In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

COMPARING MACHINE LEARNING AND HAND-CRAFTED APPROACHES FOR INFORMATION EXTRACTION FROM HTML DOCUMENTS

A Thesis Presented

by

RON SINGER

Submitted to the Graduate School
McGill University in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE

February 2003

School of Computer Science
McGill University, Montreal

Your file Votre référence
ISBN: 0-612-88296-9
Our file Notre référence
ISBN: 0-612-88296-9

ACKNOWLEDGMENTS

This thesis would not have been possible without the support of a large group of people. Thanks are in order to my supervisor Professor Doina Precup, who patiently guided my research and worked with me through some particularly difficult hurdles, whose assistance and support went beyond the call of duty.

I would like to thank the team at Netvention Inc. for their assistance both in coming up with innovative ideas and in their encouraging attitude. First and foremost, Benoit Julien, who gave me the idea for this work, and acted as supervisor in the initial stages of my research. I'd like to thank Jean-Claude Pietre for his on going technical support, Joost Overkerk for providing me with certain packages that facilitated certain functionalities of the system. I'd like to thank Piter, Michelle and Sigmond for taking the time to answer my questions, and helping me out when I needed it.

I would also like to thank Joel Martin from the CNRC, who gave me ideas, and went out of his way to provide me with relevant material, and who gave me permission to use his applications without which major aspects of this work would not have been possible. The translation of the Abstract into French was done by Danielle Azar, I would like to thank her for her help in this.

Finally, I'd like to thank my family for believing in me, and supporting me in all my endeavors. This thesis is dedicated to my father Benjamin Singer, for encouraging me to go on my way and always follow my heart, and to my Mother for always reiterating that I'll do just fine in school if I attend it.

ABSTRACT

The problem of automatically extracting information from web pages is becoming very important, due to the explosion of information available on the World Wide Web. In this thesis, we explore and compare hand-crafted information extraction tools with tools constructed using machine learning algorithms. The task we consider is the extraction of organization names and contact information, such as addresses and phone numbers, from web pages. Given the huge number of company web pages on the Internet, automating this task is of great practical interest. The system we developed consists of two components. The first component achieves the labeling or tagging of named entities (such as company names, addresses and phone numbers) in HTML documents. We compare the performance of hand-coded regular expressions and decision trees for this task. Using decision trees allows us to generate tagging rules that are significantly more accurate. The second component is used to establish relationships between named entities (i.e. company names, phone numbers and addresses), for the purpose of structuring the data into a useful record (i.e. a contact, or an organization). For this task we experimented with two approaches. The first approach uses an aggregator that implements human-generated heuristics to relate the tags and create the records sought. The second approach is based on Hidden Markov Models (HMM). As far as we know, no one has used HMM before to establish relationships between more than two tagged entities. Our empirical results suggest that HMMs compare favorable with the hand-crafted aggregator in terms of performance and ease of development.

RESUME

Le problème d'extraction d'informations de sites web est devenu très important réçamment à cause de l'explosion d'information. Dans cette theèse, nous explorons et nous comparons des méthodes crées par des personnes aux méthodes d'apprentissage automatique. Le domaine auquel nous nous intéressons est spécifiquement lié à l'extraction des noms d'organisations ainsi que leurs coordonnées (leurs adresses et leurs numéros de telephone, par exemple). Vu le nombre très élevé des compagnies qui possèdent des sites sur Internet, l'automatisation de cette procédure s'avère être d'un grand intérêt. Le système qu'on utilise consiste en deux étapes: l'étiquetage des entités prédéfinies (une "entité prédéfinie" est un mot ou un groupe de mots qui dénotent un concept spécifique dans le domaine considéré) dans les documents web et l'établissement de relations entre les différentes entités étiquetées. Nous comparons la performance des expressions régulières générées la main celle des arbres de decision dans la classification des entités prédéfinies dans les documents web. La seconde étape consiste établir des relations entre les entités étiquetées (noms, numéros de téléphone et adresses) sur des sites web, dans le but de structurer et d'orgraniser l'information d'une facon utile (un contact, ou une orgarnization). Dans ce but, nous avons conduit des expériences selon deux approches. La première approche fait usage d'un ensemble de techniques de recherche qui implémente celles performées par les humains et qui consiste relier ensemble les étiquettes et créer des ensembles d'informations qui seront requises plus tard. La deuxième approche utilise les Modèles de Markov Cachés. Selon nos recherches, ces modèles n'avaient pas été utilisés, auparavant, dans le but d'établir des relations parmi plus que deux entités. Nos resultats basés sur lexpérimentation,

suggèrent que les techniques de l'apprentissage machine performent aussi bien que les solutions désignées la main en ce qui concerne la performance et la facilité de développement.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS iii

ABSTRACT iv

RESUME v

LIST OF TABLES ix

CHAPTER

1. INTRODUCTION 1

1.1 Structure of IE systems 2
1.2 The Task Addressed in This Thesis 4

2. TAGGING 6

2.1 Related Work 8
2.2 Heuristic Approach: InfoTagger 11
2.3 Machine Learning Approach: Decision Trees 13

2.3.1 Decision Trees 14
2.3.2 Using Decision Trees for Tagging 15
2.3.3 Classification and insertion of tags into the text 19

2.4 Results 20

3. RELATIONSHIP FINDING 23

3.1 Aggregation Process 23
3.2 Heuristic Aggregator 26
3.3 Using Hidden Markov Models for Aggregation 28

3.3.1 Definition 29
3.3.2 Algorithms for HMMs 30

3.3.3 Applying HMMs to Aggregation 32
3.3.4 Training the HMMs 34

3.4 Empirical Results 37
3.5 Discussion 42

4. CONCLUSIONS AND FUTURE WORK 44

BIBLIOGRAPHY 47

LIST OF TABLES

Table		Page
3.1	Performance of the heuristic aggregator	37
3.2	Precision and recall for Address HMM with fixed emission probabilities, as a result of 5-fold cross-validation	38
3.3	Precision and recall (based on 5-fold cross-validation) for Address HMMs with different numbers of states. Both the emission and transition probabilities are estimated from data	39
3.4	Precision and recall for the Contact HMM, where the emission probabilities are fixed and the transition probabilities are learned	40
3.5	Precision and recall for an HMM with 18 states but in which both the transition and emission probabilities were trained	41
3.6	Precision and recall for an HMM with 30 states in which both the transition and emission probabilities were trained	41

CHAPTER 1

INTRODUCTION

Information Extraction (IE) is concerned with extracting domain relevant data from a corpus of documents. For instance, an IE system could be used to extract patient information from medical records (e.g. age, diagnosis, etc.). Generally, information extraction systems take free-form text documents as input and produce structured data as the output (e.g. records in a database). Traditionally, information extraction systems have been used more in research than in actual daily operation. The development of IE systems during the 1990s was stimulated by the yearly Message Understanding Conferences (MUC), sponsored by DARPA [5, 6, 4]). Tasks proposed for the MUC conference included:

- **Health care delivery:** summarizing medical records by extracting diagnoses, symptoms, test results and treatments; such systems can provide doctors with easy access to information regarding their patients, and also assist in statistical and medical research.

- **Monitoring technical and scientific literature:** IE systems were used to analyze technical articles in the domain of microchip fabrication, in the areas of layering, lithography, etching and packaging. These systems helped advance the industry by providing a good knowledge base for research, manufacturing and distribution of equipment.

- **Intelligence gathering:** IE systems were used to summarize information about terrorist activities from news articles. The specific information gathered included terrorist organization, location, and casualties. IE systems are

also used in this domain by government agencies to identify potential threats by monitoring certain web sites and the people who access them.

- **Business intelligence:** Business newspapers were analyzed using IE to find information about major business events such as buyouts, mergers, and joint ventures. The goal of the IE system was to find out who the parties involved were, the purpose of the event, and the amount of money involved.

In the last five years, because of the information overload, industrial interest in information extraction has grown a lot. IE systems have been implemented and are currently used in research (e.g., finding computer science papers [11], extracting information about proteins [10]) as well as in everyday business. For instance, Whizbang Labs has set up the (allegedly) largest collection of job advertisements on the Internet, by automatically extracting relevant information from web pages.

In order to obtain good performance, IE systems are designed to work within a certain domain, to which they are dependent to a large extent. Therefore, applying an IE system to a domain of interest different from the one it was originally designed for requires additional work. The extent of the work required to change domains depends on the design of the system, and can vary from the construction of new training data (if the system uses machine learning for producing the extraction rules) to completely rewriting various modules (if the system is hand-crafted). In this thesis we present an IE system that retrieves organization names, and their relevant contact information (person names and titles, phone number, address, etc.) from web pages. Our focus is on comparing machine learning techniques to hand-coded solutions, in terms of performance quality and ease of design.

1.1 Structure of IE systems

The process of information extraction (IE) from the World Wide Web (WWW) involves three steps [15]. The first is the harvesting of web pages that are related to

the topic of interest. This is usually done by a standard search engine and will not be discussed in this work.

The second stage is not specific to IE from the WWW, and is that of tagging named entities in the HTML/text document. Named entities are words or phrases that have been identified by the designer of the system as being of interest. For example, for a system collecting research papers, the named entities would be title, author, date, venue where the document was published etc. [9]. In general, the domain can be very structured, and thus easy for information extraction, or it can be unstructured, which makes IE difficult. For example, phone numbers or postal codes are very structured, while person names or symptoms of an illness are unstructured. In a structured domain, it is easy to find generally applicable rules to extract information. Hence, hand-crafted approaches are adequate. In a very unstructured domain, with a lot of variability, rules are much harder to find.

