THE UNIVERSITY OF CALGARY

ClusTex: Using Clustering Techniques for Information Extraction from HTML Pages containing Semi-Structured Data

by

Fatima Ashraf

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

October, 2005

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "ClusTex: Using Clustering Techniques for Information Extraction from HTML pages containing Semi-Structured Data " submitted by Fatima Ashraf in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE.

[illegible]

Supervisor, Dr. Reda Alhajj
Department of Computer Science

[illegible]

Dr. Ken Barker
Department of Computer Science

[illegible]

Dr. Svetlana Yanushkevich
Department of Electrical and Computer Engineering

Oct 18/05

Date

Acknowledgements

I really thought this day wouldn't come, but here it is.

First of all, I want to acknowledge the support my supervisor Dr. Reda Alhajj has provided me these past two years.

I thank Dr. Erkan Korkmaz for his support during the early stages of this work.

I also want to thank all my friends at the ADSA lab for camaraderie only few could offer.

A big thank you to the ladies in ICT-602, especially Jenny Cook, who I have bugged a lot lately.

On the family front:

First and foremost, I want to thank Irfan, my husband, for his unrelenting support in all ways possible. If it weren't for you, I would have given up a long time ago.

I must take this opportunity to express my heartfelt gratitude to my parents-in-law, who came a long way from Pakistan to take care of my son just so I could concentrate on my studies.

My fifteen month old son Ahmed must also be given due credit. The completion of my degree would not have been possible had he not started sleeping nights at the right time.

Last but not the least, I want to express gratitude to my wonderful family: my parents, my brother, my sister-in-law, and my two wonderful nephews. You all inspire me every day.

Table of Contents

List of Tables

List of Figures

Abbreviations

IE	Information Extraction
HTML	Hypertext Markup Language
XML	eXtensible Markup Language
IR	Information Retrieval
X	Data set
N	Number of data instances in X
x_i	A single data instance
d	Number of attributes or scalar components of x_i
x_{id}	The d^{th} scalar component of x_i
C	Cluster
l	The number of clusters found in a data set X
f_{ij}	The fractional degree of membership of an instance x_i to cluster C_j
POS	Part of speech
SGML	Standard Generalized Markup Language
ParseHTML	Algorithm implemented to parse and tokenize HTML document
RulExt	Algorithm implemented to discover the extraction rule
Refine	Algorithm implemented to refine raw clusters
htmlAttributes	Attributes of data instances (or text tokens) derived from the HTML tags surrounding each token
combiAttributes	Attributes of the token derived from orthographic information and utilizing domain knowledge
ch	A character from the input file
$input\_file$	The HTML document from which IE is intended
$char\_array$	Array containing all characters from the input file
num_char	Number of all characters in $char\_array$
Tok	Array containing all strings, tags and text, from the input file
num_tok	Number of strings in Tok
P_S	Size of a pattern
PT	A pattern that has size P_S when completed
$size\_of\_PT$	The size of PT during and after assembly
PT_Array	Array containing all valid patterns
$size\_of\_TEST$	The size of $TEST$ during and after assembly
loc_TEST	The location of the first member of $TEST$ in the token listing
occ_count	The number of occurrences of a pattern PT
PAT	The desired extraction rule
k	The size of PAT

Abbreviations (continue)

P	Precision - The percentage of correct values in the extraction information
R	Recall - The percentage prediction of all possible correct values in the input file
F	The F-value - Geometric average of P and R with parameter β as the weight of R over P
$F1$	The F-value where β is equal to one.

Chapter 1

Introduction

1.1 Motivation and Hypothesis

"We're drowning in words", writes Kushmerick [25], referring to the terabytes of information that is available on the Web today. According to a 2005 study [18], there are 11.5 billion pages on the "indexable web" [1], and it is not just text that these pages contain. There are also images, and listings of various kinds. It is not hard to imagine, then, how the cliche *information overload* was born.

It must be realized that the information on the Web is dynamic: it changes and increases continuously, so much so that the Web has been termed "the largest knowledge base" [8] in history to be developed and made accessible to the public. One must also remember the "hidden web" which is made up of documents generated by user queries to web databases. This leads us to the following questions. How much of this information is *actually* exploitable by us? Is it even *remotely* humanly possible to manually peruse through these billions of pages to extract the information that one is interested in?

