HIGH ACCURACY POSTAL ADDRESS EXTRACTION FROM WEB PAGES

by

Zheyuan Yu

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
March 2007

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "HIGH ACCURACY POSTAL ADDRESS EXTRACTION FROM WEB PAGES" by Zheyuan Yu in partial fulfillment of the requirements for the degree of Master of Computer Science.

Dated: March 30, 2007

Supervisors: _____
Dr. Evangelos E. Milios

_____
Dr. Vlado Keselj

Reader: _____
Dr. Qigang Gao

DALHOUSIE UNIVERSITY

DATE: March 30, 2007

AUTHOR: Zheyuan Yu

TITLE: HIGH ACCURACY POSTAL ADDRESS EXTRACTION FROM
WEB PAGES

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: M.C.Sc.   CONVOCATION: May   YEAR: 2007

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

_____

Signature of Author

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than brief excerpts requiring only proper acknowledgement in scholarly writing) and that all such use is clearly acknowledged.

Table of Contents

List of Tables

List of Figures

Abstract

Automatically extracting geographic information from Web pages, and analyzing such information can greatly benefit data mining and information retrieval systems. One most obvious but valuable source of geographic information is postal address.

In this thesis we present three methods for postal address extraction - rule-based, machine-learning and hybrid. Our machine-learning based system combines different sources of weak evidence and uses the word n-gram model as the underlying representation of web pages. The extracting process is fast and accurate despite not using dictionary look-up. It outperforms both rule-based systems that we built, with precision of 94.3% and recall of 71.6%.

We also proposed a hybrid method which combines the rule-based method and the machine-learning approach. The accuracy of the system is further improved with precision of 95.2% and recall of 81.1%.

Acknowledgements

With a deep sense of gratitude, I wish to express my sincere thanks to both of my supervisors, Dr. Evangelos E. Milios and Dr. Vlado Keselj, for their patient guidance and their immense help on my thesis. I would like to thank Dr. Qigang Gao for his valuable time as my thesis reader and his suggestions on the final version of the thesis.

Many thanks to Tony Abou-Assaleh for his inside comments and suggestions.

Special thanks to my wife, Miao Yu, who has played a critical role in the completion of this thesis: without her love and encouragement, this thesis would never have been possible.

Chapter 1

Introduction

The World Wide Web contains a wealth of geographical information, such as postal addresses, telephone numbers, place names, geographic contexts derived from content or hyperlinks, even networking routing information and the URL or IP address of the web server [25, 22]. The ability to automatically extract geographic information from web pages, and to analyze such information, can greatly benefit data mining and information retrieval systems. Examples of location-based applications include geospatial mapping of web pages for easy navigation [25], spatial analysis of news sources [23], geospatial search engines [12] to find local restaurants or real estate listings, and tools that help business planners to investigate an area before investing by searching documents for geographically referenced information[1].

1.1 Motivation

One of the most obvious sources of geographic information is the postal address, which can be more precisely mapped into latitude and longitude. Other geographical references, such as city names, are usually mapped into more broadly shaped regions. Information contained in postal addresses, such as street name, city, state/province, country and postal code/ZIP, can be analyzed to classify and index web pages [9, 11, 12], or find the correlation between web pages that contain those addresses and cluster them into different region classes to facilitate user navigation [25, 12].

The problem of address extraction seems simple: Given a data set that contains full list of postal addresses, search the web pages for addresses in the list by string matching. Such data sets are available in more technically advanced countries, and are usually maintained by postal services, governments or commercial data vendors. For example, the U.S. Census Bureau maintains TIGER (Topologically Integrated and

---

[1]http://www.metacarta.com

Geographic Encoding and Referencing) dataset[2], which is designed to be a complete and current list of all addresses and locations, with map coordinates, covering an estimated 115 million residences as well as 60 million businesses and other structures in the U.S. The latitude and longitude can be related to a ZIP+4 coded address by combining TIGER with Tiger/ZIP+4 data set [3], which contains over 33 million data records from more than 27,000 5-digit ZIP codes and provided by the United States Postal Service (USPS).

However, there are a number of factors that make highly accurate address extraction difficult.

First, the same address may be written in different formats because of individual preference or other considerations. One or more elements in the address may be missing or be written in different orders. It may also have some extra text, such as name of recipient, building name, or indicate corporate routing information. The wide variation in abbreviations will also degrade the performance of extraction.

The problem is even worse with web pages, which are so unstructured that addresses are more likely to be presented in informal ways. Misspellings, typos, non-standard abbreviations, incorrectly joined elements, improperly ordered elements and other errors often occur. In our testing data set, for instance, MA, Mass., and M.A. are all used to represent the state of Massachusetts, even though the last one is a rarely used non-standard abbreviation. The ZIP is most often missing. If state and city names can be inferred from the context of the web pages or the web site, they are often omitted.

Second, the format of postal addresses varies greatly from one country to another. For example, in North America, usually the address will be written in the order of *street address*, *city*, *state* or *province*, *ZIP/postal code*, then *country*, while in China, country will be written first, or omitted, followed by *province* and *city*, then *street address* and *postal code*; Postal code formats also vary - the U.S. uses 5- or 9-digit numbers whereas postal codes in Canada and the U.K. are 6 alphanumeric characters.

Third, the ambiguity of place names also makes this task difficult, as many place names also have non-geographic meanings. As well, some common words frequently

---

[2]http://www.census.gov/geo/www/tiger
[3]http://www.usps.com/ncsc/addressmgmt/tiger.htm

used in English are also place names, for example, "main" is a common word frequently used in English, while "Main Street" is a common street name in many cities. The ambiguity may lead to the problem of spurious matches or false positives. Amitay et al [3] reported that the average number of possible meanings per potential geographic name on web pages is about 2.

Fourth, many countries do not have the data set of a full list of streets, because of the associated costs to maintain such data, or the lack of policies to create the geographic data set.

All the above factors make high-performance address extraction complicated. However, research in this area has been limited in spite of the challenges this problem offers and its increasing importance for location-based services and other applications.

The hypothesis that motivates the work described in this thesis can be stated as follows:

Word n-gram based machine-learning techniques can be applied to develop an effective system to extract postal addresses from web pages.

1.2 Objective and Contributions

Our objective is to provide techniques to automatically extract postal addresses from web content.

A classification method has been applied to the address extraction task. It uses the word n-gram model as the underlying representation of web pages, and learns from different sources of weak evidence.

In addition to the machine-learning approach, two rule-based methods were exploited and implemented as the baseline systems. A hybrid method which combines the rule-based and the machine-learning approaches was also proposed in the thesis.

Since there is no public data set available for evaluating address extraction task, we collected and labeled three web page data sets to provide more reliable evaluation. These data sets can be made publicly available to other researchers.

Systems based on the three different approaches, rule-based, machine-learning, and hybrid, have been evaluated and compared using the data sets that we collected. We demonstrated that the machine-learning approach to address extraction is more affective than both rule-based approaches, and the accuracy of the system was further

improved by using the hybrid approach.

1.3 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 presents an overview of two research areas that are related to the address extraction task, *Named Entity Recognition* and *GeoParsing/GeoCoding*. In Chapter 3, three methods that we used for address extraction tasks are presented, they are rule-based, machine-learning, and hybrid. This Chapter will also provide implementation details. The data used for all of the experiments in this thesis was collected by us. Chapter 4 will provide an overview of the way that the data was collected, and describe this data in greater detail. The experimental results will also be reported in this Chapter. Chapter 5 presents the summary, conclusions and some ideas for future research.

Chapter 2

Background and Related Work

The problem of high-performance address extraction is closely related to the areas of *Named Entity Extraction* and *Geoparsing/Geocoding*. In this chapter the two topics related to this research are reviewed.

2.1 Named Entity Recognition

Named Entity Recognition (NER) is the task of extracting entities, such as proper name (person, location and organization), time (date and time), and numerical values (currency and percentage), from the source text, and mapping them into predefined categories, such as *person*, *organization*, *location name* or *"none-of-the-above"* [7]. It is a part of the Information Extraction research domain aimed at extracting various information from unstructured text-based data sources. The task of *NER* was first introduced as part of the MUC-6 [13] and was continued in MUC-7(1998). The *NER* task defines seven types of named entities, and is standardized into three subtasks:

- ENAMEX: Names of organizations, persons and locations.

- TIMEX: Mentions of dates and times (relative and absolute).

- NUMEX: Direct mentions of currency/percentage.

Our problem of address extraction falls into the ENAMEX subtask: however, instead of extracting coarse-grained geographical locations, such as international regions, bodies of water, mountains, etc., we focus on recognition of postal addresses, which are finer-grained geographical entities.

In the past decade, many NER systems have been developed using rule-based or machine-learning methods, they are introduced in following subsections.

2.1.1 Rule-based Pattern Matching Methods

Rule-based methods are widely used in the early NER systems. These systems usually rely on carefully handcrafted rules or regular expressions and are often combined with looking up of specialized dictionaries, to extract names by applying pattern matching to the source text. FASTUS [17], an information extraction system based on finite state automata, was proposed by Appelt et al. It first segments sentences into noun groups, verb groups, and other phrases. The sequence of phrases is then scanned for patterns of interest, and corresponding incident structures, which will be merged into a complete template. It achieved the state-of-the art performance in 1993, with 44% recall and 55% precision. LaSIE-II system [15] matches the input text against pre-stored lists of organization names, company designator, titles, currency units, location names, and time expressions. It then uses Named Entity grammar, a set of rules produced by hand, to identify noun phrases. Other rule-based systems include the New York University Proteus system [38], RAPIER [10] and LTG [26].

2.1.2 Machine-learning Methods

In recent years, a variety of machine-learning based approaches have been used in NER systems.

The NYMBLE system built by Bikel et al [5] used a variant of the standard Hidden Markov Model (HMM) to extract names, which achieved 93% F-Measure score on a collection of newswire articles. A HMM learns a generative model over input sequence. It represents name entity classes as hidden states, and words as the observed symbols. The problem of NER is considered as the problem to find the state sequence that maximizes the probability of generating the given word sequence. Another HMM based Named Entity tagger was built by Zhou et al [39] to combine both internal (word, semantic and gazetteer) and external (context) evidences, and achieved the state-of-the art performance with F-measures of 96.9% and 94.3% on MUC-6 and MUC-7 English NE task respectively.