Tagging can be achieved by a multitude of methods: word dictionaries, hand coded regular expressions or heuristics, various machine-learning techniques, or an ensemble of these methods (see, e.g., [9, 16, 3]). Word dictionaries are lists of words and their meta-definition. An example of such a dictionary would be a list of country names. When a word from the dictionary (e.g., Canada) appears in the text, it is identified as being a country. Regular expressions and heuristics are used to either give context to words identified by the dictionaries, or match patterns that are very likely to occur. An example of using context to label words properly would be the following rule: if the word London appears preceded by another capitalized word, then it is labeled as a surname, and not a city (e.g. John London). An example of using the structure of the text would be the labeling of postal codes in Canada, by defining a pattern: letter, number, letter, optional space, number, letter, and number.

The third stage of an IE system involves the grouping of labeled/tagged named entities to form structured records containing the information we are trying to extract.

This can be done either by creating a dictionary of word patterns that might establish a relationship between two labeled named entities, such as a typical pattern of words that relates an organization and its location [1], or by establishing heuristics that identify such a relationship [7, 8]. We will discuss tagging and relationship finding in greater detail in the following chapters.

1.2 The Task Addressed in This Thesis

The task that we are addressing in this thesis is the extraction of corporate information from web pages. Our system uses a labeling step followed by an aggregating step, as described above. For instance, for the labeling step we will analyze a web page where we may label Coca-Cola as an organization name, Mike Anderson as a person. 12 St. Atlanta as an address, (614) 876-4532 as a phone number, and Microsoft as an organization name. Following this, we will have to establish the relationship between the proper entities, and extract the information in a record. It may be established that one of the named entities, for instance Microsoft, is irrelevant, and thus does not belong to the relationship. In this case, the named entity should be left out of the record produced.

In our work, we use both machine learning and hand-crafted techniques to perform IE. Labeling is produced using existing software, the Infotagger system, developed by Joel Martin at NRC, which uses both regular expressions and word dictionaries to label data. Initially, we wrote a number of regular expression for Infotagger based on common-sense heuristics. Because we discovered that these regular expressions were insufficient for getting good results in certain cases (e.g., tagging organization names), and because the process of creating such regular expressions was very tedious and time consuming, we used an algorithm for decision tree construction, C4.5, to generate additional rules. These new rules made the system more robust and improved

performance, and at the same time were faster to construct. These two approaches will be discussed in Chapter 2.

For establishing relationships between relevant labeled named entities, we used two methods that will be explained in detail in Chapter 3: a hand-crafted aggregator and a Hidden Markov Model. The hand-crafted aggregator uses a distance heuristic to establish relationships. The HMM bases the relationships that it establishes on a natural probabilistic model learned from training data. We compare the performance of these two systems, and we also study the behavior of the HMM technique under different parameter settings.

The thesis is structured as follows. Chapter 2 discusses tagging and provides an overview of related work on classifying documents. We provide a detailed description of the methods that we used in this work to solve the labeling problem. Chapter 3 gives an overview of the work done in the field of relationship pattern matching, as well as a detailed description of the hand-crafted aggregator, Hidden Markov Models in general, and the particular Hidden Markov Model that we used in this thesis. Chapter 4 contains conclusions and avenues for future work.

CHAPTER 2

TAGGING

Tagging, also known as *named entity recognition/classification* or *labeling* [6], is the process of identifying *named entities*, i.e., words or phrases of interest in a given domain (such as person names, locations, phone numbers etc.). A word is a sequence of characters delimited by predefined separators (such as, e.g., spaces, tabs, punctuation marks, or HTML tabs). A phrase is a sequence of two or more words. Tagging is generally the easiest and most successful sub-task of information retrieval systems. However, the performance of taggers is still very dependent of the domain at hand.

Tagging requires the ability to define the class of a word and couple this knowledge with the context in which the word appears. By class we mean, for instance, whether a word is a country or a person's first or last name. Defining the class of a word can be achieved using word dictionaries. A dictionary is a collection of words of a certain type (e.g., country names, or company names). The collection can contain just the words, or additional information as well (such as the definition of a word). Even if we can define such dictionaries, ambiguities can still arise when certain words are defined in more then one dictionary. For instance, a person's last name may also be the name of a country. In this case, the context of the word has to be used in order to disambiguate between its possible classifications. Finally, certain classifications cannot be easily identified by using word dictionaries, because the dictionaries needed would be too extensive. In some of these cases, such as phone numbers, postal codes or street addresses, there is a lot of syntactic structure in the named entities being sought. If this is the case, syntactic patterns (e.g., regular expressions) can be defined in order

to achieve proper tagging. In other situations (such as seeking titles of computer science papers, for example), simple patters are not sufficient and more sophisticated methods, using both patterns and context, need to be used.

People are of course very good at tagging, largely due to the fact that they have large word dictionaries, and a lot of background knowledge regarding possible patterns that can appear in text. For instance, it is easy to recognize John Smith as a person's name, if we know that John is a first name and Smith is a last name. Similarly, identifying 12 Tapuz road as a street address is easy by using the suffix "road". Now consider a situation in which the text was in a foreign language that used Latin characters, but in which we could not understand the meaning of the words. Would we know that Xhi Yan was a name, and that avienda Tapuz 12 was a street address? We would need to know some rules that are followed by the foreign language, and which would allow us to infer the proper labels of the named entities in the examples above.

Most programs that perform tagging use the same basic tools (dictionaries and pattern matching performed on the context of a word) in order to perform their task. A tagging application needs to be told about rules of the language or domain of interest in order to make the proper inferences, just like a human would. For example, the application can be given a dictionary of street suffixes, along with a regular expression pattern matcher. The regular expression engine could infer, for instance, that if a street suffix appears in the text, then all capitalized words before it are part of a street name, and the first number encountered before the street name is the civic code of the building. This approach works very well in structured domains, but is very difficult to implement for unstructured types of information, because rules are difficult to formulate. Hence, a lot of recent research focuses on machine learning techniques, which can extract rules automatically from a corpus of documents.

In this chapter, we compare machine learning and traditional approaches to tagging. Section 2.1 summarizes a couple of successful pieces of related work. Section 2.2 describes a hand-crafted tagger based on regular expressions and word dictionaries. Section 2.3 describes a machine learning approach, based on decision trees. In section 2.4, we show how the machine learning approach can supplement hand-crafted rules in order to achieve better results.

2.1 Related Work

In some languages and domains of interest, the words may have certain textual characteristics that offer hints regarding their meaning (e.g., capitalization, bold or italics font in HTML documents, etc.). The presence of such characteristics greatly facilitates tagging. Certain languages contain a lot of textual and syntactic structure, which makes them good candidates for IE systems. For instance, Japanese is a very structured language, in which different types of script are used to write different types of words. Sekine et al. [16] successfully used machine learning in order to build a Japanese morphological analyzer. The system first performs part-of-speech tagging. Then the part of speech, the character type used in the text for the given word, and specific word dictionaries are used to define attributes for every word. These attributes are used to construct input instances for a decision tree, which returns the classification for each word. The definition of the input attributes for the decision tree is specific to the Japanese language. However, the system could be modified for use in English, if an interested party would create language-specific attributes of the same sort. However, this task would be very time-consuming, since the system depends heavily on word dictionaries that have to be manually constructed. At the same time, one would expect that the performance of a similar system used for English text would be significantly worse, given the reduced amount of structure and the large number of irregularities and ambiguities that is typical for English.

One problem that could occur with this system is that, because decisions are made locally for each tag, there could be inconsistencies on a global level. In order to solve this problem, the authors use two approaches. In the first approach, a heuristic tag priority scheme is defined, which is then used to modify certain sequences of tags. In the second approach, each leaf of the decision tree is allowed to output a probability distribution over tags, rather than just a single tag. Then, taking into account the previous, current and following tag, a Viterbi-style algorithm is used to calculate the highest global probability for the string of tags while maintaining a constant tag type throughout the string. This technique is very useful when constructing larger tagged entities (e.g., company names) out of several tagged sub-entities (corresponding, for instance, to each word).

The system was trained with 100 manually tagged articles. The average accuracy over all the named entities was relatively high, reaching 85% when the system was trained and tested in the same domain. Curiously enough, decreasing the size of the training data to 9 articles only reduced the average accuracy to 79%. This is an indication that the manually constructed dictionaries actually play an important role in the performance. One would anticipate the performance of the system to degrade considerably if the system was tested on a corpus from a different domain than the one it was trained on.

A different machine learning approach to tagging is based on using Hidden Markov Models (HMM). Unlike the previously described system, the HMM approach does not require the language to be structured in any way. It does require, however, that the application domain be known in advance. HMMs (which will be discussed in detail in the next chapter) are computationally efficient and fairly easy to implement. In addition, because HMMs rely on statistics, they can be constructed easily, and with little prior knowledge, as long as large quantities of documents are available [9].