Currently, there are two ways of accessing this information: through manual browsing (i.e. following links to go from one web page to the other), and keyword searching. Both these techniques are problematic. Manual browsing involves a lot of human time and involvement; given the sheer amount of information on the Web,

[1]Indexable web is that part of the World Wide Web which is considered for indexing by major search engines such as Google, MSN, and Yahoo!

it is akin to finding the proverbial needle in the haystack. Keyword searching seems more promising at first since it takes care of part of the task, i.e., relevant documents are retrieved by the search engine. However, on the down side, one still has to go through these documents and glean relevant information manually. This involves the field of information extraction.

Information extraction (IE) has been defined as the process of automatically "identifying a set of pre-defined relevant items" from documents [15]. These relevant items are extracted and may be used to populate a database so that a user may query this database at a later time to access the information of interest. In other words, given a document or a set of documents and an IE system, one aims to be able to extract all information of interest from it as automatically as possible.

IE promises to be a sizeable augmentation to the search engines available today. When a user does a keyword search on an engine, a large number of documents may result which might be very time consuming to sift through. An IE system can extract precisely the information a user wants from this set of documents, and provide the user with exactly the information that is required without the level of involvement that this task requires currently.

In the last few years, there has been a lot of pioneering effort in the field of IE. Researchers have worked on extracting information from various kinds of text, employing different strategies for the actual extraction. Recently, machine learning techniques have been used to learn extraction rules in the IE process. Specifically, a few feature-based systems have been developed that view IE as a text classification problem, such as SRV [16], BIEN [35], and ELIE [15]. These systems deal with IE from documents containing semi-structured text and use a complex feature set.

These systems extract the information semi-automatically since they are based on supervised learning. There is a need to automate this process further to minimize the level of human involvement.

In this thesis, the issue of automating the IE process by introducing clustering techniques is addressed, and a system called ClusTex is proposed. This approach concentrates on pages containing listings that are usually hand-coded in the HyperText Markup Language (HTML). This may be structured or semi-structured data. If we consider a specific domain such as Computer Science course listings, the examples of such data would be information related to courses such as course codes, course titles, instructors' names and so on. Generally, however, one may think of this data as any information related to the instances reported in the listings.

One may question the validity of this research direction since XML is fast becoming the standard for machine-readable web documents. However, there is still the presence of legacy data [8] to consider: in addition to the documents currently being written in HTML, there are also many HTML web documents present from the pre-XML era from which we would like to extract information and populate databases automatically.

1.2 Methodology

In this thesis, IE is viewed as a clustering problem, i.e., given some information, we want to automatically infer the natural clusters in it rather than setting up classes first and then deciding if a particular item falls in a specific class. This is achieved by using clustering techniques to first separate raw data into applicable clusters using information provided by a feature set defined for every text string or token in the page. These clusters are further refined by using patterns to cull out irrelevant information and re-classify tokens that were incorrectly clustered in the earlier step. The system ClusTex implements these ideas.

Clustering is accomplished using AutoClass [5, 19] which implements probabilistic clustering. This system has been developed at NASA.

1.3 Contribution

The contribution of this thesis is to simplify the automation of IE from HTML web pages containing semi-structured data by using clustering techniques. To the best of my knowledge, clustering has not been previously used to extract data from documents. The feature set used in this system is much simpler than the one used by the existing systems described in the literature. Furthermore, ClusTex requires very little pre-processing as compared to the existing approaches mentioned above, but gives comparable results. Three algorithms are proposed for file parsing, extraction rule estimation, and cluster refinement, respectively.

1.4 Organization of Thesis

The balance of this thesis is organized as follows. Following this *Introduction*, Chapter 2 (Background Information) defines some basic terms and acts as a primer on IE and clustering. In the first half of the chapter, the concept of IE is explained. Different types of text from which IE takes place are explored, and a simple introduction to wrappers and wrapper generation is presented. The second half of Chapter 2 goes over the concept of clustering, how clustering and classification differ, and various commonly used clustering techniques.