Sekine et al [32] applied the Decision Tree approach to Japanese NER task. Their system assigned $4n + 1$ tags to each input token, where n is the number of categories (for MUC-7, $n = 7$). For any particular category from the n categories, a token could be in one of 4 states: "start", "middle", "end", and "unique". In addition, a

token could be tagged as "other" to indicate that it is a not part of a named entity. Two consecutive tokens could be assigned incompatible tags. For instance, a tag sequence of [location_start, person_end] would be inconsistent. To solve this problem, the system kept the probabilities of the all potential tags for each token, instead of just recording the most probable tag. The Viterbi decoding algorithm [37] was used to search the most probable consistent path through the sentence.

Baluja et al [4] also used the Decision Tree approach to extract names. Character type, Part-of-Speech, punctuation, and dictionary information are used to train a decision tree to label tokens in the text. They achieved F-measure of 93.2% on 100 randomly-selected Reuters news articles. The Decision Tree approach has also been successfully used in biological domain [18].

Andrew Borthwick [7] proposed a Maximum Entropy (ME or MaxEnt) NER system. Similar to Sekine's system, it labeled each token as start, middle, end, and unique for every named entity category, then used Viterbi algorithm to discover the highest probability path in which there are no two tokens in which the second one cannot follow the first. Followings are the features used by the system:

- Binary character type feature, i.e., the word is capitalized.

- Lexical feature: tokens are compared with a vocabulary and the vocabulary indices are recorded.

- Section feature: which article section that the current token is in.

- Dictionary feature.

- External system feature that incorporates named entity output from other NER systems.

- Reference resolution feature that recognizes a name as being an alias for another name.

The system achieved F-measure of 86.56% on MUC-7 data set.

Recently, Conditional Random Fields (CRFs) [21] were applied to NER task [24]. CRFs are undirected graphical models which define the conditional probability of label sequences given a particular observation sequence. The conditional nature of

CRFs relaxes the independent assumptions required by HMM. The models have great flexibility to include a wide array of arbitrary, non-independent features. The method obtained overall F_1 of 84.04% on the CoNLL-2003 English news article test set. The main disadvantage of CRFs is the computational expense of training because of the need to perform inference repeatedly during training.

2.2   Geoparsing/Geocoding

In the field of Geographic Information Systems (GIS), the process of recognizing geographic context is referred to as *geoparsing*. Geoparsing can be considered as a part of Named Entity task, but it extracts more variety of geographic entities, such as *street, city, phone numbers, ZIP/postal codes, URL*, geographic contexts derived from content or hyperlinks, and *network routing information*. The process of assigning latitude and longitude to the extracted geographic context by the geoparsing process is called *geocoding* (also referred to as *grounding* or *localization*) [25]. Geoparsing and geocoding geographic contexts from web pages have been gaining attention recently [25, 3]. They proposed methods to assign each web page/site a geographic focus, then provide users the ability to navigate web resources by geospatial criteria.

Buyukokkten [9] proposed an approach to capture the geographical scope of the web resources by extracting geographic entities, area codes or ZIP/postal codes from web pages, and to identify the physical location of a web site by analyzing "whois" records and network routing traffic.

McCurley [25] investigated different sources of geographic contexts of a web page, including hosts, context from content, addresses and postal codes, telephone numbers, geographic feature names, context derived from hyperlinks and other potential sources. A prototype system for geographical navigation was implemented to browse web resources by geographic proximity. The system used a simple rule to recognize ZIP codes in web pages, then matched the ZIP code with a database of US postal codes to find the region that each ZIP code covered. The latitude and longitude values of the centroid of each region were then assigned to each web page.

Rauch et al [31] described a complete geography-based search system, being developed commercially by MetaCarta [1].

---

[1]http://www.metacarta.com

Google Local[2] locates neighbourhood stores and services by searching billions of pages across the web, then checks those results with Yellow Pages data to pinpoint the local resources.

The problem of address extraction is an important part of the geoparsing problem because postal addresses can be more precisely geocoded than other more general geographical names; therefore, they are much more valuable to the location-based systems. However, there is very little work on address extraction available in the literature. McCurley [25] used the Flex parser [27] to recognize ZIP codes in web pages, where a database of US ZIP codes is used to look up ZIP codes and map them into latitude and longitude values. A ZIP code is valid only if preceded immediately by a reference to the state name to prevent too many false positives. This approach assumes the ZIP code always follows the state name in addresses, which is not always the case for the unstructured web contents. Borka et al [6] applied the Hidden Markov Model to segment postal addresses into fine-grained fields, such as "*city*" and "*street name*" from existing address databases. The problem addressed in this thesis is different from theirs: our task is to extract address blocks from web pages whereas their task is to extract address elements (street name, city, state, etc.) from addresses database. Our systems extract and construct the address database used by them to extract from. All other existing research works that can be found in this area focussed on handwritten recognition of addresses from postal or facsimile images [19, 16, 33].

In the following chapter we introduce the problem definition, and present the different approaches that we adapted to the problem of address extraction.

---

[2]http://local.google.com

Chapter 3

Problem, Methodology and Implementation

3.1 Problem Definition

Our task is to extract postal addresses from web pages. An address extraction system is defined as a system that takes input from web pages, extracts and outputs addresses contained in those web pages.

During the extraction process, the system should pre-process the web pages into tokens, and output them in sequence with each token labeled as one of four classes: *START*, *MIDDLE*, *END* and *OTHER*, which can be used for system performance evaluation or for integrating with other systems. The four labels mean the labeled token occurs in an address text block as first, middle or last token, or not belonging to any address respectively.

The following sections in this chapter describe four systems that we built for address extraction which use *Rule-based*, *Machine-learning* based and *Hybrid* methods.

For the rule-based method, two systems were implemented with the *Regular Expression* approach and the *Gazetteer* approach, respectively, and are introduced in Section 3.2.1 and Section 3.2.2. The machine-learning based system is presented in Section 3.3, and the hybrid system is illustrated in Section 3.4.

3.2 Rule-Based Extraction Method

Despite of the variety of expressions and formats which may be used in addresses, a great number of US addresses showing up on the web follow the format starting with *Street number* followed by *Street name*, *City*, *State Name*, optional *ZIP code* and *Country Name*. A number that occurs in the text could potentially be a street number; therefore, it is a valuable cue for the start of a potential address, while state name and ZIP code are indicators of the end of an address. With both the starting and ending positions of an address found, an address block can be easily extracted.

By applying the above observations, we built two extraction systems. They use different approaches to identifying address elements, such as *street name*, *ZIP code*, and *address boundaries*. One uses regular expressions, the other the Gazetteer approach, as described in Section 3.2.1 and Section 3.2.2.

3.2.1   Regular Expression Approach

To extract addresses from web pages, the regular expression approach requires five address elements to be identified in order to find the boundaries of address text blocks. It applies pattern matching to find the following address elements: *PO Box*, *Street Number*, *State*, *ZIP code*, and *Country*. Among these elements, the first two are used to find the starting boundary of an address, the others are used to find the ending boundary.

Rules Used

The following heuristic rules are used to match address elements and extract addresses.

1. Numbers in the web page are identified as potential street or PO Box numbers.

2. All 5- or 9-digit numbers that are immediately preceded by a state name are marked as potential ZIP codes; other numbers will not be identified as ZIP codes, to avoid false positives.

3. State names are identified by applying patterns matching all US State names, their abbreviations and variations.

4. Text blocks that begin with street number or PO Box number, and end with a state name followed by a ZIP code with optional country name, are extracted as postal addresses.

5. An address block must have other address elements between the street number and state name. Since at least some characters occur between them, we defined the number of characters ranges from 2 to 60 based on our experiments; otherwise it is not considered to be a valid address.

Algorithm 1 Address extraction with regular expressions

Input: Web Page

Output: Extracted Addresses

1: **Pre-Processing**: The input web page is pre-processed to get clean text.
2: **Matching PO Box**: Variant forms of PO Box are matched, e.g. "PO Box", "P.O. Box" and "Post Office Box".
3: **Matching ZIP and Street Number**: Five-digit numbers followed by an optional four digits are identified as ZIP code; other numbers are identified as potential street numbers.
4: **Matching State Name**: state names and their abbreviations are identified as state name.
5: **Matching Country Name**: Strings "USA", "US", "U.S." or "United States" are matched as country name.
6: **Extracting address**: Text blocks that begin with optional PO Box followed by street number, and end with state name followed by ZIP code and optional country name are extracted as addresses.

As we can see from the above rules, there is no need to detect the *Street Name* and *City Name* for this approach; therefore, the system avoids extensive looking up in dictionaries, and eliminates the tedious work of collecting street and city name data. In many countries, the database of a full list of street names may not exist, or may be available only in multiple data sources with different data codes, and merging them into one data format requires expensive labour.

The extraction algorithm outlined in Algorithm 1 consists of the following two steps:

Pre-Processing

The input web page is segmented into tokens and its HTML structure is analyzed. HTML blocks that are unlikely to contain addresses, such as <script> and <style> blocks, are discarded. All HTML tags are removed as well. The remaining tokens are reconstructed into plain text and fed into the next step.

Address Elements	Regular Expressions
PO Box	((([P|p][.]?[ ]?[O|o][.]?[ ]?)([B|b][O|o][X|x])?)| ("Post Office Box")|([B|b][O|o][X|x])
Street No.	([0-9]{1,6})|"One"|"Two"|"Three"
State Name	Patterns matching all state names
ZIP	[0-9]{5}([-][0-9]{4})?
Country Name	Patterns matching "USA" and all it variations
Address	({POBOX}[ ])?{STREETNO}[ ].{2,60}[,]+{USSTATE}[,.]+ {USZIP}([,]?({USA}))?

Table 3.1: Regular expressions used for extracting addresses

Extracting

The rules used for address extraction are built using regular expressions, generated by using the lexical scanner generator Flex [27] for fast string matching with finite state approaches, they are listed in Table 3.1.

The pre-processed texts are matched against the rules with pattern matching; the longest matching texts are identified as addresses. A list of state names, their abbreviations and variations, are used for matching state names and created according to the Official USPS Abbreviations [1], with 187 names in total on the list.