An example of a tagging system based on HMMs is the Cora engine, developed by McCallum et al [11]. Cora's goal is to tag the words that appear in the header of computer science papers as belonging to one of the following categories: title, author, address, note, affiliation, email, date, abstract and body. Two of these categories, email and date, can be easily tagged using regular expressions. For the other categories, the authors experimented with four types of HMMs: a fully connected learned model, a hand-crafted learned model, an M-merged model, and a V-merged model. The best fully connected model achieved a maximum accuracy of 64.5%. The hand-crafted model had learned parameters, but the structure of the HMM was designed with the domain in mind. Multiple states per possible named entity were allowed, and the connectivity was based on the observed structure of the data. This model achieved a significantly better accuracy, 92.4%. In the M-merged model, each word in the training data was assigned its own state, which in turn was associated with the category of that word (i.e. person name, affiliations, etc.). Once this was done, connected states with the same class were merged together, and a self-transition loop was introduced. The M-merged model achieved an accuracy of 92.9%. The V-merged model merged any two states that had the same class, and that share a neighbor. This model achieved an accuracy of 92.7%. The success of merging is important because this technique allows learning the structure of the model directly from the data. Merging can be accomplished via a technique like Bayesian model merging, which seeks to maximize the probability of a model given the training data.

Another system that uses HMMs for tagging text is the Nymble system [2]. Nymble uses both word identity and word features (capitalized, containing digits, etc.). Word identity is established by using a large corpus of labeled data: 100000 labeled words (about 750 pages). This amount of training data is quite expensive to obtain, but the authors claim that it is still reasonable to handle when switching domain. However, even though no word dictionary is used, it is clear that the large

amount of labeled data would make this system harder to port to different domains than Cora, which requires only approximately 100 labeled pages for training. In terms of the structure of the HMM, Nymble uses a single HMM for all named entities of interest. There is a single state for every named entity, and the structure is fully connected. A special feature is that the transition and emission probabilities are connected not only with the state, but also with the previous observation.

Both Cora and Nymble are HMMs used for tagging, but their design is quite different, and leads to different trade-offs. Cora relies heavily on the relationships between words in the text, where as Nymble relies mainly on features of the individual words. The two system also have different trade-offs in terms of their requirements. Cora extracts information from very structured documents, and this allows it to use a fairly small training corpus. Nymble, on the other hand, uses free-form documents, but requires significantly more human-labeled data.

2.2 Heuristic Approach: InfoTagger

Our heuristic tagger is built using as a basis the InfoTagger system, developed by Joel Martin at NRC. InfoTagger takes as input the contents of a web page and a list of labels, and returns an HTML document in which named entities have been tagged. InfoTagger uses a user-defined word dictionary, a user-defined regular expression dictionary and a general inference engine, which uses the two dictionaries in order to accomplish the task at hand. In a first phase, InfoTagger uses the word dictionary and attempts to find every word from the document in the dictionary. For every match identified, the label corresponding to the class of the word is inserted into the text. For example, if the named entity England appears in the text, and this word is found in the dictionary under the classification Country, then the label `<Country>` will be inserted into the text. Note that if England also appears in the word dictionary under the classification LastName, then the `<Last Name>` label will

be inserted as well. Once this phase has been completed, a regular expression engine processes the tagged text.

The regular expressions that we provided were aimed toward two tasks. The first was the identification of simple patterns, such as emails and phone numbers, that were not covered in the dictionaries. The second task was to solve ambiguities, in the case in which the same word had more than one classification (like in the example above). In the second case, the regular expressions are based on the context of the word.

Our system contains 11 regular expressions, ranging from very simple ones (e.g., Email) to complex ones (e.g., Organization Name and Street Address). The time needed to evaluate a regular expression depends on the number of different branches that the corresponding finite state machine can take. If the automaton has a long sequence of branching points, each with many possible branches, then evaluating the regular expression over an entire page will take longer. As a result, in our system, tagging a single web page can take up to 5 seconds, if we look for organization names, or just 0.1 seconds, if we are looking for postal codes.

InfoTagger is capable of handling multiple threads, for tagging multiple documents. It takes regular expressions written in Perl syntax, and translates them into non-deterministic finite state machine automata (NFA). The performance of an NFA degrades exponentially with the increase in complexity of the regular expressions. In the future, we hope to use deterministic finite state machines (DFA) instead. The advantage of using a DFA is that regardless of the complexity of the regular expression, the upper bound on the execution of the regular expression through a DFA is the longest path through the DFA. In this way polynomial and even linear performance can be achieved.

2.3 Machine Learning Approach: Decision Trees

Writing the extraction patterns (regular expressions) is a complicated iterative task, which is extremely expensive in terms of time, and possibly money. In essence, a programmer needs to write the regular expression, test it, see where it fails, and then tweak it to improve the performance of the system.

Instead of refining the regular expressions by hand, a machine learning approach can be used to generate either regular expressions or extraction rules that may otherwise be missed by human programmers. Automating this tedious task can be beneficial both in terms of cost and in terms of the performance of the system. Our approach was to use the C4.5 algorithm for learning decision trees [13] in order to generate extraction rules that could latter be applied directly, or converted into regular expressions.

In this work we focused on learning to identify organization names in HTML documents. We chose to learn the organization names because this is the only kind of named entity in our domain for which a dictionary cannot be used, due to the diversity and large number of names. For other named entities, such as countries or even person names, dictionaries of reasonable size are readily available and they are sufficient to achieve acceptable performance.

Generally, organization names are easy to discern due to context. For example, Yaya Inc. is easily identifiable as an organization, because of the suffix "Inc" that is part of the name. However, we also encountered very often (approximately 50% of the time according to our data), situations in which suffixes are not added to organization names. In these instances, additional criteria must be added to the labeling system in order to be able to determine that a particular word or phrase is indeed an organization. This makes the development of regular expressions very difficult. We now describe decision trees in general, and our approach to using decision trees for classifying organization names.

2.3.1 Decision Trees

Decision tree construction is an inductive inference learning method for approximating discrete valued functions [12]. The functions learned are represented either as trees, or as if-then rules. Instances are classified by sorting them down the tree from the root to a leaf, where each internal node (including the root) represents an attribute, and the branches represent possible values of the attribute. An instance (example) is classified by starting at the root, testing its value and then transcending down the branch that corresponds to the value of the attribute to the next node in the tree. This process continues until a leaf is reached, at which point the tree generates a classification.

Ideally, the attributes in a decision tree should be ordered from the most important at the root, to the least important ones, further down the tree. The importance of an attribute is usually defined by the information gain that is achieved by using that attribute for classification. Information gain is a statistical property of an attribute that shows how well any given attribute separates the training examples according to the target classification [12]. A lot of related measures have also been developed in the decision tree literature, all with similar results [12].

Decision trees appear to be a good machine learning technique for this problem for several reasons. First, the rules induced by the decision tree are easily understood by humans, and can be (almost) directly incorporated into any rule-based labeling system. Secondly, decision trees can be learned off-line, before the information extraction system is operational. Off-line learning is the preferred approach for this application, because it would be very difficult to come up with a proper supervision scheme with which to teach the system online. Third, decision trees are fairly robust to noise and their performance is comparable to other classification techniques [12]. Finally, the training set required is relatively small, and thus cheap to obtain. This

is especially important in our case, because the training set has to be generated by a person tagging documents by hand.

2.3.2 Using Decision Trees for Tagging

In order to learn decision trees for tagging organization names, we need to identify elements of the HTML document, besides the organization suffix, that can predict that a word or phrase is an organization name. After looking at several web pages, we decided on using the attributes described below. These attributes are Boolean unless otherwise noted.

- InCaps: the word is capitalized
- InBold: the font used is bold
- InEmail: the name appears in the domain of an email address. We allowed some leeway as to the exactness of the match. For instance, if the organization name is Nortel Networks and the email is hr@nortel.com then a match would still be positive. This attribute is continuous-valued.
- InLink: similar to InEmail.
- InTitle: does the phrase appear in the area marked up as <TITLE> in the HTML document.
- AdjToLocation: is the word at a distance of up to three words before a location (e.g., Shell Canada)
- AdjToOrgSuffixStrong: is the word at a distance of up to three words before an organization suffix that provides a strong indication of the phrase/word being an organization (e.g., inc, ltd, etc.).

- AdjToOrgSuffixWeak: : is the word at a distance of up to three words before an organization suffix, that provides a weak indication of the phrase/word being an organization. (e.g., Corporation, etc.).
- InCell: does the phrase appear in a cell of an HTML table, marked by <TD>.
- InList: does the phrase appear in an HTML list, marked by <LI>.
- NoDef: true for words that are not in capital letters.
- OrgName: the classification of each example, either yes or no (i.e. this phrase is an organization, or not).

All organizations in the corpus were manually tagged. We wrote regular expressions for computing each attribute. These regular expressions were applied to the text using InfoTagger. For example, the text "in the last quarter Bell Canada inc. reported profits", was tagged as follows:

```
<NoDef>in</NoDef> <NoDef>the</NoDef> <AdjToLoc><NoDef>last<NoDef/>
<NoDef>quarter</NoDef> <OrgName1><AdjToOrgSuffix><InCaps>Bell</InCaps>
<InBold><InCaps>Canada</InCaps></InBold></AdjToLoc>
inc.</AdjToOrgSuffix></OrgName> <InBold><NoDef>reported</NoDef>
<NoDef>profits</NoDef></InBold>
```

Note that the OrgName tag was inserted manually, as it is the classification action.