Chapter 3 (Related Work) briefly describes relevant research in the IE area. This work is classified into two groups. The first group includes research work that concentrate on IE from HTML documents. The second group includes work that considers IE as a text classification problem.

Chapter 4 proposes ClusTex, a novel approach for IE that uses clustering to extract information from semi-structured data in HTML pages. The assumptions are stated after the problem definition. The proposed process consists of four stages: data preparation, clustering, estimation of the extraction rule, and refinement. Each stage is described in the chapter. A simplistic example is used to explain the approach.

Chapter 5 contains the results of test runs on web documents. In this chapter, the various characteristics of the testing environment and definitions of the evaluation metrics, specifically, precision, recall, and the F1 value are stated. Details of experiments on seven web pages are presented, and results are compared with those reported in relevant works.

Chapter 6 concludes this thesis. The advantages and disadvantages of this approach and its differences from other existing methods are discussed. Future research directions are also suggested.

Chapter 2

Background Information

This chapter presents a primer on the major topics which form the basis of my proposed approach. In the first section, information extraction, its definition, and some other relevant concepts such as various types of text that can be extracted, wrappers and wrapper generation are discussed. In the second section, the basic ideas behind clustering and various clustering techniques are presented.

2.1 Information Extraction

2.1.1 What is Information Extraction?

Information extraction (IE) has been defined by Glickman *et al.* [17] as "a process that takes unseen texts as input and produces fixed-format data as output". In other words, the goal of IE is to transform text into a structured format so as to reduce the information in a document to a tabular structure [10].

IE helps us format documents containing bits of structured text along with irrelevant material into a "database-like representation" [32]. One can use this information for analysis at a later stage through techniques such as data mining for discovery of patterns in the data.

IE is useful for all kinds of documents from which we would want to extract data including web documents. An IE system enables users to gather all the relevant information that maybe strewn about in various sources and integrate it into one structured form.

2.1.2 Types of text for extraction

Information extraction can be performed on free, structured, or semi-structured text. These terms are defined as follows.

Free Text

This is usually natural language text [36], for example, news articles and research paper abstracts. One would want to extract the important information from these sources. This is most commonly done using natural language processing (NLP) techniques, and the extraction rules are based on patterns involving syntactic relations between words or semantic classes of words.

Structured Text

Structured text is defined [10] as textual information in a database or a document following a predefined and strict format. Usually the information can be easily extracted using the format description if the format is known. Otherwise the format must be learned.

Semi-structured Text

It is the intermediate point between free text and structured tuples of data [36]. It maybe ungrammatical and does not follow any rigid format. It is often telegraphic in style, i.e., uses abbreviations of words. NLP techniques cannot be used very successfully to extract data from these documents because those are better suited for grammatical text. On the other hand, some researchers have tried to use machine learning techniques for IE from semi-structured data. In the next chapter, I present the relevant research in this area.

2.1.3 Information Extraction vs. Information Retrieval

One commonly reads the terms information extraction and information retrieval (IR) mentioned in the same breath. This can make one wonder if the two concepts are the same. In principle, IE is different from IR both in aims and objectives and the methods used to achieve those aims [11, 10].

The objective of IR is to select a subset from a larger collection of documents based on a query. In contrast, the goal of IE is to extract relevant information from documents.

Despite the two technologies being inverses of each other, it should be realized that they are complementary, and, can be combined to form powerful information integration systems [4, 14]. An example of the complementary nature of IE and IR is a work done by Hu *et al.* [21] in which the titles extracted from web documents are used in picking relevant documents during the retrieval stage. The needed information is then extracted from the retrieved documents.

2.1.4 Web Documents

Since the advent of the Web and Internet, researchers have shown an interest in the vast information source that the Web is. The terabytes of information available on the Web maybe in the form of free, structured, or semi-structured documents. Researchers have different opinions on what information to categorize as structured and what to categorize as semi-structured. However, Eikvil [10] gives a better categorization of types of web pages, which is as follows.

- A web page is structured if each attribute in a tuple can be correctly extracted based on some uniform syntactic clues, such as delimiters or the orders of attributes.
- A web page is semi-structured if it contains tuples with missing attributes, attributes with multiple values, variant attribute presentations, and exceptions.
- A web page is free text if linguistic knowledge is required to extract the attributes correctly.