This system achieved precision of 73.5% and recall of 60.7% on our testing data set. The implementation of the system is easy; it is also computationally efficient since no dictionary look-up is needed. We used this system to extract 414,677 US addresses in 74 minutes from 1.2 million pages of TREC .GOV [2] collection on a 1G byte memory single processor Linux server.

This system also helped us to collect the address data set used in this research;

the details of collecting data set are described in Section 4.3.

3.2.2 Gazetteer Based Approach

We built another rule-based system to extract address blocks by combining heuristic rules and gazetteer look-up to identify address elements.

Rules Used

The following heuristic rules are used:

1. The starting element of an address is street number. Any number found in the input text is an indicator of the starting point of a potential address.

2. An address consists of at least four elements: *Street Number*, *Street Name*, *Street Suffix* (such as Road or Avenue) and *City*. The other elements are optional, such as state name, ZIP code, and country name.

3. A text block that starts with a number and followed by a valid street name, street suffix, and city name is an address block.

4. Any immediately following tokens of city name are likely to be state name, ZIP code or country. If they can be validated by gazetteer look-up, they should be included as part of the address.

Gazetteer

A gazetteer is a dictionary of named geographic places including names, types, locations, and other descriptive information [14]. The gazetteer used in our system to validate potential address elements is TIGER/Line digital mapping data[1], developed by the US Census Bureau, which refers to public-use data from the internal TIGER(Topologically Integrated Geographic Encoding and Referencing) database. It covers the entire US nation, with street names, address ranges, geographic codes, demographics of each side of road/street segments, and latitude-longitude of each intersection.

---

[1]http://www.census.gov/geo/www/tiger

Two tables are built by converting data from the TIGER/Line data. One is *Street* table, which lists all unique streets, cities, states, and ZIP codes. Each row of the table is assigned a unique index. The other is the *Street Index* table, which serves as an index of the Street table for fast street name searching. It contains all unique street names and indexes that point to the position of their first occurrence in the Street table.

Alias Expansion

Many address elements have different abbreviations or additional names; if an alias is not presented in the gazetteer, the name will not be matched. To overcome the problem, we use *Alias* table to convert those names to common names that are used in TIGER/Line data; for example, North East and Northeast are all converted to NE.

Extraction Procedures

The extraction procedures include two phases as follows.

- **Pre-Processing Phase**

  The first step of the address extraction is Pre-Processing, which is described in Subsection 3.2.1. The input web page is tokenized, and HTML blocks that are unlikely to contain addresses are removed, as well as all tags.

- **Extracting Phase**

  After pre-processing, the remaining tokens are fed into the system in sequence. The system marks every number found to be the potential beginning of an address, then recursively parses the following text to find address elements in the sequence of *Street Name*, *City*, *State Name*, *ZIP code* and *Country Name* by matching them with the Street Index and Street tables.

  Specifically, the system searches the following text in the Alias table; if found, replaces it with common names. Then the system matches the string in the Street Index table. Once the street name is matched, it will get its index, go to the index row of the Street table, and begin matching the city name. If the

city name is matched, the system accepts the current input as an address. The same process is applied to State and ZIP code matching, but they are optional for a valid address. All the matching is done with the longest string searching method.

A few noise words are allowed to occur in the middle of address. If a number is not followed by a valid street name, the system will reject it quickly. This system only looks up street name, city and state from the database; information of segments, latitude-longitude were not exploited.

In the example below, Table 3.2 consists of aliases with their common names, Table 3.3 lists a few unique street names with their indexes, and Table 3.4 is an example of a street name table.

To extract the following example address from text:

5500 University Avenue, San Jose

The system first locates the number "5500" as a potential street number, then starts parsing. It will apply alias expansion by replacing "Avenue" with "Ave" after searching the Alias table, then "University Ave" is found in the Street index table. The system therefore goes to Street table, and starts from index 354342 to match the city name "San Jose". Once the city name is matched, it will further match state name, ZIP code, and country name, then output the extracted address.

Name	Converted Name
Street	St
St.	St
Avenue	**Ave**
North East	NE
Northeast	NE
New York	NY

Table 3.2: Example of Alias table

Our gazetteer-based system is developed based on Geocoder, an open source Geocoding system whose author, Dan Egnor, won the 2002 Google programming contest[2]. Our system uses the same TIGER/Line data file that Dan Egnor built. The

---

[2]http://dan.egnor.name/google.html

Street Name	Index
Main St	112355
University Ave	**354342**
Woodland St	482345

Table 3.3: Example of Street Index table

Index	Street Name	City Name	State Name	ZIP
112355	Main St	Houston	Texas	77005
112356	Main St	Santa Monica	CA	90405
112357	Main St	Bellevue	WA	98004
354342	University Ave	Berkeley	CA	94710-2023
354343	University Ave	San Jose	CA	95126
354344	University Ave NE	Seattle	WA	98105

Table 3.4: Example of Street table

extracting procedure we described above is based on the method used in Geocoder, with our modifications to search the exact starting and ending point of address blocks.

The system achieved precision of 83.1% and recall of 68.9% on our testing data set. Benefiting from gazetteer look-up, both the precision and recall of this system are better than those of the regular expression based system. However, the recall of this system is lowered by the fact that it can not recognize university and government addresses, which usually do not contain street numbers and, hence, are not even treated as potential addresses by the system. PO Box addresses are also not extracted because they are not included in TIGER/Line. In addition, the system could not handle the lack of a street suffix (Ave, Street, etc.) or street direction (east, west, etc.), as it considers those to be part of the street name, which must be matched. Lastly, new addresses could not be extracted if they are included in the TIGER/Line.

3.3 Machine-learning Approach

Both the rule-based approaches described above require heuristic rules created manually by human experts, and the rules are not portable when applied to different domains.

The regular-expression approach is computationally efficient and does not rely on any database, but its performance is lower than that of the gazetteer-based approach.

The latter approach has a reasonable performance, but requires a data set that covers every street and city in a country, which is hard to obtain and maintain.

In this section, we propose a *machine-learning* approach that avoids the problems of the above rule-based approaches. The system represents the context surrounding a token using the *word n-gram* model, and features that could potentially help to extract addresses are generated for each token in the n-gram. All the features of the n-gram are input to a *decision tree* inducer trained to determine the role of the features. The inducer will automatically choose the relevant features and learn the extraction rules. The trained classifier is then used to extract addresses. Every candidate token is presented to the classifier as a potential address token, and the classifier will label it as one of four classes: *START*, *MIDDLE*, *END*, and *OTHER*. At the final step, post-processing will be applied to the labeled tokens to extract and output addresses.

The word n-gram model is introduced in Subsection 3.3.1, and the classification method is presented in Subsection 3.3.2. We outline the algorithm in Subsection 3.3.3, also present the system architecture in Subsection 3.3.4. The features used in the system are described in Subsection 3.3.5.

3.3.1 Word n-gram Model

Word n-gram model is used to represent the surrounding context of a token to be classified. In this model, the token at position i is defined as t_i, the feature set of the t_i is defined as $f(t_i)$. The n-gram of token t_i with order n is defined as:

$$ngram_i(n) = \{t_{i-n+1}, .., t_{i-1}, t_i, t_{i+1}, .., t_{i+n-1}\} \tag{3.1}$$

We define the feature set of a n-gram as the combined feature sets of all tokens contained in the n-gram:

$$f(ngram_i(n)) = f(t_{i-n+1}) \cup .. \cup f(t_{i-1}) \cup f(t_i) \cup f(t_{i+1}) \cup .. \cup f(t_{i+n-1}) \tag{3.2}$$

For each $ngram_i(n)$, the value of each feature in the feature set is calculated, and subsequently used along with class label of t_i for training and testing classifier. After the classifier is trained, it reads the feature values of each token and decides which

class label should be assigned to token t_i according to its feature values along with information from its $n-1$ pre-words and post-words.

The number of $n-1$ dummy words shall be added to both the beginning and ending of source text to make the first and last $n-1$ tokens of the text have the same feature set as other tokens.

Following is an example of the n-grams (n=2), along with their generated feature attributes and labeled/predicated class labels.

Example: Input text

California Fish and Game Commission

1416 Ninth Street Sacramento

California 95814 For more information, please call (916)653-4899

Example: Word n-gram output(n=2)

_ **California** Fish $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow OTHER$

California **Fish** and $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow OTHER$

Fish **and** Game $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow OTHER$

and **Game** Commission $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow OTHER$

Game **Commission** 1416 $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow OTHER$

Commission **1416** Ninth $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow START$

1416 **Ninth** Street $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow MIDDLE$

Ninth **Street** Sacramento $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow MIDDLE$

Street **Sacramento** California $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow MIDDLE$

Sacramento **California** 95814 $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow MIDDLE$

California **95814** For $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow END$

95814 **For** more $\Longrightarrow$ {attribute, attribute, ..., attribute} $\Longrightarrow OTHER$

For **more** information $\implies$ {attribute, attribute, ..., attribute} $\implies OTHER$

more **information** , $\implies$ {attribute, attribute, ..., attribute} $\implies OTHER$

... ...

3.3.2   Decision Tree Classifier

The address extraction is viewed as a classification problem. In this case, a classification model is created to analyze the features of each token in the web page, and classify every token into one of four classes: $START$, $MIDDLE$, END and $OTHER$, which mean the token occurs in an address text block as first, middle, last token, or not belonging to an address. We used the C4.5 decision tree [29, 30] as the classification method in our system.

Classification by the decision tree is a supervised approach. By learning from a *training set* of objects whose class is known, the induction algorithm iteratively develops classification rules that are expressed as a decision tree and determines the class of any object from its values of *attributes*: each attribute measures the importance of the feature of an object.

C4.5 recursively partitions the training set by the attribute that divides the data items with the highest *information gain* and adds that attribute to the evolving tree. The iteration continues until the decision tree can classify all items in the training set.

The decision tree is then pruned by the C4.5 learning algorithm to reduce the effect of overfitting the training data set. Pruning a tree is the action to replace a whole subtree by a leaf. It recursively calculates the expected error rate of each subtree: if it is greater than the error rate of a single leaf or a branch, the whole subtree will be replaced by the leaf or branch. The pruned decision tree will be able to better classify more general data.

