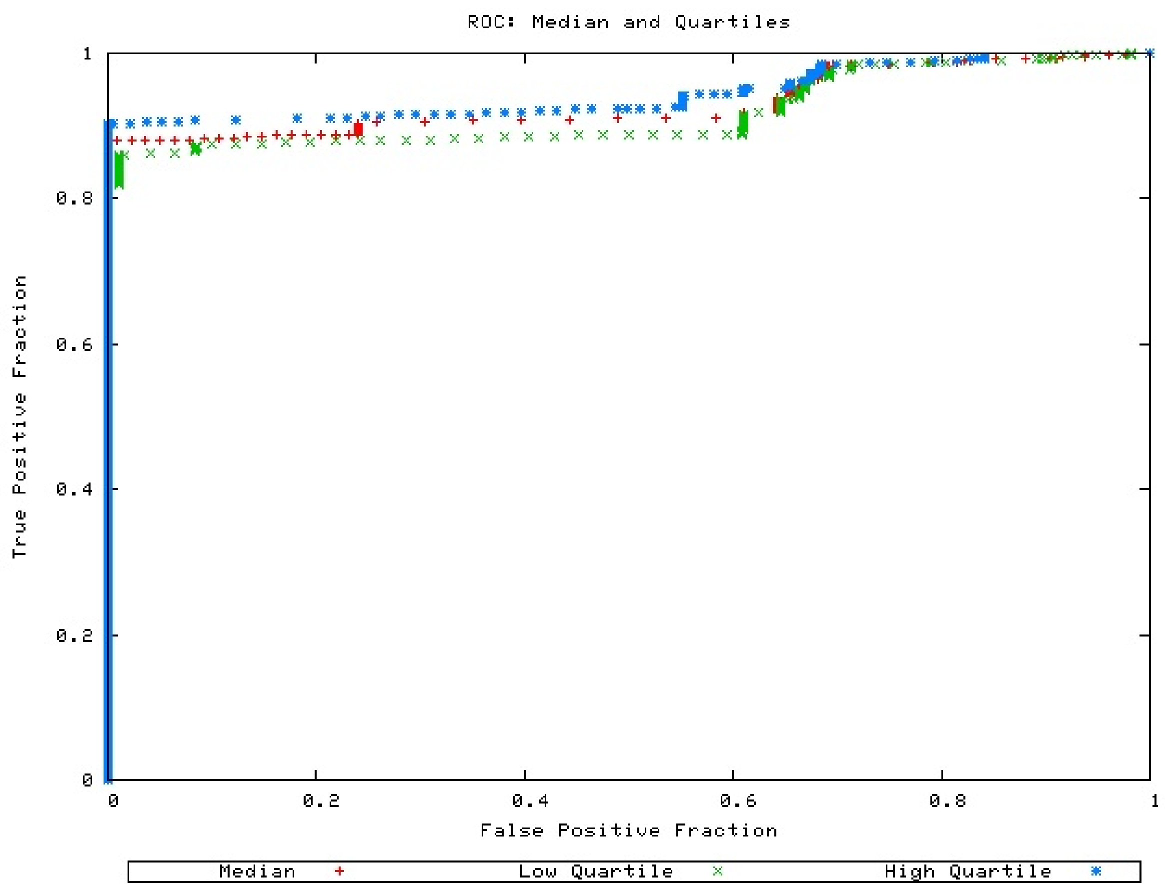


Figure 6: ROC for HMM segmenter over 10 folds. The low quartile, median, and high quartile results are plotted with AUC of 0.918, 0.931, and 0.944, respectively.

quartile, median, and high quartile across our 10 folds. These 3 plots are very close together. The Area Under Curve (AUC) is 0.918, 0.931, and 0.944 for the low quartile, median, and high quartile, respectively. Based on our average precision and recall values and median AUC our HMM segmenter's performance is excellent. The low standard deviation and tight quartiles indicate that this performance is consistent across our document corpus.

5.2 First/Last Classifier

In order to evaluate our first/last classifier we used the 10 separate outputs from our segmenter experiment as inputs for our second stage. Each testing partition labeled with full names by the first stage was used as an input to our second stage. We measured overall precision and recall over first and last names jointly. Each testing partition's contribution to the average performance was weighted according to the number of predicted names produced by the segmenter. The first/last classifier was scored against true first and last names. Thus the first/last classifier's theoretical maximum score on a given testing partition would be the score of the segmenter on that partition. Though scoring the first/last classifier against the ”truth” predicted by the segmenter would increase the first/last classifier's score this is less realistic, since the two are run in sequence in actual operation.