Usually machine generated web pages are structured and human generated web pages have less of a structure. However, there are always exceptions.

2.1.5 Wrappers and Wrapper Generation

According to a recent study [18], there are billions of documents available on the Web. These documents can be found either by manual browsing or keyword searching using a search engine. However, as was mentioned earlier, these techniques are time-consuming, inconvenient, and too dependent on human users. Furthermore, there is also an increase of information in the so-called "hidden web". This is the name given to web pages generated dynamically from databases based on user requests. These pages cannot be reached through search engines, and require certain tools to extract information from them. Wrappers fulfill this need.

Wrappers [10] are programs or procedures designed for extracting content of a particular source and delivering the content of interest in a self-descriptive representation. In other words, a wrapper is a software component that converts data and queries from one model to another.

An ideal wrapper for an information source on the Internet accepts queries from users about the content of the source, pulls out relevant pages from this source, gets the requested information, and returns the result. Formally, it is a "function from a page to the set of tuples it contains" [24].

Laender *et al.* [28] states the problem of generating a wrapper for Web data extraction in the following words.

> Given a web page *S* containing a set of implicit objects, determine a mapping W that populate a data repository *R* with the objects in *S*. The mapping *W* must also be capable of recognizing and extracting data from any other page *S'* similar to *S*.

Thus, a wrapper is a program that executes the mapping W. Wrappers maybe hand generated or made through the semi-automatic or automatic approach.

Manually generated wrappers

For obvious reasons, manually generated wrappers are time-consuming because they require manually encoded dictionaries of vocabulary. These also require domain-specific extraction rules and patterns which are not easily portable. Writing one requires a deep understanding of the domain on the part of the developer. Similarly, new sources appear frequently and the format of existing sources may change. Thus, to keep up with these challenges, it is imperative that technology aids the construction of wrappers. According to Laender *et al.* [28], tools like Minerva, TSIMMIS, and Web-OQL are used for generating wrappers manually.

Semi-automatically generated wrappers

For semi-automatic wrapper generation, the developer uses support tools to help design the wrapper. For instance, the user could show the system an example of the information to be extracted using a graphical interface. This approach has many advantages. It is less tedious, requires lesser domain knowledge for the developer than for manually generated wrappers, and is less error prone also. However, the system needs to be shown the information to be extracted for every new site because the system cannot induce the structure of the site itself. Tools based on NLP, wrapper induction, and modeling are usually semi-automatic [28]. Examples are SRV [16] and BIEN [35].

Automatically generated wrappers

Automatic wrapper generation uses machine learning algorithms. However, it is not completely automatic either. These systems need a little intervention from human experts during the training phase where the system is fed training examples. The eventual accuracy of the wrapper generation system depends on the number and quality of these examples. Examples of systems generating wrappers automatically are RoadRunner [8] and W4F [28].

2.1.6 Desirable Features in a Wrapper

According to Laender *et al.* [28], an IE system should possess a few desirable features. These features are as follows.

1. Degree of Automation : An important feature of a data extraction system is its degree of automation: the amount of work a user has to do while generating

a wrapper for data extraction.

2. Support of objects with a complex structure : Most of the data available on the Internet has a complex structure. This structure is usually vague and presents degrees of variation that are typical of semi-structured data. In other instances, web data maybe organized in hierarchies with multiple nesting levels. Thus, it is imperative that a good data extraction system be able to deal with these complex objects.

3. Page contents : Page content maybe of two kinds: semi-structured data and semi-structured text. Pages of the first kind feature data items implicitly formatted to be recognized individually. According to Hong and Clark [20], this is fielded data unconstrained by a global schema. Pages of the second kind have free text from which data items can be inferred. It needs to be decided whether a data extraction system will deal with either or both these types.

4. Availability of a Graphical User Interface : This is one of the most important features of a data extraction system. Traditionally this has been accomplished by writing code using some general-purpose language.

5. XML output : Since XML is fast becoming a standard for data representation and exchange on the Internet, thus an important feature for a data extraction system would be the ability to provide its output in XML. XWrap [29] and Lixto [7] are two such systems that extract information from HTML pages and convert it into XML.