A more elaborate description of C4.5 can be found from [30, 29].

3.3.3   Algorithm Outline

Our proposed machine-learning approach consists of training and extracting modules. The training module is outlined in Algorithm 2, it takes labeled web pages as input,

and outputs trained classifier model. The training is a a one-time job; after the classifier is trained, unseen web pages will be fed into the extracting module, which is outlined in Algorithm 3.

Algorithm 2 Training module of the machine-learning method
___
Input: Web pages with each token labeled as one of four classes: *START*, *MIDDLE*,
   END and *OTHER*

Output: Trained classifier model
 1: **for** each input web page **do**
 2:    The web page is pre-processed to get HTML layout information, then tokenized.
 3:    Tokens are tagged part-of-speech tags, other feature attributes are extracted as
        well.
 4:    n-grams are generated for each token, the feature attribute set of each n-gram
        is then constructed.
 5:    The classifier is trained with the feature attributes and corresponding class
        labels.
 6: **end for**

The following subsection describes the above algorithms in more detail. The features used in the system will be described in Subsection 3.3.5.

3.3.4   System Architecture Overview

The address extraction system consists of four components: tokenization, feature extraction, classification, and post-processing. The tokenization component divides web pages into tokens, and the feature vector for each token is calculated in the feature extraction component. The classification component learns from the training set and is used to classify each token. At the last step, the post-processing will read the classification results predicted by the classifier, then identify the address blocks and output them. The overall system architecture is presented in Fig. 3.1. The classification component is implemented using C4.5 Release 8 [30] developed by Ross Quinlan in C: all other modules are developed in C++.

Algorithm 3 Extracting module of the machine-learning method

Input: Trained classification model, and unlabeled web pages

Output: Extracted addresses

1: The classifier loads the trained classification model.

2: **for** each input web page **do**

3:   The web page is pre-processed to get HTML layout information, then tokenized.

4:   Tokens are tagged part-of-speech tags, other feature attributes are extracted as well.

5:   n-grams are generated for each token, the feature set of each n-gram is then constructed, and fed to the classifier.

6:   The classifier labels each token as one of four classes: *START*, *MIDDLE*, *END* and *OTHER*.

7:   **if** the first token of a token block is labeled as *START*, the last token is labeled as *END*, and there is at least one token in the block is labeled as *MIDDLE* **then**

8:     **if** the total number of tokens in the block < 20 **then**

9:       Output the token block as extracted address

10:    **end if**

11:   **end if**

12: **end for**
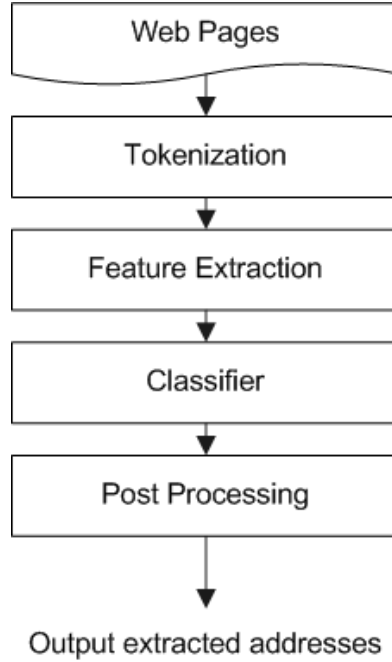
Figure 3.1: System architecture

Tokenization

The tokenization component scans web pages and breaks them into tokens by the process of lexical analysis. HTML tags are analyzed to get layout information: after this phase, they are removed. The tokenizer is implemented using the lexical scanner generator Flex [27].

Feature Extraction

Tokens output by the tokenizer are tagged Part-of-Speech tags using Brill Tagger [8]. The dictionary is searched to assign geographical features such as *state name* and *street direction* to the tokens.

Other features, such as Word Level, Punctuation and Layout features are generated at the same time as the input web pages are tokenized.

n-grams are generated for each token in the web pages, the feature set of each *n*-gram is then constructed and fed to the decision tree classifier for training and testing.

Decision Tree Classifier

The C4.5 classifier is trained with the training data set using features of each n-gram along with the class label of the middle token in the n-gram. The learned classifier will process feature values for each n-gram in the testing data set, and label them as one of *START*, *MIDDLE*, *END*, and *OTHER*. The predicted results are passed to the Post-Processing step.

Post-Processing

This is the last component of the extraction system: it outputs the final extracted addresses.

The predicted results of the classifier are a sequence of tokens with predicted labels, which are one of *START*, *MIDDLE*, *END*, and *OTHER*. This component identifies the address block by applying rule matching to the predicted results. A block of tokens is identified as an address only if its first token is predicted as *START* by the classifier, and its last token is predicted as *END*. There is at least one token in the middle of the block which is predicted as *MIDDLE*. Moreover, the total number of tokens in the block must be less than 20.

More sophisticated methods can be explored in future research. For example, ranking potential address blocks with the confidence scores that the classifier assigned to the tokens as potential *START* or *END*, then using a threshold to filter out the least likely address blocks to improve the precision. Another approach is to train the second classifier to identify the address block(s) automatically.

The features used by the system are broadly grouped into five categories: Word Level, Geographical, Part-of-Speech Tagger, Punctuation, and Layout features. They are described in the following subsections.

3.3.5   Features Used

Word Level Features

The spelling characteristics of words can be exploited to provide evidence for address detection, such as capitalization and digitalization. Word level features are widely used in Information Extraction systems [4, 28]. For example, the feature ALLCAPS

stands for a word with all capitalized characters, and is an indicator of a potential place name, such as abbreviations of US state names. ALLDIGITS, which means a number, is a strong indicator of street numbers or ZIP codes.

A total of 16 word level features were chosen as potentially useful sources of evidence, and extracted using regular expressions with Flex [27]. They are presented in Table 3.5.

Feature	Description	Example
INITCAP	Starts with capital letter	Washington, Main Street
ALLCAPS	All characters capitalized	AVE, WA, USA
CONTAINDIGITS	Contains at least one digit	16th Avenue
ALLDIGITS	All characters are digits	59, 1201
ACRONYM	Abbreviation	U.S.A.
WORD	Other word	street, south
PUNCTUATION	Punctuation	, . -
DATETIME	Date or Time	Mar 8, 2006, 3/8/2006
URL	Web URL	http://www.dal.ca
CONTRACTION	Contains '	What's
SINGLEINITIAL	One character initial	M.
UPPERCASECHAR	One uppercase character	D
LOWERCASECHAR	One lowercase character	d
CONTAINDASH	Contains at least one dash	iso-9001
PHONE	Phone Number	(902) 555-6666
EMAIL	Email address	info@dal.ca

Table 3.5: Word level features

Geographical Features

A total of six Geographical features are used to provide address specific cues to the extracting system, as listed in Table 3.6. They are (1)US State Name; (2)Street Direction; (3)Street Suffix; (4)Secondary Unit Designator; (5)USZIP and (6)POBOX.

The first four features are extracted by looking up input tokens in a dictionary, with all state names, street directions, suffixes, and secondary unit designators, as well as their abbreviations. The dictionary contains a total of 687 entries and is built by us according to the Official USPS Abbreviations [1]. The purpose of using this small size dictionary is to study how well the system can perform without using a large street level geographical database, because such databases do not exist in many

Feature	Examples	Entries in Dictionary
US State Name	AL, CA, FL, Florida	63
Street Suffix	AVENUE, Ave, Street, St, Str	562
Secondary Unit Designators	Apartment, Apt, Building, Bld	46
Street Direction	South, North East, NW	16
POBOX	P.O.Box or PO Box	0
ZIP	Number with ZIP/ZIP+4 format	0

Table 3.6: Geographical features

countries. Also, it is more computationally efficient than using a larger database, which is very important for large scale address extractions. In addition, it requires trivial work to construct the dictionary and integrate it with the system.

The last two features, USZIP and POBox, are used to help the system detect the boundaries of an address text block, since they are usually located at either the starting or ending position of a US address. The USZIP feature is generated by extracting numbers in US ZIP or ZIP+4 formats, which are 5-digit or 9-digit numbers containing dashes. The texts in the form of P.O. Box or PO Box are identified as POBOX features.

Part-of-Speech Tagger

Part-of-Speech(POS) tags are used as Natural Language features, which are widely used in the named entity system [4], as well as in the biomedical entity recognition system [35].

POS are tagged using the Brill tagger with 284 contextual rules, 93,696 words, and 171 lexical rules, trained on a Wall Street Journal(WSJ) corpus. It is reported to have an overall accuracy of about 97% to 98% [8]. A total of 36 Penn Treebank POS tags were used in the POS feature.

Punctuation

Punctuation can also help the classifier capture the syntactic context of the token and improve the accuracy of address boundary detection. For example, addresses in web pages are typically segmented into fields by commas. We consider the following punctuation marks: (1) comma; (2) period; (3) exclamation mark; (4) question

mark; (5) semi-colon; (6) colon; (7) plus or minus sign; (8) apostrophe; (9) left/right parentheses; (10) double quote; (11) left/right brackets (12) equal; (13) underscore; (14) dash and (15) number sign.

Layout

In formal format, addresses are segmented into lines by line breaks or html tables, and shown as blocks in the web pages as following:

Street address

Optional Postal box address

City, State and optional ZIP

Three layout cues, START_OF_LINE, END_OF_LINE, and MIDDLE_OF_LINE, as listed in Table 3.7, are used by the classifier to capture the address format information. They are extracted by parsing HTML line break, span, table, and header tags.

Feature	Description
START_OF_LINE	Word at the beginning of a line
MIDDLE_OF_LINE	Word at the middle of a line
END_OF_LINE	Word at the end of a line

Table 3.7: Layout features

Trained with instances of all the features above, the system automatically learns correlations between different features, and uses them to make address extracting decisions.

The machine-learning based system achieved 94.3% precision and 71.6% recall over our testing data set. In Section 4.4.3, we present the detailed experimental results obtained from the system, including investigations of the effects of features and varying parameters such as order of n-gram, amount of training data and dictionary size.