The weighted average precision and recall values found by this experiment were 84% and 82% respectively. The standard deviations were 0.32 for precision and 0.31 for recall. We examined the trials that performed poorly and determined that they had an abundance of short rosters. On a short roster our first/last classifier has less information about the distribution of name types across token positions, resulting in worse performance. This systematic flaw in the first/last classifier is being examined in ongoing work, as discussed below.

6 Discussion and Future Work

We have extended previous NER work to a new problem domain, rosters. Rosters present several new challenges that our NER system overcomes. Names are not embedded within a natural language grammar, denying us a rich set of part of speech based features. The use of several orderings of first and last names when listing names requires us to recognize these orderings, a problem that does not arise in natural language NER. Our solution to these problems combines HMM techniques leveraging existing NER work with geometric techniques operating on the rendered representation of documents. Though our HMM performs very well on our entire corpus our first/last classifier's performance is consistently poor on short rosters and therefor highly variable.

The first/last classifier makes only limited use of the rendered representation of documents. Our HTML renderer provides extensive information, including the bounding box of every token and a parse tree. Of this information we utilize only a single coordinate per token, and then only to detect rows of text. In ongoing work we are exploring the potential for using the entire bounding box of each token. Specifically, we are attempting to segment the coordinate space of the rendered document into regions containing names and regions not containing names. By doing so we hope to eliminate large portions of the document early, decreasing the noise level and overall size of the input to our classifiers in order to improve accuracy and performance. We hope to overcome the difficulties posed by name orderings and lack of natural grammar by more fully exploiting the rendered representation.

Since beginning this work we have targeted a dozen more universities in order to generate more real world data for our system. We downloaded approximately 100,000 candidate documents and identified hundreds of rosters. Drastically expanding the size of our data set has revealed new issues not fully addressed in this paper. Based on this data we have discovered that RosterFinder introduces significant error into the system by mis-classifying non-rosters as rosters. These non-rosters do not conform to our assumptions about roster formats, and thus our segmenter performs very poorly on these documents. In order to remedy this we are evaluating improvements to RosterFinder such as evaluating sections of a document independently. We hope to reduce the noise level by, for example, detecting that the main content section of a page is a roster and that the navigation bar section is not. Though many of the true rosters found do match our assumptions many do not. In ongoing work we are adapting our segmenter to be more flexible. Specifically, we are exploring making greater use of the parse tree in order to create more stable features for identifying lists.

Rosters are a novel source of information for social network analysis and the mining of competitive information. Social networks derived from rosters have some unique properties ripe for study. The co-location of many names on a single rosters introduces a clique into the social network. The impact of so many cliques on the structure of mined social networks should be explored. Mining rosters provides universities and other organizations the ability to automatically assess their compliance with polices on the disclosure of personal information. These techniques also present competitive risks, for example giving one university the ability to automatically compare their admissions decisions to those of another university. These risks should be quantified in a specific case study.

Acknowledgements

The authors wish to thank William Cohen for developing and providing help with his Minorthird text learning software. We would also like to thank the other members of the Data Privacy Laboratory for helpful discussions and comments. This research was supported by the Data Privacy Laboratory in the Institute for Software Research International in the School of Computer Science at Carnegie Mellon University.

References

[1] Adamic, L., Adar, E. (2001).

Friends and Neighbors on the Web. Xerox PARC, 2001. http://www.parc.xerox.com/istl/groups/iea/papers/web10/

[2] Adelberg, B., Denny, M. (1999).

Building Robust Wrappers for Text Sources. Chicago: Department of Computer Science, Northwestern University, Technical Report. http://citeseer.ist.psu.edu/adelberg99building.html

[3] Beeferman, D., Berger, A., & Lafferty, J. (1999).

Statistical Models for Text Segmentation. Machine Learning Vol. 34, No. 1, 1999. Association of Computing Machinery. http://citeseer.ist.psu.edu/beeferman99statistical.html

[4] Bikel D.M., Schwartz, R., & Weischedel, R.M. (1999).