6. Support for non-HTML sources : Not all data on the Internet is available in

HTML files. For example, there is a lot of data in text files. It is very important for a data extraction system to be able to handle such data sources.

7. Resilience and adaptiveness : To deal with the constantly changing structural and presentation features of web pages, a wrapper needs to be resilient. A wrapper built for pages of a specific web source on a given application domain should be able to work properly with pages from another source in the same application domain, i.e., a wrapper should be adaptive.

2.2 Clustering

2.2.1 What is Clustering?

Clustering is a partitioning of a data set into subsets or *clusters*. Each cluster contains data that are similar to the data in the same cluster and dissimilar to those in the other clusters. In other words, clustering divides data into groups of similar items [22].

According to Berkhin [1], "from a machine learning perspective clusters correspond to hidden patterns, the search for clusters is unsupervised learning, and the resulting system represents a data concept".

Clustering for data mining works best on large data sets with numerous attributes of various types. An example of clustering [22] is shown in Figure 2.1. Figure 2.1(a) shows the input data and the cluster output is shown in Figure 2.1(b).

2.2.2 Formal Definition and Notations

Jain *et al.* [22] defines clustering and related concepts as follows.

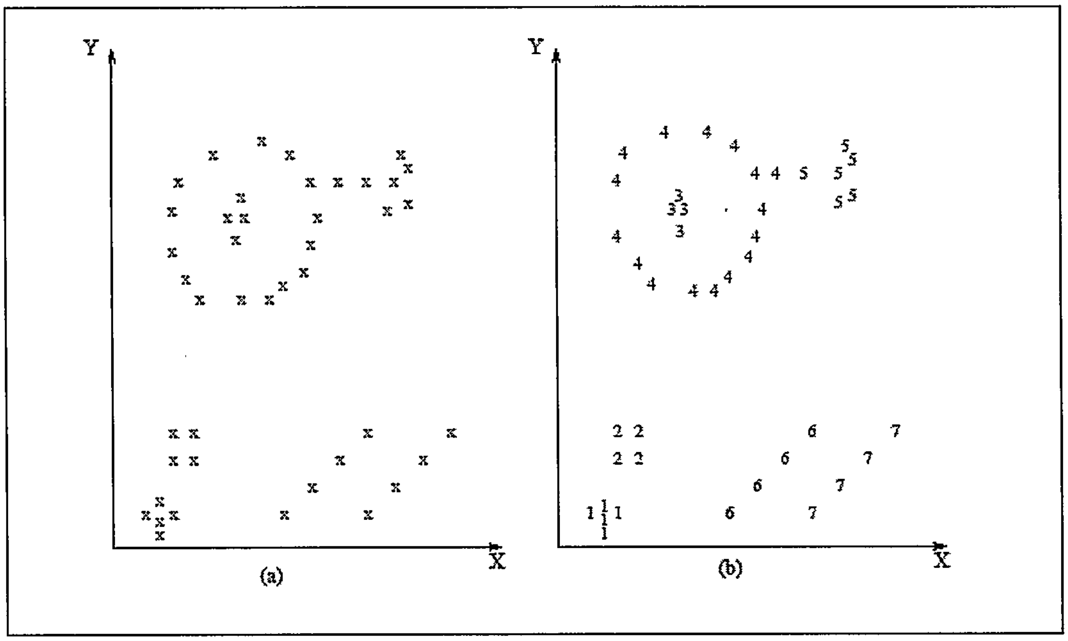


Figure 2.1: Data Clustering [22]

A data set X containing N data points (hitherto referred to as data instances) is denoted $X = \{x_1, \dots, x_N\}$.

A data instance is usually represented by a multidimensional vector of dimension d such that $x_i = (x_{i1}, \dots, x_{id})$. The d scalar components of x_i are called its *attributes.*

The data set X is viewed as an $N \times d$ matrix by the clustering algorithm.

Clustering aims to assign the N data instances to a finite system of l clusters such that $X = C_1 \dots C_l$. *Hard* clustering techniques assign each data instance to exactly one cluster. *Fuzzy* clustering techniques, on the other hand, give to each data instance x_i a fractional degree of membership, f_{ij}, to each cluster C_j.