3.4   Hybrid Approach

As described in the previous section, both precision and recall of our machine-learning based system outperformed the two rule-based systems. However, its recall of 71.6% has more room for improvement. In this section, a hybrid approach combining the rule-based and the machine-learning methods is proposed, based on the following considerations:

- The rule-based methods suffer from the drawback of requiring intense work from experts to manually construct rules. They also have a portability issue when using the system for data set from a new domain. However, the rules handcrafted by experts may capture patterns that statistical algorithms cannot learn from given features. In addition, those patterns may be fairly predictable and hence, could be processed more efficiently with rule pattern matchings.

- On the other hand, the machine-learning method can learn more sophisticated rules than the heuristic rules developed by human experts. It works well in solving information conflicts between instances by learning from diverse sources of features to predict the final label.

  However, not all information used in our rule-based system is provided to the learning system. For example, the Gazetteer-based system exploits street level information by searching its gazetteer, but such information is not included in any feature of our machine-learning based system. Another piece of information missing in the features is the words themselves, which can provide the system with valuable lexical information. For instance, a word sequence of "Mailing Address:" is a strong indicator that the following is an address text block. Its not easy to integrate such lexical information into our model. However, such information can be captured by expert and integrated into the rules.

By combining the rule-based approach with the machine-learning approach, we hope the hybrid system will benefit from both approaches and outperform either one.

Specifically, the input web page is pre-processed into tokens, and the features discussed in Subsection 3.3.5 are generated at this phase. The tokens are then passed into two rule-based systems. Both systems extract addresses from the input tokens,

then label each token as one of four possible values: *START*, *MIDDLE*, *END*, and *OTHER*. Each token will have two labels assigned after output by the systems: these two label values are added to the existing feature set as two new features. Tokens with the new feature set are then fed to the machine-learning system to learn and extract addresses.

The hybrid system consists of the training and extracting modules, similar to the machine-learning system. They are outlined in Algorithm 4 and 5 respectively.

Algorithm 4 Training module of the hybrid method

Input: Web pages with each token labeled as one of four classes: *START*, *MIDDLE*, *END* and *OTHER*

Output: Trained classifier model

1: **for** each input web page **do**
2:     The web page is pre-processed to get HTML layout information, then tokenized.
3:     Tokens are fed into two rule-based systems.
4:     The tag assigned to each token by each system is noted, and used as a feature of the token.
5:     Tokens are tagged part-of-speech tags, other feature attributes are extracted as well.
6:     n-grams are generated for each token, the feature attribute set of each n-gram is then constructed.
7:     The classifier is trained with the feature attributes and corresponding class labels.
8: **end for**

During training, the system will learn its features including the two new features output by the rule-based systems. With the two new features, the system has two more reliable evidences for making better prediction. If it sees conflicts between the two new features and other features, it will adjust the classification model to solve the conflicts automatically according to the training instances. The experimental results of the hybrid system are shown in Section 4.4.4.

Algorithm 5 Extracting module of the hybrid approach

Input: Web pages

Output: Extracted addresses

1: The classifier loads the trained classification model.

2: **for** each input web page **do**

3:     The web page is pre-processed to get HTML layout information, then tokenized.

4:     Tokens are fed into two rule-based systems.

5:     The tag assigned to each token by each system is noted and used as a feature of the token.

6:     Tokens are tagged part-of-speech tags, other feature attributes are extracted as well.

7:     n-grams are generated for each token, the feature set of each n-gram is then constructed, and fed to the classifier.

8:     The classifier labels each token as one of four classes: *START*, *MIDDLE*, *END* and *OTHER*.

9:     **if** the first token of a token block is labeled as *START*, the last token is labeled as *END*, and there is at least one token in the block is labeled as *MIDDLE* **then**

10:         **if** the total number of tokens in the block < 20 **then**

11:             Output the token block as extracted address

12:         **end if**

13:     **end if**

14: **end for**

Chapter 4

Empirical Study

4.1 Performance Measures

We measure the performance using *Precision*, *Recall* and *F-Measure* which are common standards to evaluate the performance of a classifier.

We define TP as the number of true positive words, FN as the number of false negative words, FP as the number of false positive words.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

The *F-measure* [36] provides a method for combining Precision(P) and Recall (R) into a single summary number:

$$F\text{-}Measure = \frac{(\beta + 1.0) \times P \times R}{\beta^2 \times P + R} \tag{4.3}$$

The parameter β represents the relative weight of Recall to Precision. In the information extraction field, F_1 score is typically used by setting β to be 1, which equally favors Recall and Precision.

$$F_1 = \frac{2 \times P \times R}{P + R} \tag{4.4}$$

F_1 is set to be 0 if both Precision and Recall are 0.

4.1.1 Measures for Exact Matching

For measuring the performance of exact matching, an extracted address is defined to be correct only if its start and end positions exactly match the start and end positions of the address labeled by the expert.

4.1.2 Measures for Per-class Matching

While exact extractions are ideal, partially extracted addresses are also valuable for many type of applications. For example, geocoding applications are usually able to correctly map the address into coordinates, even if its state name or ZIP is missing. For those applications, measuring Recall, Precision and F-measure for each class will be more appropriate: in our case the classes are *START*, *MIDDLE*, *END*, and *OTHER*.

In addition, measuring per-class matching gives us the insight on the performance of predicting each class, which helps developing and improving the systems: we can tell which class is difficult to predict, and adjust rules or features accordingly.

4.2 Data Collecting

As far as we know, there is no public data set available for evaluating address extraction. To collect data, we queried Google with different sets of queries. For each query, Google returned 1000 web pages. Every web page that contains US addresses was collected and labeled. Web pages with the same URL returned by the different queries were saved only once. In this way, we collected and labeled three web page collections. The first, "*Contact Collection*" was collected using two queries: "contact us" and "contact information". The second,"*Hotel Collection*" was collected using the queries "Hotel Los Angeles", "Hotel San Francisco", "Hotel New York", and "Hotel Seattle". The last, "*Pizza Collection*" was collected using the queries "Pizza Los Angeles", "Pizza San Francisco", "Pizza New York", and "Pizza Seattle".

An address data collector was created using the regular expressions approach described in Section 3.2: it was used to help us to efficiently and accurately label addresses in the web pages. The system serves as a HTTP proxy server, and users surf the Internet through its proxy service. During the Internet surfing, the system automatically highlights the addresses that it detected, and allows users to input feedback for each page to indicate whether the addresses are highlighted correctly, and whether there is any address in the page that is not recognized by the system. Web pages containing address information are automatically tagged by the system and saved, and user input is also logged. After all web pages were collected, we manually

Figure 4.1: A screen-shot of the address data collector. It highlights the extracted addresses, and allows users to input how many addresses are extracted correctly. By clicking the "Save Webpage" button, the user feedback and tagged web page are automatically saved.

relabeled the incorrectly tagged web pages according to the logged user input. A screen shot of the address data collector is shown in Fig. 4.1. This method allows a large number of web pages to be accurately collected and tagged with a minimum of effort. A total of 2,375 web pages with 12,895 US addresses were collected and labeled in seven person-days.

SGML begin tag <GEO TYPE="US"> and end tag </GEO> were used to label the US addresses in the web pages. Following is an example of one labeled address in a collected web page:

Best N.Y. style wood-fried oven pizza in the L.A. area. </br>

*<GEO TYPE="US">2764 Rowena Avenue
*

*(1/2 Block West of Glendale Blvd)
*

*Silverlake, CA</GEO> (323) 664-3333
*

4.3 Data Set

All of our experiments were conducted using the three collections that we collected and labeled, as described in Section 4.2. The three collections contain a total of 2,375 web pages and 12,895 US addresses with 11,343 distinct. There are 2,106,749 tokens after pre-processing, among them 116,043 tokens belonging to US addresses. The majority of the tokens are non-address tokens, only 5.5% of the tokens belong to addresses. The average number of tokens that each US address contains is 9, including punctuation.

The "Contact Collection" comprises 897 unique web pages. Most of the collected pages are company, government and university web pages with high search engine ranks. The "Hotel Collection" and "Pizza Collection" have 956 and 504 unique web pages respectively. Most of those pages are from the hotel or pizza company web sites, Yellow Pages, travel guide, or consumer review web sites. A summary of the data set is listed in Table 4.1.

	Number of Web Pages	Address Count	Unique Address	Average Length	Token Count	Address Token Count
Contact	897	2,804	2,464	9.17	432,059	25,701
Hotel	956	6,150	5,363	9.00	1,049,135	55,411
Pizza	504	3,941	3,539	8.86	625,555	34,931
All	2,357	12,895	11,343	9.00	2,106,749	116,043

Table 4.1: Data set summary

4.4 Experimental Results

In this section, we present the experimental results for the systems that we implemented. The whole data set was split into three sets: *training set*, *testing set*, and *validation set*.

20% of the total web pages were randomly selected as *testing set*, which all systems were tested against and which has 471 web pages with 2,257 labeled addresses. The remaining 1,886 web pages were further split into *training* and *validation* sets in the manner of *ten-fold cross validation* [20]. Training set was used to train the classifier, and validation set used to prune the decision tree to prevent data over-fitting. The

1,886 web pages were randomly split into 10 equal size mutually-exclusive subsets. The classifier inducer was trained and tested 10 times, and each time it used all but one of the 10 subsets: the unseen subset was used for validation. The training was ended after the error on validation set was minimized. After training, the induced classifier was evaluated on the entire independent 471 web page testing set.

Each experiment was conducted 10 times using different subsets for training and validation to prevent idiosyncrasies in the splitting of training and validation data set from affecting our results. The reported results are the averaged results of all 10 runs. Note, the training and validation sets were not used by the rule-based systems.

Experiments were conducted over each system: rule-based system using regular expression approach, gazetteer-based approach, machine-learning based system, and hybrid system. The experiment results are presented in the following subsections. Both exact matching and average matching were measured, but due to space limitations, we only report exact matching in most reports.

4.4.1 Regular Expression Approach

This system recognized 1,864 addresses from testing web pages, which have a total of 2,257 labeled addresses, 1370 of which are correct with exact matching. The F-Measure for exact matching is 0.665 with precision of 0.735 and recall of 0.607, as shown in Table 4.2. The result of extraction by each class is listed in Table 4.3.