Once the attributes were tagged in the text, we extracted all the tags with the TagExtractor module, producing a list of tag objects. Each tag object maintains the tag type, its start and end indices in the text, and the text content itself. Because the tag objects are independent, and overlapping of indices exists between different tag objects, multiple occurrences of a single-named entity may appear in the tag list (once for each tag that overlaps the named entity). In order to capture all valid attributes for every given named entity, we compressed and converted the tag list

into an Instance list, where the information from overlapping tags is used to generate an Instance object.

The instance object represents the data that will be used by the learning algorithm. For this application, having one instance for every word, and only for words is not a good idea, since it does not allow considering the context of a word. Using phrases of a fixed length is also not feasible, since organization length can vary in size. Therefore, we adopted a heuristic approach, in which instance generation is guided by the existing tags.

Initially, one instance object is generated for each tag. The instance object maintains all the information from the tag, but in addition it has a list of attributes. The value of each attribute in the list is set with respect to the examined named entity. For example, if a named entity were tagged as being capitalized, then an instance would be produced where all attributes are set to no (or default values) and the InCaps attribute is set to yes. When an instance is created, it inherits all the information from all the instances that overlap it. For example, in the above text fragment "Bell Canada Inc." the instance corresponding to the word Bell, would have two attributes set: InCaps corresponding to the word Bell, and AdjToOrgSuffixStrong which corresponds to the entire text fragment. Similarly, the instance corresponding to the word "profits" in the tagged text would have the InBold Attribute set, as the InBold tags overlaps the phrase of which the word "profits" is a part of.

Only instance objects corresponding to the InCaps, NoDef, and AdjToOrgSuffixStrong/Weak, InLink and InEMail are created (InBold, InCell, etc. are not). Once all the instances have been created, instances with overlapping indices are merged. For example, in the tagged text above, three of the initial instances corresponding to three attributes would be AdjToOrgSuffixStrong containing the text Bell Canada inc., InCaps containing the text Bell, and InCaps containing the text Canada, which would have the InBold attribute set as well. Given this setup, the instance corresponding to

the AdjToOrgSuffixStrong attribute would become a union with the information from instances of the Bell and the Canada corresponding instances, and those instances would be removed from the list. In other words, one instance would remain which would have the AdjToOrgSuffixStrong, InCaps, and InBold Attributes set to yes, and the contents of the remaining instance would be Bell Canada Inc. The instances that are enclosed by an OrgName form the positive examples, while the instances that were not overlapped by an OrgName tag form the negative examples.

As discussed in detail below, we experimented with keeping the instances at word level, as well as allowing for phrase instances. Originally, we conjectured that sometimes only part of a phrase enclosed by a tag may be an OrgName, and therefore classification would be more accurate if we kept the instances as atomic as possible. In this case, adjacent words that were classified as being organizations would be aggregated together into a single organization name. This approach showed good results for classification by C4.5, but was not as successful when actually inserting the tags into the text.

Our second approach allowed the creation of instance objects for tags that enclosed phrases in the text. Initially, we created instances for all tags. This however, proved to reduce performance, as some tags may encompassed very long phrases (e.g. InCell and InBold), and may also encompass many other tags that are mutually exclusive. Since phrase instances contain all the information from component word instances, inaccuracies are bound to occur. For example, if the word Canada is in bold but the word Bell is not, the entire phrase would still have the Inbold attributes set to true. Another example is the phrase: "Barney loves children". This phrase would be tagged as:

```
<InBold><InCaps>Barney</InCaps> <NoDef>loves</NoDef>
<NoDef>children</NoDef></InBold>
```

which in turn would form an instance object corresponding to the entire phrase where both the InBold and the InCaps attributes would be set. This setting would not be a proper reflection of reality as only one word in the phrase is actually capitalized. Such occurrences would bring about the creation of instance objects that would confuse the classifier. After noticing this phenomenon, we restricted the number of tags that were used to create phrases. This allowed for a good trade-off between the gain in performance due to allowing phrases, and the imprecision introduced.

We also experimented with a system where overlapped instances were removed from the instance-list only when the overlap was exact (i.e. the text start and end indices were equal). This approach, however, skewed the precision and recall measured by C4.5. The cause of this skewing was that most of the time phrases rather than words turn out to be organizations. Therefore, multiple instances for each OrgName appear in the instance list (i.e., one instance for the phrase and one for each word in the phrase). This multiplicity causes the weight of the positive examples to become heavier, and as a result the classification as well as the count of correct classifications became distorted.

2.3.3 Classification and insertion of tags into the text

The classifier module built by C4.5 consists of a set of rules that can be applied to instances, and produces class labels for those instances. However, once labels are known, OrgName tags still need to be introduced in the text in order to achieve the purpose of the system. This step is done by a greedy algorithm, which attempts to aggregate as many instances as possible in order to create an organization name.

For example, a negative instance whose content is a stopword, but which appears between two positively labeled instances, will also be deemed part of the organization name. This design decision ensures that organization names will not be incomplete in cases such as, e.g., A. Gold and Sacs., where the stop word "and" would probably

not be classified as being part of an organization name, resulting in two separate organizations. One draw back of this design decision is that if there are two consecutive organization names that may or may not be separated by a stop word, they would be aggregated into one organization name. For example, the phrase "yesterday Microsoft Inc and Apple Corp. announced...." would generate `<OrgName>Microsoft Inc and Apple Corp.</OrgName>`. This design also results in the possibility of misclassifying words that precede an organization name as being part of the organization. Unfortunately, this is a frequent phenomenon, because instances in a close vicinity tend to share the same attribute values (e.g., InCell, InBold, AdjToLocation, etc.) which means that they often are classified similarly. However, we felt that in this particular application, getting too much text is better then not capturing a complete name, since incomplete organization names would be quite useless to end-users.

2.4 Results

In order to measure performance of the tagger, we use precision and recall, which are two standard metrics from the information extraction literature[15]. Precision is defined as the ratio of the number of correctly tagged named entities out of the total number of named entities tagged. Recall is defined as the ratio of the number of correctly tagged named entities out of the total number of instances of named entity of the type that was tagged in the corpus. Ideally, both precision and recall should be close to 1.0. However, one measure usually decreases as the other increases.

We trained the system on a corpus of 100 web pages, which produced approximately 17000 instances, 273 of them being positive (i.e., organization names). Instances were either words and/or phrases depending on the experiment. We used ten-fold cross validation to generate rule-sets [13, 12]. Each page was tagged by the system based on a rules set learned by C4.5. After that, an evaluator was used to compare the machine tagged data to hand tagged data. We present two types of

results: the precision and recall as computed by C4.5 on test data, and the precision and recall measured by the text evaluator after the <OrgName> tags have been inserted. As seen below, there is a discrepancy between these two measures. This is due partly because of our design decisions regarding the tag insertion mechanism, and partly to a difference in what is being measured.

First, we evaluated the precision and recall of the hand-crafted system. Precision was 66.9% and recall was 51.9%, using the same folds as the decision trees. However, we would like to point out that the hand-crafted system was tuned to roughly 70% of the files under consideration. When evaluating the system on the files that we considered when designing the regular expressions, precision and recall were 90% and 62% respectively. On the other files, these figures dropped drastically, to 59% and 33% respectively for precision and recall. This makes us believe that the performance of the hand-crafted system would be even worse if it was used on new pages.

In the second experiment, we used decision trees constructed by using instances that are limited to individual words only. In this case, the precision and recall evaluated by C4.5 on training data were of 77.8% and 68.9%, while on test data they dropped to 65% and 62.1%. The estimates of the text evaluator were slightly worse: 70.4% and 54.9% on the training data, and 54.1% and 46.8% on the test data. It is worth noting that in this system, the post-processing phase, during which the tags are inserted, proved to be especially harmful. This is due to the fact that no phrases are allowed, and the aggregation of words into organization names is hence based on very weak information.

In the third experiment, we used both words and phrases as instances. In this case, the precision and recall measured by C4.5 (averaged over the folds) were of 74.7% and 54.7% on the training data, and 63.8% and 45.7% on the test data. However, the performance reported by the evaluator is in fact better in this case: 80.2% and 57.1% for precision and recall on the training data, and 68.2% and 51.9% on the test data.

The classification done by C4.5 indicates that allowing phrases degrades the performance. This seeming degradation is due to the fact that when no phrases are allowed, there are more positive instances. Instances benefit from the information that is inherited from tags that encompass them, and as a result the word-based instances are still likely to be classified most of the time the same as if they were part of a phrase (excluding stop words). Due to these two facts, the proportion of correctly classified instances is likely to increase, while the actual number of organization names that were classified correctly does not actually change.

Overall, however, the performance increased as a result of allowing instances to include phrases. This confirms our initial assumption that phrases are more robust, as they contain more information. Also, allowing phrases to form instances means that fewer instances need to be aggregated, which in turn means that there is a smaller likelihood for errors that result from our particular greedy strategy for aggregation.

We also want to emphasize the fact that, although overall the performance of hand-crafted and machine learning systems are roughly the same on the average, the hand-crafted system seems particularly tuned to the documents that were used during its construction. The learning systems seems a lot more robust in this respect: although its performance does degrade on unseen data, as expected, the reduction is not nearly so drastic. This makes us suspect that the learned tagger would be significantly more useful in practice.

It is also worth comparing the two systems in terms of development time. The hand-crafted system took approximately three months to construct, while using learning allowed achieving similar results in a much shorter time period (approximately three weeks). Also, the process of constructing a detailed regular expression for the hand-crafted system was an extremely painful iterative process, which was mostly avoided by using machine learning. Hence, we feel that there are good reasons to use learning for named entity recognition.