2.2.3 Clustering Versus Classification

Classification and clustering are very similar terms and, in certain cases, are used interchangeably. However, there are key differences between the two techniques which should be pointed out. The two vary greatly in terms of the methodology they use.

Classification is the partition of the data set into subsets which have been already defined. In case of supervised classification, the data analyst has already labeled or classified some examples. These examples are then used to deduce the class properties. When an unlabeled example is encountered, these class properties are used to classify this new example into one of the already existing classes.

On the other hand, clustering partitions the data instances, which are a collection of unlabeled or unclassified examples, into clusters. This is done by a clustering algorithm which uses information provided solely by the data instances.

Thus, classification is usually taken to mean supervised classification, while clustering is unsupervised classification [22]. According to Liu [30], the biggest disadvantage of supervised classification methods is that they are limited in their capacity to test hypotheses. They can help or reject the hypothesis but are unable to uncover any unexpected information and do not lead to a new hypothesis. In contrast, unsupervised methods are able to mine through the data and reveal unexpected results.

Clustering is considered to be a special case of classification. Jain *et al.* [23] suggests a "tree of classification problems" as shown in Figure 2.2. Each node of the tree shows different types of classification problems.

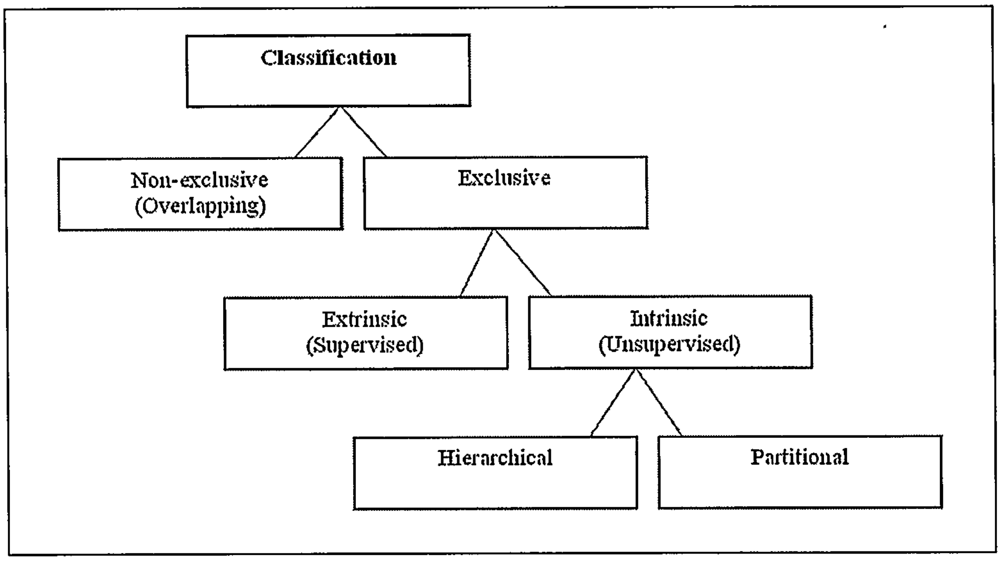


Figure 2.2: Hierarchy of Classification Techniques [23]

The terms in Figure 2.2 are defined by Jain et al. [23] as follows. *Exclusive* classification partitions data so that each instance belongs to one class only. *Non-exclusive* classification lets instances belong to several classes. *Extrinsic* classification uses category labels in addition to the data itself to classify items, and thus, is supervised. *Intrinsic* classification uses only unlabeled data and is also called unsupervised learning (i.e., clustering). The terms *hierarchical* and *partitional* are described in detail as follows.

2.2.4 Clustering Techniques

Berkhin [1] categorizes the various clustering techniques as follows.

Hierarchical Clustering

Hierarchical clustering transforms the data instances into a "sequence of nested partitions" [23], which can be visualized as a tree of clusters known as a *dendogram.* Once a data instance is assigned to a cluster, it cannot be moved.

Hierarchical clustering algorithms are usually categorized as follows [22].

1. *Agglomerative algorithms* use bottom-up clustering, i.e., they begin with each data instance in singleton clusters, and merge clusters until a stopping condition is fulfilled.