	Precision	Recall	F_1	True Positive	False Positive
Exact Matching	0.735	0.607	**0.665**	1370	494

Table 4.2: Performance for exact address matching

	Precision	Recall	F_1
START	0.746	0.620	**0.677**
MID	0.856	0.840	**0.848**
END	0.964	0.801	**0.875**
OTHER	0.992	0.995	**0.993**

Table 4.3: Performance by classes

The low recall is mainly because of the heuristics that we used: we assume that an address starts with a street number or PO Box and ends with a state name plus

ZIP code. This pattern is used by the system to extract the starting position of an address. However, in the testing data set, there are 272 addresses without ZIP codes, about 12.1% of the total addresses; and 215 or 9.5% addresses without a state name. Also, 112 or 5% of addresses do not start with a street number or PO Box. In addition, there are some addresses with country occurring between the state and the ZIP code, or with a variation of the state name which is not included in the dictionary: therefore, they could not be matched by our regular expressions.

For example:

- *16633 Dallas Parkway Suite 800 Addison, Texas USA 75001*

- *102 Manatee Avenue West Bradenton, Fla. 34205*

The first address has USA between the state and the ZIP code; in the second, Fla. is not included in our dictionary of state names.

The system is not able to recognize any of the addresses mentioned above.

The low precision is largely attributed to the low precision of *START* class, as shown in Table 4.3. The *START* class achieved lower precision and recall than the other classes. It reflects that there are more varieties of formats at the starting parts of addresses, and it is not easy to match them by applying handcrafted rules. The result of this problem is that some addresses are extracted with incorrect starting boundaries - they are either partially extracted or fully extracted, but with extra non-address tokens included. The followings are some examples of such addresses extracted with incorrect starting boundaries:

Examples of partially-extracted addresses with incorrect starting boundaries; the extracted portions are shown in bold font.

- *Suite **540 901 South Bond Street Baltimore, Maryland 21231***

- *75 Rev. Dr. Martin Luther King Jr. Blvd., Room **231 St. Paul, MN 55155-1606***

- *Seven High Street, Suite **407 Huntington, NY 11743***

Examples of fully-extracted addresses with extra non-address tokens included:

- *2 For 1 Pizza Co 1905 South Western Avenue , Los Angeles , CA 90018*

- *2005 Contact Us Duluth News Tribune 424 W. First St . Duluth , MN 55802*

Instances labeled *OTHER* are classified with the highest F_1: this is also the case for our other system. Since only *START*, *MID*, and *END* will be used for extracting address, we will not discuss the performance of *OTHER* class in the later sections.

4.4.2 Gazetteer Based Approach

The system using the gazetteer-based approach achieved both better precision and recall than the regular expression based system, as can be seen from Tables 4.4 and 4.5. The better precision is not surprising: by validating address elements with gazetteer covering all United States cities and street names, the system avoids many false positives, and hence, improves its precision. The precision for recognizing *START* is lower than the other classes, which are affected by the addresses extracted with incorrect starting boundaries.

	Precision	Recall	F_1	True Positive	False Positive
Exact Matching	0.831	0.689	**0.753**	1555	317

Table 4.4: Performance for exact address matching

	Precision	Recall	F_1
START	0.856	0.711	**0.777**
MID	0.934	0.794	**0.858**
END	0.924	0.767	**0.838**
OTHER	0.990	0.998	**0.994**

Table 4.5: Performance by classes

Unlike the regular expression based approach, this system can correctly extract addresses without a ZIP code and sometimes even without state names: this advantage contributes to its better recall.

However, the following facts degrade its recall:

Addresses missing street suffix or direction could not be recognized because they are needed by the system to match street names in TIGER/Line.

The system assumes address elements appear in a fixed order, and requires that a street must be followed by a valid city name, then optional state name and ZIP code.

However, many addresses have extra words between street name and city, such as in the following:

800 North Point Street, (at the corner of Hyde Street), San Francisco, CA

The current implementation could not extract the above address correctly.

In addition, PO Box addresses could not be recognized by the system, since TIGER/Line data does not include them. New addresses cannot be extracted either, if they have not been included in TIGER/Line.

4.4.3 Machine-learning Approach

In this subsection we present results of the experiments conducted on our machine-learning method based system.

Effects of Features

We examined how well the Word-Level (Word-Lev.), Part-Of-Speech (POS), Geographical (Geo.), Layout (La.) and Punctuation (Punc.) features perform when used independently of each other, as can be seen from Table 4.6. The effects of n-gram order was also tested. Results of Layout and Punctuation are not included in the table: their precision and recall are zero for all orders of n-gram, because the classifier simply labels every instance as *OTHER*, since the confidence of predicting them to be other classes is so low.

In Table 4.6, the first row shows the performance of Word-Level feature, second to fourth rows show the performance of Word-Level feature with different order or n-gram. The remaining part of the table shows results for Geographical, and POS features respectively.

None of the features performed well on the address extraction task when used alone. The Word-Levelfeature performed best. Inside of the induced decision trees, INITCAP, USZIP, NNP(Proper noun) and CD(Cardinal number) were put at the root level of the tree for Word-Level, Geographical and POS features respectively. It indicates they are the most effective values in the feature set for detecting address tokens.

Features	n	Precision	Recall	F_1
Word-Lev.	1	0	0	**0**
Word-Lev.	2	0.986	0.158	**0.272**
Word-Lev.	3	0.918	0.453	**0.605**
Word-Lev.	4	0.925	0.503	**0.651**
Geo.	1	0.452	0.017	**0.032**
Geo.	2	0.771	0.038	**0.073**
Geo.	3	0.889	0.088	**0.161**
Geo.	4	0.885	0.279	**0.424**
POS	1	0	0	**0**
POS	2	0.675	0.005	**0.010**
POS	3	0.874	0.360	**0.510**
POS	4	0.875	0.386	**0.536**

Table 4.6: Performance of individual features and the effects of n-gram

We then conducted experiments using each possible pair-wise combination of Word-Level, Geographical and POS features, and the results are listed in Table 4.7.

Features	n	Precision	Recall	F_1
Word-Lev. & POS	1	0	0	0
Word-Lev. & POS	2	0.960	0.186	0.311
Word-Lev. & POS	3	0.928	0.531	0.675
Word-Lev. & POS	4	0.926	0.550	0.690
Word-Lev. & Geo.	1	0.454	0.017	0.033
Word-Lev. & Geo.	2	0.900	0.326	0.479
Word-Lev. & Geo.	3	0.927	0.661	0.771
Word-Lev. & Geo.	**4**	**0.934**	**0.707**	**0.805**
Geo. & POS	1	0	0	0
Geo. & POS	2	0.870	0.276	0.419
Geo. & POS	3	0.912	0.590	0.716
Geo. & POS	4	0.904	0.642	0.751

Table 4.7: Performance of pair-wise combination of features and the effects of n-gram

The combination of Geographical feature with Word-Level and POS feature both increased the performance, which implies that the geographical feature contains information that both Word-Level and POS features do not have. The Geographical and Word-Level pair achieved the best F_1, 80.5%. Meanwhile, the Word-Level and POS pair gained little on performance. This can be explained as, in POS feature, some strong indicative information for identifying an address, such as NNP and CD, could also be exploited from Word-Level features, such as INITCAP and ALLDIT,

which often indicate a proper noun.

Lastly, we tested the performance by adding one feature at a time until all of the features were used. Punctuation and Layout features were also included in the testing and used as one set of features. The result is shown in Table 4.8.

Features	n	Precision	Recall	F_1
Word-Lev.	1	0	0	0
Word-Lev.	2	0.986	0.158	0.272
Word-Lev.	3	0.918	0.453	0.605
Word-Lev.	4	0.925	0.503	0.651
+Geo.	1	0.454	0.017	0.033
+Geo.	2	0.900	0.326	0.479
+Geo.	3	0.927	0.661	0.771
+Geo.	4	0.934	0.707	0.805
+POS	1	0.454	0.017	0.033
+POS	2	0.916	0.379	0.536
+POS	3	0.936	0.652	0.769
+POS	4	0.936	0.717	0.812
+Punc.+La.	1	0.871	0.500	0.631
+Punc.+La.	2	0.947	0.704	0.807
+Punc.+La.	3	0.939	0.709	0.808
+Punc.+La.	**4**	**0.943**	**0.716**	**0.814**

Table 4.8: Performance gained by adding features and the effects of n-gram order

The Punctuation and Layout features helped to improve the system performance even though they did not perform well if used alone. Adding the POS feature increased the F_1 somewhat, only about 0.7% when n is 4. Inside the induced decision tree, the geographical feature USZIP is put at the root, indicating USZIP has the largest information gain, which is the most indicative cue of an address.

Effects of n-gram Order

In order to determine the effects of n-gram order over the extraction performance, we conducted experiments by varying the order of n-gram using all features for training and testing.

Fig. 4.2 displays the precision, recall and F_1 for each order or n-gram. Since the experiments are conducted 10 times for each order of n, we also have an error bar included. By increasing the order of n-gram, the recall of the system was improved
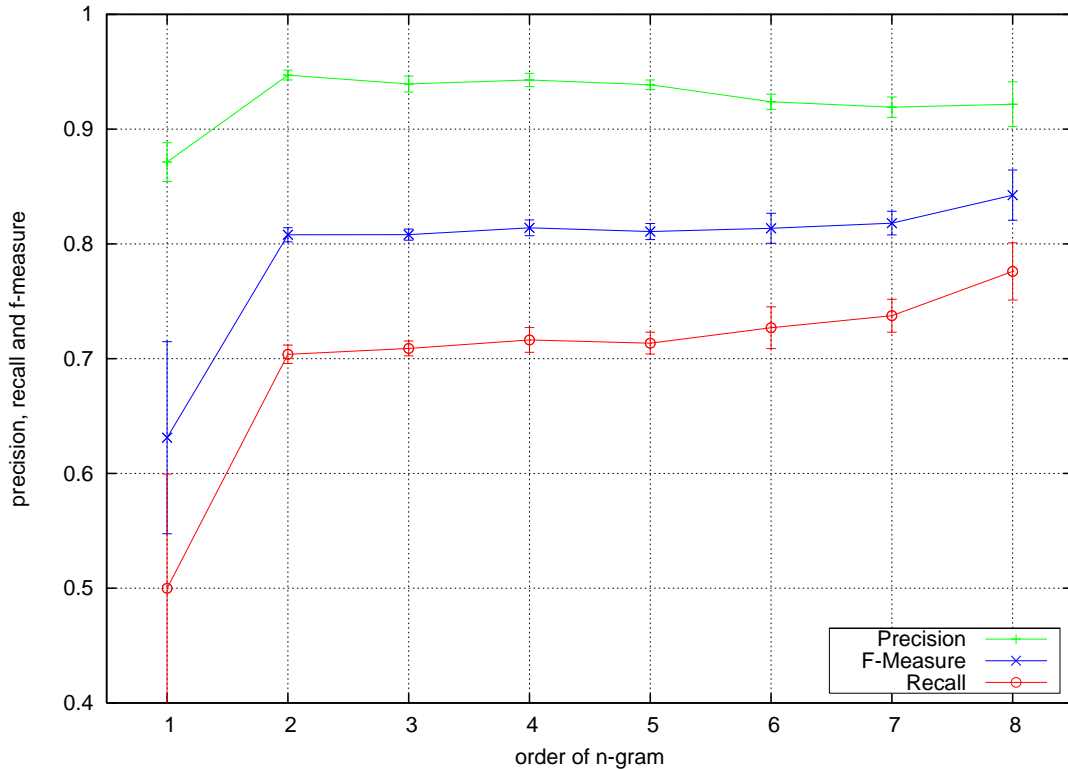
Figure 4.2: Effects of n-gram over performance