CHAPTER 3

RELATIONSHIP FINDING

We will now focus on the third step of our IE system: creating a relationship between different named entities in the text. Them goal of our system is to create a business directory from the web. To do this, we must be able to establish relationships between different named entities on web pages, such as company names, addresses and phone numbers. Finding these relationships is also referred to as entity-relation recognition [6]. The process consists of filling out pre-specified patterns, with the named entities identified in the text. The implementation details of this step are discussed in Section 3.1. The difficulty of this step consists in the fact that more that one named entity of the desired type is usually present on the web page. For instance, a company may list its address as well as a list of partners on the same page. How can we identify which of the company names should be associated with the address? The naive approach would be to use adjacency information, as well as the specific order in which the named entities appear in the text. This is the basic idea behind the first approach to solving the problem, presented in Section 3.2. This approach is based on adjacency heuristics, and human knowledge of the problem. In Section 3.3, we present an alternative approach, based on Hidden Markov Models (HMMs). In Section 3.4, we present empirical results comparing the performance of the two approaches.

3.1 Aggregation Process

The goal of the aggregation process in our system is to build organization, contact and address objects from HTML documents. There are two kinds of objects that

we are going to manipulate: basic objects (corresponding to named entities), and composite objects (corresponding to the desired patterns that we want to fill out). The tagging process, described in the previous chapter, inserts simple tags into the text, such as Email, Phone, StreetAddress, etc. Once tagging is complete the TagExtractor module takes as input the tagged page, extracts the tags from the page and stores the relevant information in tag objects, which in turn are inserted into an ArrayList. Each tag object contains the tag type (e.g., Email, Phone, etc.), the start and end indices in the original text, and the actual text contained within the tag. The TagCollector module takes the ArrayList containing the tags in order, and constructs from it separate ArrayList objects for every tag type. This allows tags of particular kinds to be rapidly accessed.

Composite objects are created during the aggregation process, based on the basic tags already assigned during the named entity recognition phase. For example, the street address, city, and region basic objects (tagged previously) can be used to compose the Address object. The construction of composite objects always starts with a seed basic object. Then, other basic objects are added, to the composite object, depending on the aggregation rules. The fact that the TagCollector creates lists with all the tags of specific types allows seed objects to be located quickly. For example, the seed for building an Organization composite object is an OrganizationName tag object. The TagCollector will use the ArrayList containing OrganizationName tags to locate a seed organization. Afterwards, contact information will be added to the Organization object, based on the specific aggregation rules used.

An object, whether basic or composite, can only be used in the assembly of a single composite object. In our system, we have three types of composite objects: Address objects, Contact objects, and Organization objects. These correspond to the three kinds of templates we need to fill out.

Address objects can be defined by one of the following patterns:

- StreetAddress, City, Region (optional), Country, PostalCode (optional)
- (for U.S. addresses) StreetAddress, City, Region, PostalCode (optional)

An Address object is valid if all this information can be filled out. Otherwise, there is not enough information for the address to be useful, and the object is deleted.

A Contact is a composite object representing a contact person within the organization (e.g., sales representative, production manager, etc.). A valid Contact must follow one of the following patterns:

- PersonName, Email, Phone, Fax (optional), PersonTitle (optional)
- PersonName, Phone, Fax (optional), PersonTitle (optional)
- PersonName, Email, Fax (optional), PersonTitle (optional)

An Organization is a composite object that must have an organization name, and this name must be related to either an address object or a contact object, or some other contact information. Organization objects are valid if they satisfy one of the following patterns:

- OrganizationName, Address, Contact (Phone or Email are also accepted)
- OrganizationName, Address
- OrganizationName, Contact (Phone or Email are also accepted)

Composite objects are built in a bottom-up manner: first the Address and Contact objects are constructed from basic tag objects, then Organization objects are constructed. As an illustration of this process, consider the excerpt below, from an HTML document that was tagged:

```
<TD width="33%"><FONT color=#0000ff face=Arial size=4>
<STRONG><PersonTitle>Advertising Manager</PersonTitle>:</STRONG>
</FONT><FONT color=#000000 face=Arial><STRONG><BR>Mr <PersonName><FirstName>Anthony T
</FONT><FONT color=#0000ff face=Arial size=3><STRONG><Phone>(0414)
       788-900</Phone></STRONG></FONT><FONT face=Arial size=2><BR></FONT><FONT
       color=#800000 face=Arial size=3><STRONG>Fax:</STRONG></FONT><FONT
       face=Arial size=3> </FONT><FONT color=#0000ff face=Arial
       size=3><STRONG><Fax>(03) 9432 1177</Fax></STRONG></FONT><FONT face=Arial size=
<P align=center><IMG height=5 src="A5A_files/Turquoise_and_Gray6043.gif" width=536></
```

The composite objects that would be extracted from the text are the following:

- Address1: StreetAddress(P.O. Box 1117), City(Bundoora), Region(Victoria), Country(Australia), PostalCode(3083).
- Contact1: PersonName(Anthony T. Schmidt), Address(Address1), Email(ats@epcgroup.co Phone((03) 9432 1166), Fax((03) 9432 1177), PersonTitle(Advertising Manager).
- Organization1: OrganizationName(EPC Audio-Visual), Contact(Contact1).

Above, we use the notation Object(Content) to denote the type of the objects involved and their textual content. The objects are generated in the order in which they are listed above (Address, then Contact, then Organization). The construction of the composite objects is a greedy process, which means that even though an email by itself might suffice to create a valid contact, if there is other information available (such as phone or fax, in the example above), we prefer to include that information as well.

In the next two sections, we described two different approaches for constructing composite objects: a heuristic approach, and an approach based on Hidden Markov Models. Both approaches use the data structures and composite objects described above.

3.2 Heuristic Aggregator

The creation of composite objects always starts with a seed basic object. Seed basic objects are of type OrganizationName for Organization objects, PersonName for Contact objects, and StreetAddress for Address objects. Seeds are found by looking at the ArrayList objects for tags of the corresponding types. Once a seed is found, a composite object is created, and the aggregator attempts to fill out its fields. Only objects whose indices are greater than that of the seed may be added to the composite

object. As noted above, the aggregator first constructs Address objects, then Contact objects, and finally, Organization objects.

The main idea behind the algorithm is to use adjacency information in order to determine which objects should be aggregated together. The algorithm maintains an *object boundary*, which corresponds to the ending index of any tag included in the object. Every time a tag is added to a composite object, the boundary of the object is expanded to include the new tag. Only tags that start within a pre-specified maximum distance from the object boundary can be added. For example, suppose that a Contact object is currently being constructed, and that currently its start index is 100, and its end index is 180. suppose that the Phone slot of this contact object is still empty. If the next basic object on the general tag list is a Phone object whose start and end indices are 190 and 200 respectively, and if the maximum distance allowed for aggregation is set to 10, then the Phone object will be added to the Contact object. The new boundary of the Contact object would have the start index set to 100 and the end index set to 200. The maximum distance is a parameter f the algorithm, and we used exploratory experiments in order to set it.

The detailed outline of the aggregation algorithm is given below:

1. Identify a basic seed tag

2. Initialize the boundary of the object to the start index of the basic seed tag.

3. Use the templates to identify what tag objects are necessary in order to create a valid composite object

4. For each type of tag needed, do:

   (a) Search in the ArrayList corresponding to this type of tag for tags that occur after the seed. If such tags can be found, compute their distance to the boundary of the object.

(b) Take the tag closest to the boundary of the object. If this tag is within a maximum distance from the boundary:

i. Add the tag to the composite object

ii. Update the boundary of the composite object to include the new tag.

(c) Continue for the next type of tag

5. If a composite object was not completed, backtrack, as long as there are tags within the required maximum distance to the object

6. If all the required slots of the object have been filled, check if optional slots are available. If so, fill them in the same manner described above.

7. If a valid composite object was created, add it to the appropriate ArrayList, and to the basic tag list. Remove all the tags used by the new object from the general tag list.

The algorithm described above is based on the empirical observation that in general, company web pages tend to cluster contact information together. This algorithm is entirely domain-specific, and we would not expect it to work well for other information extraction tasks.

3.3 Using Hidden Markov Models for Aggregation

Hidden Markov Models (HMMs) have initially emerged from the speech processing community as a useful tool for performing speech recognition. The main intuition was that we observe a set of output symbols (sounds) produced by the speaker. The speaker has an internal model of what they want to say. This model stochastically determines the observations. HMMs are a formalization of this idea. HMMs have been very successfully applied to speech processing for many years. With the growth of the Internet and the information overload that we are facing, HMMs have, in

recent years, been applied to various aspects of information extraction. We will now introduce briefly the theory of HMMs, following the main ideas of Rabiner's tutorial on the topic [14]. Then we will describe how we applied HMMs to construct an aggregator for our task.

3.3.1 Definition

A *Hidden Markov Model (HMM)* consists of a finite set of hidden (internal) states $S = \{s_1, \ldots s_n\}$ and a finite set of observations symbols $O = \{o_1, \ldots, o_m\}$. Without loss of generality, we will denote the states and observations by their indices from now on. The HMM evolves on a discrete time scale $t = 0, 1, 2 \ldots$. At the initial time $t = 0$, the initial state of the HMM is drawn according to a starting probability distribution π:

$$\pi_i = Pr(s_0 = i).$$

On each time step t, and observation symbol k is emitted according to a probability distribution b_i dependent on the internal state i at that time step:

$$b_i(k) = Pr(o_t = k | s_t = i)$$

We denote by B the $n \times m$ matrix containing the emission probabilities for all states.