2. *Divisive algorithms* use top-down clustering, starting with all data instances in a single cluster, and successively splitting the cluster into smaller ones until a stopping condition is fulfilled.

Hierarchical clustering methods aid in constructing taxonomies, and for this reason, are generally used in biological, social and behavioral research [23].

Partitional Methods

As opposed to hierarchical methods, partitional methods help in discovering natural groups in the data by generating a single partition [23]. Partitional methods "learn" clusters directly as opposed to hierarchical methods which "grow" clusters slowly.

Berkhin [1] suggests the following types of partitional methods.

1. *Partitioning relocation methods* find clusters by assigning the instances to clusters first and then iteratively relocating them between clusters.
2. *Density-based partitioning methods* "identify clusters as areas highly populated with data".

Partitioning Relocation Methods

Partitioning relocation methods have an edge over hierarchical clustering because there is no restriction on the reassignment of an already-assigned data instance to another cluster if it improves the clustering [37]. Checking all possible permutations of data assignment to clusters is infeasible, so greedy heuristics are used for iteratively reassigning data instances between clusters. Berkhin [1] suggests the following types of partitioning relocation methods.

1. *Probabilistic methods*: These methods identify the clusters with certain models with parameters that are unknown and that need to be found. Hence, the data is assumed to be an independently drawn sample from "a mixture model of several distributions". An example of this method is the algorithm AutoClass [5, 19] that I also use in this thesis. It is based on a mixture model as well and covers various distributions such as Bernoulli, Poisson, Gaussian, and Log-normal distributions. More detail about AutoClass follows in Chapter 4.

2. *K-Means methods*: These are a popular tool used in the scientific and the industrial communities. The name arises from the fact that, in this approach, each cluster C_j is represented by the mean or weighted average c_j of its members instances. The sum of differences between data instances of a cluster and their mean is used as an objective function. Thus, this approach does not work well with categorical data. However, it is well-suited to numerical data.

3. *K-Medoids methods*: In these methods, a cluster is represented by one of its member instances, called the medoid. After the selection of medoids, clusters are presented as subsets of data instances close to a particular medoid. The objective function is defined as a dissimilarity measure between a cluster member and its respective medoid. Unlike the K-means methods, the K-medoid methods deal well with both categorical and numerical data. Furthermore, outliers do not affect the medoids so these have an "embedded resistance" against outliers [1].

Density-Based Partitioning Methods

According to Jain *et al.* [23], clusters can be imagined as regions of data instance space in which instances are dense, i.e., occur frequently. Areas in which instances occur sparsely separate these regions of high density. Thus, the problem of finding clusters is reduced to finding regions of high density in the instance space. Berkhin [1], on the other hand, defines a cluster as a "connected dense component". Clusters can grow wherever density leads. A cluster is not limited to a specific shape, and can acquire any arbitrary shape.

Other Clustering Techniques

In recent years, many other techniques have been developed for clustering data of various kinds. These include grid-based techniques, co-occurrence of categorical data, constraint-based clustering (such as evolutionary methods), clustering of high-dimensional data, and scalability and VLDB (Very Large DataBase) extensions. It is out of the scope of this treatise to go into the details of these methods. One can refer to an excellent overview of these techniques in Berkhin [1].

Chapter 3

Related Work

The field of IE has seen active research in the last decade. Many researchers have worked on finding efficient IE strategies for semi-structured information from web pages. In this chapter relevant work is briefly reviewed.

In Section 3.1, I discuss approaches that work on HTML pages exclusively. In Section 3.2, I review systems that view IE as a text classification problem and work on general text files that may or may not be coded in HTML.

3.1 Systems dealing with HTML pages

Since a lot of web data is found in HTML pages, there has been much effort in IE from such pages. This section reviews some work in the area of IE from HTML pages.

Buttler *et al.* [2] model a web document as a "tag tree" in which the internal nodes are HTML or XML tags and the text forms the leaf nodes. The extraction process requires fetching a web document, cleaning it up using a syntactic normalization algorithm, and then locating "objects of interest" in this web page. This is done by first locating the minimal object-rich subtree (i.e., the smallest subtree that contains all objects of interest). To separate objects from each other and other information in the page, the correct object separator tag is discovered using heuristics. Using this separator tag, the objects of interest are discovered. Finally, the set of objects is refined to eliminate irrelevant objects.