gradually by getting more contextual information. While the highest precision was reached with 2-gram, thereafter increasing order of n-gram slowly degraded the precision. The best F_1, 0.843, was achieved with the highest order of n-gram that we experimented with, which was 8. However, we could not simply conclude that the system with the higher order of n-gram performs better than lower order. The appropriate n-gram order will be decided by the applications. For some applications, precision is more important than recall, and a smaller n should be preferred. If recall is more important, a larger n should be considered. Also, increasing the order of n-gram will require more time for generating features and training system. For large scale address extractions, a small n should be preferred.

Effects of Training Data Size

The amount of training data required to achieve acceptable performance is an important question in all extraction tasks, because for most information extraction problems, large amounts of unlabeled data usually can be found easily, while labeled data
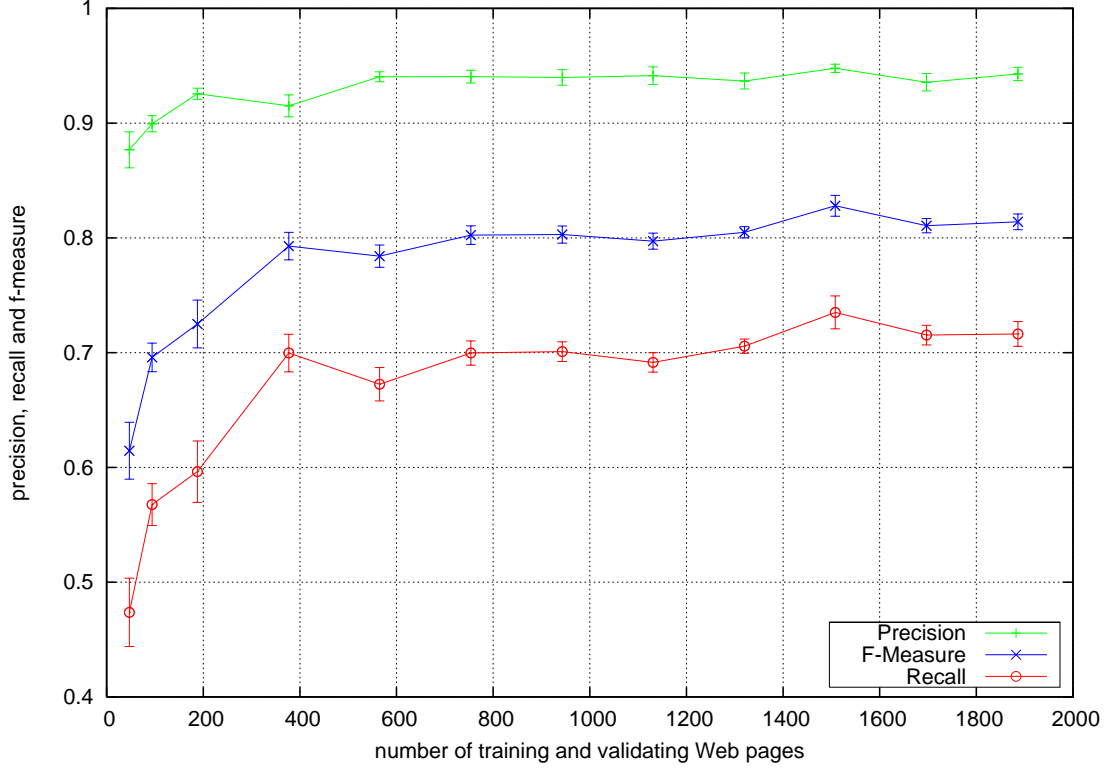
Figure 4.3: Performance as a function of the size of training/validation set with 10 fold cross validation

is often scarce and costly to obtain. We studied how the performance varies as the training set size is increased. We ran a sequence of experiments on the system with order of word n-gram fixed to 4. For each experiment, we increased the number of web pages used for training and validation: 90% of those pages were randomly chosen for training and the remaining 10% used for validation. Then we tested the system with the same testing set, composed of 471 web pages used for other experiments, and each experiment was run 10 times using different parts training and validation data, for training and validation. The results are shown in Fig. 4.3 with the error bar displayed.

By increasing the training size from 47 to 377 web pages, the F_1 goes up from 0.615 to 0.793 quickly. But when the training size was increased from 377 to the total 1886 web pages, the recall and F_1 gained little, only 1.6% and 2.1% respectively. The precision was improved slightly more than recall, with an increase of 2.8% increasing.

The highest F_1 was achieved when using 1508 web pages for training and validation. As we can see from the trend of F_1, more training web pages help improve performance, but further training beyond about 500 pages did not increase it significantly.

Effects of Dictionary Size

This system uses a dictionary with total 687 entries to construct geographical features such as state name, street suffix and street direction, etc.: the dictionary does not cover street or city level names.

The following experiments were conducted in order to see whether using a larger dictionary that covers street and city names would help performance. Also how well the system performances if no dictionary was used. The order of word n-gram was fixed to be 4.

To test the system with the larger dictionary, we used TIGER/Line to lookup street, city and state names by matching the longest string with database entries. The TIGER/Line files cover geographic features of the entire United States, including features such as street and city names. New features called STREET and CITY were created, which indicated whether the token was a street or city name. The system then trained with the new features combined with existing features.

Another test was conducted to test the system performance without dictionary, we removed features that require dictionary look-up from the training and testing data, such as state name, street suffix.

	Precision	Recall	F_1
Baseline	0.943	0.716	0.814
No Dic.	0.935	0.718	0.813
+TIGER/Line	0.932	0.753	0.833

Table 4.9: Effects of dictionary size

Table 4.9 summarizes the experiment results. The first line of the table shows the performance of the baseline system, which uses a small dictionary with 687 entries. The second line shows the performance without using any dictionary. The last one used the whole TIGER/Line data, with extensive dictionary look-up. The result

shows that using a small dictionary helped the address extraction performance somewhat. The interesting thing is that using the TIGER/Line data lowered the system precision rather than improving it. The ambiguities of place names are most likely to be the reason for the lower precision. Smith et al [34] reported that about 92% of names in their corpus are ambiguous. Amitay et al [3] found that the majority of names on the web are ambiguous. There is an average two possible meanings for each potential geographic name mentions in their data set. For example, in the United States, names of many streets or towns are also person names. The classification inducer would be "confused" if it saw many false positive street or city names.

More sophisticated approaches to utilizing a dictionary to identify street and state name could be explored in future research. The rules that we used to validate address elements in our rule-based systems can be applied. A stop list could be used to avoid identifying some tokens as address elements if they have an overall high false-positive value. Amitay et al [3] proposed a method by assigning confidence scores to disambiguated potential mentions, in which only the ones exceeding a given threshold are identified as place names.

Error Analysis

Table 4.10 shows the classification confusion matrix of address extraction when using 8-gram.

	START	MID	END	OTHER
START	1968	46	1	242
MID	75	14260	23	1174
END	0	50	1984	223
OTHER	206	829	116	403028

Table 4.10: Confusion matrix of address extraction when using 8-gram

The class *OTHER* is the most confusing one, and most error instances of *START*, *MID* and *END* are misclassified as *OTHER*. This is caused by the fact that the majority of tokens in the data set belong to OTHER. In our data set, only 5.5% of the total tokens belong to address, as we summarized in Section 4.3. Since *START*, *MID*, and *END* class labels appear much less frequently than *OTHER*, their instances are more likely to be mislabeled as *OTHER* by the classifier to achieve an overall

lower error rate. To get a better model with such "unbalanced data", a cost-sensitive classifier could be built with an appropriate cost matrix to balance the distribution of class labels when a decision tree is trained.

START is the most difficult class to be predicted: its precision of 0.88 is lower than others.

4.4.4 Hybrid System

To test the hybrid system, we conducted experiments for three hybrid system versions by combining machine-learning with the regular expression approach, with the gazetteer approach, and with a combination of the three.

The experiments for the system combining the machine-learning and the regular expression approaches are conducted with the following steps, with word n-gram is fixed to be order 4.

First, web pages are fed into the regular expression based system. Addresses are extracted and tokens in the web pages are output from the system. Each token is labeled as one of *START*, *MIDDLE*, *END* or *OTHER*.

Second, the class label of each token is added to the token's feature set as a new feature. Those tokens are then fed into the machine-learning based system for training and testing.

The machine-learning system outputs the extracted addresses, and the performance results are measured.

Similar steps are used for the other two systems.

System	Precision	Recall	F_1
Machine-learning Based System	0.943	0.716	0.814
Machine-learning+Regular Expression	**0.952**	**0.811**	**0.876**
Machine-learning + Gazetteer	0.943	0.784	0.856
All Systems Combined	0.953	0.811	0.876

Table 4.11: Performance of the hybrid system

The first line shown in Table 4.11 is the performance of the machine-learning based system, which is used as a baseline.