After the emission, the system transitions to another internal state j. The transitions between hidden states are governed by transition probability matrix A:

$$a_{ij} = Pr(s_{t+1} = j | s_t = i)$$

The Markov property means that both the state transitions and the probabilities of different observations depend only on the state at time t and not on any other events before time t.

This process continues until a terminal state is entered. The result is a sequence of observations $o_0, o_1, \ldots o_T$. The state transition probabilities A, the emission probabilities B and the starting probabilities π form the *model of the HMM*, denoted λ..

Models in which all states can be revisited are called *ergodic*. A special case of ergodic model is the *fully connected HMM*, in which any state is accessible from any other state (i.e., $a_{ij} > 0 \forall i, j = 1, \ldots n$).

Some applications require imposing constraints on the HMM. Constraints are implemented by forcing transitions between some states and/or disallowing transitions between others. Most of the time this leads to *non-ergodic* models. An important special case are *upper-triangular* or *left-to-right* models, where transitions are only possible to states with a higher index (i.e., $a_{ij} = 0$ if $i \geq j$). These models impose a temporal order on the HMM.

3.3.2 Algorithms for HMMs

Rabiner (1989) discussed three problems of interest for HMMs:

1. Given an observation sequence and a model, compute the probability that the given model produced the observation sequence

2. Given an observation sequence, and a model, compute the most likely state sequence that would generate the observed sequence

3. Given several observation sequences, determine the model that is most likely to generate the sequences.

Each of these problems has a standard solution algorithms, briefly described below.

Given an observation sequence $\langle o_0 o_1 \ldots o_T \rangle$, and a model λ of the HMM, we want to compute the probability that the observation sequence was generated by the model,

$Pr(\langle o_0 o_1 \ldots o_T \rangle | \lambda)$. Knowing this probability is important if we are currently considering several candidate models. By computing this probability, we can see which model is most likely to generate the sequence.

Consider a fixed state sequence $\langle s_0 s_1, \ldots s_T \rangle$, with s_T being a terminal state. The the probability of the observation sequence given the state sequence is:

$$P(\langle o_0 o_1 \ldots o_T \rangle | \langle s_0 s_1, \ldots s_T \rangle, \lambda) = b_{s_0}(o_0) b_{s_1}(o_1) \cdots b_{s_T}(o_T)$$

The probability of the state sequence occurring under the model is:

$$P(\langle s_0 s_1, \ldots s_T \rangle | \lambda) = \pi_{s_0} a_{s_0, s_1} \cdots a_{s_{T-1}, s_T}$$

The probability of the observation sequence being generated under any state trajectory can be obtained by summing over all possible trajectories:

$$\begin{aligned} Pr(\langle o_0 o_1 \ldots o_T \rangle | \lambda) &= \sum_{\langle s_0 s_1, \ldots s_T \rangle} P(\langle o_0 o_1 \ldots o_T \rangle | \langle s_0 s_1, \ldots s_T \rangle, \lambda) P(\langle s_0 s_1, \ldots s_T \rangle | \lambda \\ &= \sum_{\langle s_0 s_1, \ldots s_T \rangle} \pi_{s_0} b_{s_0}(o_0) a_{s_0, s_1} b_{s_1}(o_1) \cdots a_{s_{T-1}, s_T} b_{s_T}(o_T) \end{aligned}$$

This algorithm would need $2T \cdot N^{T+1}$ multiplications and N^T additions to perform the computation, since at every time step $t = 0 \ldots T$ there are N possible states and $2T$ computations are required for each. A more efficient approach is the *forward-backward method* [14], which uses dynamic programming to compute partial probabilities and each step, and then uses these intermediate variables to compute the cumulative probability of the next transition. We refer the reader to Rabiner's tutorial for details.

The most likely state sequence for a given observation sequence can be computed by the *Viterbi algorithm.* However, a more important algorithm for us is the Baum-Welch algorithm, which allows learning of the parameters of the HMM model. The

algorithm is a special case of expectation maximization [12]. The algorithm requires a set of observation sequences, which are used to estimate probabilities based on empirical counts. Then, the model is changed in such a way as to maximize the probability of the set of observations being produced. This is an iterative process, and it converges to locally optimal setting of the model parameters.

3.3.3 Applying HMMs to Aggregation

The main hurdle in applying HMMs to our information extraction task is formulating the model of the system. Previous work in tagging using HMMs (discussed in Chapter 2) used raw words as observations. However, this relies on a large data corpus, which we do not have at our disposal. The only previous work on HMMs for information extraction that we are aware of is Leek's thesis [10], in which he used HMMs to extract information regarding genes from scientific papers. Leek uses a hierarchical structure of HMMs, in which the HMM at the top level has only a few states, corresponding to the main parts of a sentence. At the low level, he uses finite automata and simple HMMs, to recognized simple word structures.

At some level, we can regard our system structure as similar. Our HMM builds on the results of the tagger (which is recognizing simple entities in the text). So our HMM plays the role of a higher-level decision maker. Unlike Leek's work, which uses as observations a combination of raw words and symbols generated by lower-level HMMs, we use only tags as observation symbols. This allows us to build a more compact HMM, and to use less data. However, the disadvantage is that the observation sequences are less precise, and tagging mistakes will affect the aggregation process too. Another difference between our work and Leek's is that we try to relate several basic objects in order to create composite objects, whereas Leek looks only for relationships between pairs of entities (genes and positions at which they are located on chromosomes).

We use two HMMs, one for each composite object to be constructed (Address and Contact). An extra HMM would be used to recognize organizations, but this is left for future work. The output symbols of our HMMs correspond to the tagged named entities (i.e., PersonName, StreetAddress, Phone, Email, Fax, City, Region, Country,). There are five output symbols corresponding to the Null tag (Null0, Null1,Null2, null3, Null4), which indicates text that is irrelevant to the domain of interest. We have several Null symbols in order to be able to measure the distance between the relevant tags. Every sequence of four non-relevant words is replaced with a corresponding null symbol, such that the first four words in a non-relevant sequence are replaced with the Null0 symbol, the second four are replaced with Null1, etc. There is also an output symbol for the start of a sequence, and two symbols corresponding to the negative and positive termination of an observation sequence. For instance, the Address HMM has a positive termination symbol, which is emitted in the positive termination state and indicates that an Address composite object can be created with the information contained in the observation sequence. This is similar for the Contact HMM.

Each state "corresponds" to one observation symbol. This correspondence is achieved by forcing the emission probabilities of exactly one observation symbol in each state to be close to 1.0, while setting all other emission probabilities for that state close to 0.0. For example, in state 7 the emission probability of the Phone observation symbol would be 0.99, and all other emission probabilities would cumulate to 0.01. This approach makes it easier for a human to interpret the model of the HMM, and the results it produces, e.g. when finding the most likely state sequence for an observation. We also trained and tested a model where the emission probabilities were randomized. As we will discuss in the next section, the results were very similar, but in this case, the prediction of the HMM was harder to interpret, and hence harder to use to actually build the composite objects.

Given the correspondence between states and observation symbols, the HMMs we are using have eighteen states (for Addresses), and nineteen states (for Contacts). State 10 is always the start state, and state 11 is the negative terminal state. States 12 and 18 are the positive terminal states for Addresses and contacts respectively. States 0-4 correspond to the Null tag symbols.

The initial models are fully connected, which means that revisiting some states can occur. This structure is needed for our application because tags can appear in different orders, and some of them may appear twice (e.g., a contact may have more then one phone number).

3.3.4 Training the HMMs

The HMMs are trained using labeled data produced by a person. He/she needs to identify the composite objects on tagged web pages, then use the manual-tagger to insert tags corresponding to the composite object on the web page. The document is then processed by the TagExtractor module, which creates a list of objects, as described in Section 3.1. This list is then passed to the ObservationSymbolGenerator module, which processes it and produces a list of observation symbols for the HMM.

The ObservationSequence Generator module creates symbol objects from the tag objects produced by the TagExtractor. A symbol object contains all the information from the corresponding tag object, as well as the code of the symbol. All the attribute tags that were inserted into the document during the tagging phase are removed from the symbol list. The InCaps and NoDef attributes are removed when a relevant tag encapsulates them. Otherwise, since these attributes isolate single words, they are used in order to create Null symbols. These symbols encode the number of words in a non-relevant sequence, as described above.

For example, the tagged text:

```
<InCaps>Contact</InCaps> <NoDef>us</NoDef>: <OrgName><InCaps>Yoyo</InCaps>
<InCaps>Inc</InCaps>.</OrgName><NoDef>head</NoDef> <NoDef>office</NoDef>
```

```
<NoDef>located</NoDef> <NoDef>at</NoDef> <NoDef>address</NoDef>
<StreetAddress>66 <InCaps>Avenue</InCaps> rd.</StreetAddress>
<City>"<InCaps>Toronto</InCaps></City>
<Region>"<InCaps>Ontario</InCaps></Region>
<Country>"<InCaps>Canada</InCaps></Country>
```

would be converted into the following observation sequence:

```
<Null0><OrgName><Null0><Null1><StreetAddress><City><Region><Country>
```

Like the heuristic aggregator, the HMM aggregator also makes three passes on the data with the objective of creating composite objects bottom-up. In the first pass, Address objects are created. In the second pass, the tags that compose the addresses are removed (because the addresses have already been found and the basic tags that compose them are no longer useful) and Contact objects are created. The third pass, which is not currently implemented, would be similar to the second one, except for Organization objects. For instance, the observation sequence in the example above would be converted after the first pass in:

```
<Null0><OrgName><Null0><Null1><Address>
```

The second pass would have no effect in this case, as there is no contact to aggregate in this observation sequence. However, a third pass would create the observation sequence:

```
<Null0><Organization>
```

as the OrgName and the Address are enough to form an Organization object. Null tags that appear inside the tag string corresponding to a composite object are discarded.