An Algorithm that Learns What's in a Name. Machine Learning (Special Issue on NLP), 1999. 34(3). 211-231.

[5] Brants, T., Chen, F., & Tsochantaridis, I. (2002).

Topic-Based Document Segmentation with Probabilistic Latent Semantic Analysis. In Proceedings of the Eleventh International Conference on Information and Knowledge Management. Mclean, VS, USA, November 2002. Association for Computing Machinery.

[6] Chei, H.L. & Ng, H.T. (2002).

Named Entity Recognition: A Maximum Entropy Approach Using Global Information. Proceedings of the Nineteenth International Conference on Computational Linguistics (2002), 190-196. Taipei, Taiwan.

[7] Choi, F.Y.Y. (2000).

Advances in Domain Independent Linear Text Segmentation. In Proceedings of NAACL'00, Seattle, USA, April 2000. ACL. http://citeseer.ist.psu.edu/choi00advances.html

[8] Chung, C.Y., Gertz, M., & Sundaresan, N. (2002)

Reverse Engineering for Web Data: From visual to semantic structures. In ICDE, 2002. http://citeseer.ist.psu.edu/chung02reverse.html

[9] Cohen, W.W. (2000).

Automatically Extracting Features for Concept Learning from the Web. In Proceedings of the Seventeenth International Conference on Machine Learning, 2000.

[10] Cohen, W.W. (2006).

Minorthird: Methods for Identifying Names and Ontological Relations in Text using Heuristics for Inducing Regularities from Data, http://minorthird.sourceforge.net

[11] Family Educational Rights and Privacy Act. 33 CFR Part 99. http://www.ed.gov/policy/gen/guid/fpco/ferpa/index.html

[12] Ferdinand, P. and Barbaro, M. 2002.

Yale Tells FBI of Rival's Breach of Web Site Princeton Suspends Admissions Official Over Snooping Into Student Files. Washington Post. July 26, 2002; p A02. http://www.washingtonpost.com/ac2/wp-dyn?pagename=article&node=&contentId=A2983-2002Jul25&notFound=true

[13] Google API. 2004. http://www.google.com/apis/

[14] Hearst, M.. 1997.

TextTiling: Segmenting Text into Multi-Paragraph Subtopic Passages. Computational Linguistics, 23 (1), pp. 33-64, March 1997

[15] Hofmann, T.. 1999.

Probabilistic latent semantic analysis. In Proceedings of the 15th Conference on Uncertainty in AI, 1999. http://citeseer.ist.psu.edu/hofmann99probabilistic.html

[16] Isozaki, H., & Kazawa, H. 2002.

Efficient Support Vector Classifiers for Named Entity Recognition. Proceedings of COLING-2002, 2002.

[17] Kautz, H., Selman, B., & Shah, M. 1997.

Referral web: Combining social networks and collaborative filtering. Communications of the ACM, 40(3), 63-65.

[18] Kazuhiro, S., & Mostafa, J. 2003.

An approach to protein name extraction using heuristics and a dictionary. Laboratory of Applied Information Research Tech Report 2003-2. Indiana University, Bloomington, IN, USA, 2003.

[19] Klein, D., Smarr, J., Nguyen, H., & Manning, C.D. 2003.

Named Entity Recognition with Character Level Models. Seventh Conference on Computational Natural Language Learning, 2003.

[20] Lalmas, M. 1999.

A Model For Representing and Retrieving Heterogeneous Structured Documents Based on Evidential Reasoning. The Computer Journal, Vol. 42, No. 7, 1999.

[21] Malin, B. et al. 2003.

Identifying Computer Science Undergraduates. http://privacy.cs.cmu.edu/dataprivacy/projects/ICU/index.html

[22] Phelps, T. 2006.

Multivalent: Digital documents research and development. http://multivalent.sourceforge.net/

[23] Rabiner, L.R. 1989. *A tutorial on HMM and selected applications in speech recognition.* In Proc. IEEE, Vol. 77, No. 2, pp. 257–286, Feb. 1989

[24] Sweeney, L. 2004.

Finding Lists of People on the Web. ACM Computers and Society, 34 (1) April 2004.

[25] Wilkinson, R. 1994.

Effective retrieval of structured documents. In Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Dublin, Ireland, July 1994. Association for Computing Machinery. http://citeseer.ist.psu.edu/wilkinson94effective.html