The second and third lines show the performance of combining machine-learning with the regular expression and gazetteer based systems respectively. In the last

experiment, we combined all three systems, and tested the performance.

The combination of the machine-learning and regular expression approaches has the highest performance gain, in which F_1 is improved by 6.2%. Combining all three systems has almost the same performance as that of the machine-learning and regular expression combination: therefore, our hybrid system only uses the machine-learning and regular expression combination for more efficiency.

4.4.5 System Comparison

In this section, we first compare side-by-side the best performance achieved by each system, then discuss the advantages and disadvantages of each system.

The first three lines in Table 4.12 show the performance of the two rule-based systems and the machine-learning based system, using 4-gram. The last is the result for the hybrid system, which combines machine-learning system with the regular expression based system.

System	Precision	Recall	F_1
Regular Expression Based	0.735	0.607	**0.665**
Gazetteer Based	0.831	0.689	**0.753**
Machine-learning Based	0.943	0.716	**0.814**
Hybrid	0.952	0.811	**0.876**

Table 4.12: Summary of best results

The machine-learning based system outperforms the two rule-based systems. The hybrid system further improves the performance with 6.6% improvement in F_1 over the machine-learning based system.

There are advantages and disadvantages for each system. For certain types of applications, one may be more appropriate than the others.

Rule-based systems are easy to implement and they have reasonable performance. They are more computationally efficient than the machine-learning based system. They require either no dictionary lookup or dictionary look-up only after a potential street number or PO Box number is matched.

However, they suffer from the following drawbacks:

1. Data sets like TIGER/Line, which cover every city and street in a country, may not exist in countries other than the United States. Even if such data

sets do exist, accurately integrating them from various sources with different formats and levels of accuracy is a challenge. Integrating such databases with the extracting system for accurate address matching is also difficult and time consuming.

2. The rules that do well for one country may not do well for other countries. The address format varies greatly from one country to another. Specific sets of rules need to be constructed for each country. The rules may also need to be manually changed when the system is used with other data sets.

3. Even for addresses from within the same country, the formats vary, which makes it difficult to manually construct rules. Some will have "street number, street, city, state and ZIP", some will miss one or more of them, or be written in a different order. They may also have extra text in them, such as name of recipient, building name, or corporate routing information.

4. Misspellings and the wide variation in abbreviations will also degrade the performance of a particular method, e.g. Mass., Fla., and Calif. will not be considered as state names unless they are included in the database. To replace misspellings and abbreviations, an alias type of table usually needs to be manually created.

The machine-learning based system has a higher performance. It is easy to add new features to the system, and the system will automatically ignore any irrelevant features. It can also be easily ported to other domains and languages. However, in terms of computational cost, it requires classifier training and feature generating, and in this respect is more expensive than rule based system.

The hybrid system has the highest precision and recall on address extraction among all systems that we built. However, it takes more time than the machine-learning based systems to perform address extracting.

Chapter 5

Conclusion

This thesis addresses the problem of high performance automatic extraction of postal addresses from web pages. We proposed three methods for recognizing addresses: rule-based, machine-learning, and hybrid.

We demonstrated that a machine learning approach to address extraction is more effective than the rule based approaches. Our machine-learning based system uses the word n-gram model and classification method, and possesses the following desirable characteristics:

- It is trainable and does not require hand-built rules.

- If new knowledge sources become available, the system allows the user to integrate them as additional features easily.

- It is easier to port the system to extract addresses from different countries.

- It produces very accurate results even with no dictionary look-up.

- The machine-learning methods we adapted in this research can also be used in other information extraction tasks.

The hybrid method we propose combines the rule-based method and the machine-learning approach. The accuracy of the system is further improved.

Three data sets were created to provide more reliable evaluation on our systems. An extraction system was built to help us collect and tag web pages accurately and easily. These data sets can be made publicly available for other researchers.

In the future, we hope to study how character n-gram could be applied to our system, and evaluate our machine-learning base system on extracting addresses from other countries and in different languages. We also wish to adapt our systems to other information extraction tasks.

Bibliography

[1] Official USPS abbreviations. http://www.usps.com/ncsc/lookups/usps_abbreviations.html, accessed on March 8, 2006.

[2] Text retrieval conference 2003: .GOV test collection. http://ir.dcs.gla.ac.uk/test_collections/, accessed on March 8, 2006.

[3] Einat Amitay, Nadav Har'El, Ron Sivan, and Aya Soffer. Web-a-where: geotagging web content. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval*, pages 273–280, New York, NY, USA, 2004. ACM Press.

[4] Shumeet Baluja, Vibhu Mittal, and Rahul Sukthankar. Applying machine learning for high performance named-entity extraction. *Computational Intelligence*, 16, November 2000.

[5] D. Bikel, S. Miller, R. Schwartz, and R. Weischedel. Nymble: A high-performance learning name-finder. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pages 194–201, 1997.

[6] Vinayak R. Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. Automatic segmentation of text into structured records. In *SIGMOD Conference*, 2001.

[7] A. Borthwick. *A Maximum Entropy Approach to Named Entity Recognition*. Phd thesis, New York University, 1999.

[8] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.

[9] O. Buyukkokten, L. Gravano J. Cho, d H. Garcia-Molina, and N. Shivakumar. Exploiting geographical location information of web pages. In *Proceedings of Workshop on Web Databases WebDB99*, pages 91–96, 1999.

[10] Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.

[11] Junyan Ding, Luis Gravano, and Narayanan Shivakumar. Computing geographical scopes of web resources. In *26th International Conference on Very Large Databases, VLDB*, Cairo, Egypt, September 10-14 2000.

[12] C. Jones et al. Spatial information retrieval and geographical ontologies: An overview of the SPIRIT project. In *Proc. of the 25th Annual International ACM SIGIR Conference on Research and Development in Informaiton Retrieval*, pages 387–388, Tampere, Finland, 2002.

[13] Ralph Grishman and Beth Sundheim. Design of the MUC-6 evaluation. In *MUC6 '95: Proceedings of the 6th conference on Message understanding*, pages 1–11, Morristown, NJ, USA, 1995. Association for Computational Linguistics.

[14] Linda L. Hill, James Frew, and Qi Zheng. Geographic names: The implementation of a gazetteer in a georeferenced digital library. *D-Lib Magazine*, January 1999.

[15] K. Humphreys, R. Gaizauskas, S. Azzam, C. Huyck, B. Mitchell, H. Cunningham, and Y. Wilks. University of Sheffield: Description of the LaSIE-II system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conferences MUC-7*, Morgan Kaufman, 1998.

[16] Craig Nohl Jan. A system for address extraction from facsimile images. In *Symposium on Document Image Understanding Technology, Apr. 1999, Annapolis, Maryland*, 2004.

[17] H. Jerry, R. Douglas, E. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson. FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In *Roche, E., Schabes, Y., eds.: Finite-State Language Processing*, Cambridge, MA, 1997. MIT Press.

[18] Sittichai Jiampojamarn, Nick Cercone, and Vlado Keselj. Biological named entity recognition using n-grams and classification methods. In *Proceedings of the Conference Pacific Association for Computational Linguistics, PACLING'05*, Meisei University, Hino Campus, Tokyo 191-8506 Japan, August 2005.

[19] Tatsuhiko Kagehiro, Masashi Koga, Hiroshi Sako, and Hiromichi Fujisawa. Address-block extraction by Bayesian rule. In *17th International Conference on Pattern Recognition (ICPR'04)*, volume 2, pages 582–585, 2004.

[20] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. Int. Joint Conf. Artificial Intelligence*, pages 1137–1145, Montreal, Canada, 1995.

[21] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[22] A. Lakhina, J.W. Byers, M. Crovella, and I. Matta. On the geographic location of internet resources. *IEEE Journal on Selected Areas in Communications*, 21(6):934–948, Aug 2003.

[23] L. Lloyd, P. Kaulgud, and S. Skiena. Newspapers vs. blogs: Who gets the scoop? In *AAAI Spring Symposium on Computational Approaches to Analyzing Weblogs(AAAI-CAAW)*, Palo Alto, California, USA, March 2006.

[24] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Seventh Conference on Natural Language Learning (CoNLL)*, 2003.

[25] Kevin S. McCurley. Geospatial mapping and navigation of the web. In *Proceedings of the Tenth International Conference on World Wide Web*, pages 221–229. ACM Press, 2001.

[26] A. Mikheev, C. Grover, and M. Moens. Description of the LTG system used for MUC-7. In *Seventh Message Understanding Conference MUC-7: Proceedings of a Conference held in Fairfax, Virginia*, April 29 1998.

[27] G. T. Nicol. Flex - the lexical scanner generator. The Free Software Foundation, Cambridge, MA., 1993.

[28] F. Peng and A. McCallum. Accurate information extraction from research papers using conditional random fields. In *Proceedings of Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, 2004.

[29] J. R. Quinlan. Induction of decision trees. In *Machine Learning*, volume 1-1, pages 81–106, 1997.

[30] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[31] E. Rauch, M. Bukatin, and K. Baker. A confidence-based framework for disambiguating geographic terms. In *Proceedings of the HLT-NAACL 2003 workshop on Analysis of geographic references*, pages 50–54, 2003.

[32] S. Sekine, R. Grishman, and H. Shinnou. A decision tree method for finding and classifying names in Japanese texts. In *In Proceedings of the Sixth Workshop on Very Large Corpora*, 1998.

[33] Prasun Sinha. Cursive script postal address recognition. Master thesis, Michigan State University, 1997.

[34] David A. Smith and Gregory Crane. Disambiguating geographic names in a historical digital library. In *ECDL '01: Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries*, pages 127–136, London, UK, 2001. Springer-Verlag.

[35] Lorraine Tanabe and W. John Wilbur. Tagging gene and protein names in biomedical text. In *Bioinformatics*, volume 18, pages 1124–32, 2002.

[36] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition.* Dept. of Computer Science, University of Glasgow, 1979.

[37] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT(13):260–269, April 1967.

[38] Roman Yangarber and Ralph Grishman. NYU: Description of the Proteus/PET system as used for MUC-7 st, 1998.

[39] GuoDong Zhou and Jian Su. Named entity recognition using an HMM-based chunk tagger. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 473–480, Morristown, NJ, USA, 2001. Association for Computational Linguistics.