Our initial idea was to train the system using both positive and negative examples, generated based on the human-classified data. The OnbservationSequenceGenerator extracts several examples from each document, in the following way. The system identifies the last observation symbol from the document that could be part of the

composite object being sought (Contact or Address). This becomes the last symbol of the sequence, l. The first symbol of the sequence is then set to be the first symbol in the document (with index 0). the start symbol is inserted in the beginning, and a positive or negative termination symbol is inserted at the end, depending on the manually labeled data. Then the start symbol is moved by one position and a new sequence is generated. The process continues until the start and end symbol are at the same location. Then, the generator looks for the next symbol in the text that could potentially be part of the composite object, by decrementing l until such a symbol is found. The process of creating sequences resumes in the same way. This algorithm generates all the possible sequences from the document that have the potential of creating composite objects of the sought type.

Once all the observation sequences from all documents are generated, the model of the HMM is learned using the Baum-Welch algorithm, described in Section 3.3. In order to classify a new observation sequence from an unlabeled document, we generate two sequences which ware identical except the terminal symbol: one has a positive terminal symbol, the other has a negative terminal symbol. We present both sequences to the HMM, and compute the probability of each sequence using the forward-backward algorithm (described in Section 3.3). If the positive sequence has higher probability, the we assign a positive label to the sequence, otherwise we assign a negative label. If the sequence is assigned a positive label, then the corresponding composite object tags are inserted in the text. As before, the HMM will be asked to classify all sequences that could possibly contain a composite object.

The problem with this approach is that due the large ratio of negative to positive examples and the nature of the forward-backward algorithm, the probability of transition to the positive terminal state always approached 0.0, on all sequences. This results in the positive classification never being made. To overcome this problem, we decided to use only positive observation sequences to train the HMM. The

identification of composite objects is then achieved by presenting the model with all possible positive observation sequences (generated as described above). The sequence with the highest probability of being emitted is then classified as a composite object, provided that its probability is above a threshold. If several sequences have similar probability of being emitted, and all are above threshold, then all will be converted into composite objects.

3.4 Empirical Results

We compared the performance of the HMM aggregation method with the heuristic aggregator on a benchmark of 96 HTML documents, which generated 112 addresses and 93 contacts. Since the aggregator is not a learning system, we tested it on all the data. The results are summarized in table 3.1. We note that the heuristic aggregator was created based on 76 of the files, and manually creating regular expressions that would fit this data. Hence, we did expect it to have very good results.

Tested on	True Total	Guesses	Correct	Precision	Recall
Addresses	112	104	104	1.000	0.928
Contacts	93	81	61	0.753	0.656

Table 3.1. Performance of the heuristic aggregator

Since the HMMs are learning systems, in order to assess their performance, we need to use cross-validation. We used k-fold cross-validation, with $k = 5$ for the Address HMMs and $k = 3$ for the Contact HMMs. Note that, since we are dealing with document processing, we constructed the folds by dividing the documents into 5, respectively 3 groups, then using the documents in each group to generate data. This means that our folds do not contain the same number of examples, as is usually the case in experiments run on supervised learning tasks.

For both HMMs, we first trained models in which each state "corresponded" to an observation symbol. In this case, the emission probabilities were fixed and only the

transition probabilities were trained. We then performed a subsequent experiment, in which both the emission and transition probabilities were initially chosen at random, and they were all trained. We trained models with different numbers of states in this case. The goal of this experiment was two-fold. On one hand, we wanted to see if the initial structure we provided helped or hindered in terms of precision and recall. On the other hand, previous work by McCallum and his colleagues [11] suggested that it can be useful to build some mount of redundancy into the model of the HMM. We wanted to see if this was the case in our task as well.

Fold	True Total	Guesses	Correct	Precision	Recall
1	16	16	16	1.000	1.000
2	36	36	35	0.972	0.972
3	19	19	17	0.895	0.895
4	29	26	25	0.962	0.862
5	12	12	12	1.000	1.000
Mean $\pm$ St.Dev	22.4 ± 9.86	21.8 ± 9.44	21 ± 9.14	0.966 ± 0.043	0.946 ± 0.063

Table 3.2. Precision and recall for Address HMM with fixed emission probabilities, as a result of 5-fold cross-validation

Table 3.2 shows the precision and recall of the HMMs with predefined structure, in terms of the emission probabilities. Both precision and recall are very similar to the hand-crafted aggregator, with precision being slightly lower, and recall slightly higher. The differences are not significant. Indeed, the address construction task is quite easy, so it is not surprising that both techniques can achieve it well.

Table 3.3 shows the precision and recall of the HMMs for which both the emission and the transition probabilities are randomized. The data shows that both precision and recall improve with the number of states and then reach a plateau. Using more than 18 states improves recall a bit, but the difference is not significant. There does not seem to be any benefit in having more than 30 states. Again, we attribute this to the fact that we are dealing with a fairly simple problem. A paired t-test shows that the only statistically significant difference is between 6 states and more.

Number of States	Correct	Guesses	Precision	Recall
6	14.6	29.6	0.451 ± 0.208	0.588± 0.216
8	20.8	23.6	0.901 ± 0.118	0.937 ± 0.065
13	21	23.6	0.921 ± 0.087	0.952 ± 0.047
18	21	22	0.963 ± 0.034	0.949 ± 0.071
25	22	23.2	0.961 ± 0.037	0.984 ± 0.024
30	22	23	0.966 ± 0.031	0.984 ± 0.024
40	22	23	0.966 ± 0.031	0.984 ± 0.024
50	22	23.2	0.961 ± 0.037	0.984 ± 0.024

Table 3.3. Precision and recall (based on 5-fold cross-validation) for Address HMMs with different numbers of states. Both the emission and transition probabilities are estimated from data

We performed a similar set of experiments for aggregating contacts, using 3-fold cross-validation. Initially we trained the Contact HMM on all positive examples. This approach however resulted in very poor results, so we restricted the training to positive examples in which the first named entity in the relationship is a person name. This restriction resulted in a significant improvement in performance. The main reason for the poor results is that in a page with multiple contacts, an error in the aggregation of the first contact would propagate onwards and cause errors in the aggregation of the other possible contacts examined. As an example, consider the following sequence:

```
<Null0><PersonName><Phone><Null0><Null1><PersonName><Null0>
<Email><PersonName><Phone><Null0>
```

The correct aggregation of the contacts is:

```
<Null0><Contact><PersonName><Phone></Contact><Null0>
<Null1><Contact><PersonName><Null0><Email></Contact>
<Contact><PersonName><Phone></Contact><Null0>
```

However, because the HMM starts analyzing the observation sequence from the end, the first contact found would be:

<Contact><Email ><PersonName><Phone></Contact>.

Because the Email symbol was related to the wrong contact, the error would propagate and the next Contact object found would be:

<Contact><Phone><Null0><Null1><PersonName></Contact>,

which is incorrect as well. This propagation would likely continue throughout the document, resulting in a complete misaggregation. Since most Contacts start with a person name (or, put differently, since most of the time when a person name appears followed by contact information, this really constitutes a contact) training a contact to always start with a person name lowers the chance of errors as described above and thus improves precision.

Table 3.4 shows the precision and recall obtained when using 18 states, each associated with a particular observation symbol. As can be seen from the table, both precision and recall are superior to the ones obtained by the hand-crafted aggregator, although the difference is not statistically significant.

Fold	True Total	Correct	Guesses	Precision	Recall
1	16	7	8	0.875	0.437
2	42	33	34	0.971	0.786
3	35	33	50	0.660	0.943
Mean ± StD.		24.33 ± 15.01	30.67 ± 21.20	0.835 ± 0.159	0.722 ± 0.258

Table 3.4. Precision and recall for the Contact HMM, where the emission probabilities are fixed and the transition probabilities are learned

In order to analyze the effect of the structure we built into the HMM on it performance, we also trained one HMM with the same number of states (18), but with random initial emission probabilities. The results are presented in table 3.5. As can be seen from the table, there is a significant drop in precision, but an improvement in recall.

Fold	True Total	Correct	Guesses	Precision	Recall
1	16	12	44	0.272	0.750
2	42	40	50	0.800	0.952
3	35	31	49	0.632	0.886
Mean ± StD.		27.67 ± 14.29	47.67 ± 3.21	0.586 ± 0.269	0.862 ± 0.103

Table 3.5. Precision and recall for an HMM with 18 states but in which both the transition and emission probabilities were trained

Fold	True Total	Correct	Guesses	Precision	Recall
1	16	15	34	0.441	0.938
2	42	41	51	0.804	0.976
3	35	31	39	0.794	0.886
Mean ± StD.		29.00 ± 13.11	41.33 ± 8.73	0.680 ± 0.207	0.933 ± 0.045

Table 3.6. Precision and recall for an HMM with 30 states in which both the transition and emission probabilities were trained

Finally, we trained an HMM with 30 states, in which both the transition and emission probabilities were randomized. The goal of this experiment was to test again the hypothesis that building redundancy into the HMM can help. The results are shown in Table 3.6.

As can be seen, both precision and recall are better than those for the randomized HMM with 18 states. However, the result is not statistically significant. Paired t-tests show that the precision of the randomized HMM with 18 states is significantly worse than that of the structure 18-state HMM. The recall of the 30-state model si significantly better than the one for the structure 18-state HMM. The data supports the hypothesis that indeed having extra states is beneficial. However, more experimentation would be needed to truly confirm this hypothesis. It would be necessary to try several randomized HMMs for each number of states. We did not perform this sort of experiment in the thesis because in the current setup of our system, assessing precision and recall requires a person to analyze the documents and decide whether

the contacts extracted are valid. This is a very tedious and time-consuming operation, and should be automated in the future.

It is clear that all the learning systems have better recall then the hand-crafted aggregator. The difference is statistically significant at the 0.05 level for the 30-state randomized HMM. The superiority in recall is not too surprising. Our hope from the beginning was that the HMMs would be able to identify patterns that are missed by human programmers. This was indeed the case, as shown by a more careful analysis of the data. Fro example, the hand-crafted system was not able to associate a person name with a phone number if the name of an organization appeared in between, while the HMM aggregators handled this case correctly. On the other hand, the precision of the hand-crafted system is excellent for addresses and quite good for contacts. This is also not too surprising, since programmers tend to "debug" their systems to fit well a restricted range of cases.

Introducing structure in the HMM (in terms of the emission probabilities) helped the precision but hindered the recall. Again, intuitively this is due to the fact that the completely randomized systems are capable of covering a broader range of structures. However, introducing structure was very helpful in terms of interpreting what kinds of contact structure the HMM had identified. This is not possible with the randomized HMMs.

3.5 Discussion

In this chapter, we presented two approaches to information extraction: a hand-crafted aggregator, and a learning approach based on HMMs. Both approaches performed well in terms of precision and recall, and we cannot assert that one system clearly outperformed the others in all respects. The learning systems did however outperform the hand-crafted system in recall alone when extracting contacts, which

indicates the potential superiority of such systems for information extraction when the problem is complex.

In simple problems such as the extraction of addresses, it is easy for a programmer to identify most of the cases that are to be solved, and thus a hand-crafted system can attain both good precision and good recall. More complex problems require an extensive analysis of the data in order to identify all the possible cases, and thus achieve good recall. When there are many such cases, it is expected for a person to miss a few. A learning algorithm on the other hand, given an extensive training set, will have the opportunity to cover more possibility, and thus achieve better recall then the hand-crafted system. Good precision on the other hand is easier to achieve for a hand-crafted system as long as a few good rules can be identified and incorporated into the system.

One of the major advantages of using machine learning to solve this problem is the time savings in building the system. Machine learning only requires the construction of a system that will interface with the data and prepare the examples for the learning algorithm. Of course, data needs to be labeled manually, but this is a less time-consuming process than building all the rules of the system. In developing the hand-crafted aggregator, we ended up analyzing lots of documents anyway. Overall, the development of the HMM aggregator (including the label ling of the training data and the evaluation of the system), took half as much time as the development of the hand-crafted aggregator. This is partly due to the fact that in the hand-crafted system, several passes were necessary to refine and debug the regular expressions. No such iterative process was necessary for the HMM aggregators. The most time-consuming part in the case of the HMMs was designing the model, in terms of emission symbols and state connectivity. Of course, representation is always a major problem in such systems, and in this case we did not find it too difficult to tune.

CHAPTER 4

CONCLUSIONS AND FUTURE WORK

We have seen that the difficulty of information extraction is in finding the heuristics that will be used by the machine to extract information from web pages. There are two fundamental approaches to composing such heuristics. The first is for a human to evaluate the domain and specify the rules for the machine. The second approach involves using machine-learning algorithms to determine the rules that are to be used to perform IE. Advantages and disadvantages of such approaches were discussed throughout this work.

In this thesis we investigated two aspects of information extraction, classification of named entities and the establishment of relationships between named entities for the purpose of creating meaningful records. For each of these aspects we compared the performance of a hand-crafted system and a machine learning system. For named entity classification we compared a system that uses decision trees (i.e., C4.5) to a hand-crafted approach that relies completely on regular expressions written by a human. The decision tree tried to identify elements within the HTML script that would provide evidence that a word or phrase is an organization name. The hand-crafted approach was constructed by following an iterative process where regular expressions were written and then tested and tweaked iteratively until acceptable levels of performance were achieved. Both systems performed at the same level, but the regular expressions took significantly longer to develop.

For finding relationships between named entities with the purpose of identifying addresses and contacts we compared a hand-crafted aggregator and a learning system

using Hidden Markov Models. The empirical evidence suggests that more complex problems benefit more from machine learning than simple problems. We also found that, if a large variety of patterns has to be covered (as is the case for contact information) a human developer will likely miss some of those, while a learning algorithm that is given a sufficiently large training set will be able to identify them.

In summary we have seen that the machine learning systems equaled or improved upon the results of the hand-crafted system. The machine learning systems also took significantly less time to develop than the hand-crafted systems, and are likely more adaptable to work in new domains in which the same solution is applicable. However, new heuristics need to be found by training on new corpuses, so porting a learning system to a different application domain would still require a person to generate labeled training data.

Several aspects of this problem have not been thoroughly researched in this thesis and may prove interesting for future work. For instance, when learning to identify relationships we used only positive examples to train the HMMs, which resulted in lower precision (due to aggregating too much information). We believe that using negative examples would help prevent such errors. Further, we taught the system to extract only addresses and contacts. The problem of extracting organization objects is less challenging than the extraction of contacts, so we did not tackle it in this thesis. However, creating a training corpus for organizations and training an HMM on it would provide a sense of closure for the problem, although we do not anticipate that it provide any more insight into the problems discussed in the thesis.

The information extraction systems presented in the thesis were always trained and evaluated on human tagged data at every stage. To get a better feel of how the system we created really works it would be necessary to put the named entity extraction and the aggregation steps together and apply the entire system to the data. We know that doing so will reduce the performance of both the hand-crafted

and the learning systems. We expect however that the performance of the machine learning systems would degrade less than that of the hand-crafted system. HMMs are probabilistic models, and thus one would expect them to be less sensitive to noise in the data than deterministic models.

Of course, we also picked particular learning techniques for the experiments reported here. It would be interesting to experiment with other learning techniques as well. In particular, recent results in machine learning suggest that ensemble methods improve over the performance of any single algorithm, and they have been successfully applied to text processing. It would be interesting to see if using an ensemble of decision trees for the tagging step of our system would significantly improve performance.

BIBLIOGRAPHY

[1] Agichtein, Eugene, and Gravano, Luis. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries* (2000).

[2] Bikel, D. M., Miller, S., and Schwartz, M. Nymble: a high-performance learning name finder. In *Proceedings of ANLP* (1997), pp. 194–201.

[3] Carbonell, Jaime, Craven, Mark, Fienberg, Steve, Mitchell, Tom, and Yang, Yiming. *Report on the CONALD Workshop on learning from text and the web.* Carnegie Mellon University, Pittsburgh, PA, 1998.

[4] Cardie, Claire. Empirical methods in information extraction. *AI Magazine 18*, 4 (1987), 65–79.

[5] Cowie, J., and Lehnert, W. Information extraction. *Communications of the ACM 39*, 1 (1996), 80–91.

[6] Cunningham, Hamish. Information extraction - a user guide. Tech. Rep. 99-07, University of Sheffield, United Kingdom, 1999.

[7] Embley, D.W., Campbell, D.M., Liddle, S.W., and Smith, R.D. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *To appear* (1998).

[8] Embley, D.W., Y.S.Yang, and Ng, Y.K. Record-boundary discovery in web document. In *Proceedings of the ACM SIGMOD Conference* (1999), pp. 467–478.

[9] Kristie Seymore, Andrew K. McCallum, and Rosenfeld, Roni. Learning Hidden Markov Model structure for information extraction. In *AAAI Workshop for machine learning in information extractio n* (1999).

[10] Leek, Timothy R. *Information Extraction using Hidden Markov Models.* PhD thesis, Department of Computer Science, University of California, San Diego, San Diego, USA, 1997.

[11] McCallum, Andrew K., Nigam, Kamal, Rennie, Jason, and Seymore, Kristie. Autmating the construction of internet portals with machine learning. *Information Retrieval Journal 3* (2000), 127–163.

[12] Mitchell, Tom. *Machine Learning.* McGraw-Hill, New York, NY, 1997.

[13] Quinlan, John Ross. *C4.5: Programs for empirical learning.* Morgan Kaufmann, San Mateo, CA, 1992.

[14] Rabiner, Lawrence R. A tutorial on Hidden Markov Models and applications to speech recognition. *Proceedings of the IEEE 77*, 2 (1989), 257–286.

[15] Salton, Gerald. *Automatic Text Processing: The transformation, Analysis and Retireval of information by computer.* Addison Wesley, New York, NY, 1989.

[16] Sekine, Satoshi, Grishman, Ralph, and Shinnou, Hiroyuki. A decision tree method for finding and classifying names in japanese texts. In *Proceedings of the Sixth Workshop on Very Large Corpora* (1998).