Crescenzi *et al.* [9] present a system to automatically extract data from large data-intensive web sites [1]. Their "data grabber" explores a large web site and infers a model for it, describing it as a directed graph with nodes describing classes of structurally similar pages and arcs representing links between these pages. After pinpointing classes of interest, a library of wrappers can be generated, one per class with the help of an external wrapper generator and appropriate data can be extracted.

It is simple for a human to find a table of interest in an HTML document, parse it and then determine its meaning, but having an algorithm accomplish these tasks is much harder. Embley *et al.* [13] propose an approach to automatically extract information from HTML tables. Tables of interest are located in a web page and information extracted from them in a step-wise manner. As the first step, an extraction ontology is formulated. An extraction ontology is a "conceptual-model instance" that serves as a wrapper for a narrow domain of interest [12] . A table is located based on recognizing expected attribute names and values from the ontology. Then attribute-value pairs are formed and adjusted so that they are more meaningful. In the fourth step, the extraction patterns are analyzed to refine the extracted information further. Finally, given the input from the earlier four steps, a mapping can be inferred from the source to the target.

Another relevant approach is the RoadRunner [8]. It extracts information from data-intensive websites in which pages are automatically generated by scripts that

[1]These are the sites in which data is scattered in a large number of structurally similar HTML documents.

fetch data from a back-end database management system. These pages are presented in a structured format representing tuples from the database. RoadRunner compares two of these similarly organized pages to infer their common structure and a wrapper. This wrapper is used to extract information from all similar pages. RoadRunner does not assume any a-priori knowledge about the structure or the contents of an HTML page. The advantage of this approach is that it is automatic, and can deal with nested objects. However, the disadvantage is that it requires two pages to be able to infer the structure. If this same information was on one page, RoadRunner would not be able to extract the data from it. Furthermore, it cannot extract from web sites in which attributes appear in various orders.

Labsky *et al.* [27, 26] attempts IE from web product catalogues using Hidden Markov Models (HMMs) as a step towards building a domain-specific "semantic search engine" which can answer queries about different product attributes, such as names, prices, etc. In this approach, each token from a document is assigned a semantic tag by an HMM tagger. The advantage of this approach is that it extracts images as well as text. However, it requires manually annotating training examples.

3.2 Systems solving IE as a classification problem

Many researchers have solved IE as a text classification problem. This section reviews some important work in this domain.

The Sequence Rules with Validation (SRV) [16] is a top-down relational algorithm for information extraction from semi-structured pages. Generally IE involves multiple sub-tasks including, but not limited to, syntactic and semantic preprocessing and slot-filling. SRV considers only the slot-filling aspect and views IE as a classification problem where all possible tokens from the text are considered as possible slot fillers. Input to SRV is a set of documents with field instances already labeled for extraction and a set of features defined over tokens present in the pages. The output is a set of extraction rules. SRV is comprised of three classifiers of tokens, the first being a look-up table containing all correct slot-fillers found in the training set. The second classifier calculates the probability of being a correct slot-filler for each token. The third classifier makes use of constraints obtained by rule induction over predicates like token length, capitalization, and semantic and relational features.

Peshkin *et al.* [35] also view IE as a text classification problem and use Dynamic Bayesian Networks to automatically extract information from semi-structured documents. The authors show how to combine various aspects of a language in one probabilistic model to build a robust IE system. The first step in the extraction is tokenization, the token being the smallest part of text which is treated as an entity in later steps. The features used for each token are its lemma (i.e. the root word for an inflected form), Part-Of-Speech information, capitalization and length, and semantic and orthographic features.

Eliassi-Rad *et al.* [11] describe intelligent agents that retrieve documents from the Web and extract relevant information from these documents. Their system called Wisconsin Adaptive Web Agent (WAWA) is able to build an intelligent agent for information retrieval and extraction after interacting with the user and the Internet. The system has two sub-systems, one for information retrieval and the other for information extraction.

The system WHISK [36] is able to handle structured, semi-structured, and free text documents. It uses supervised learning to learn extraction rules from hand-tagged training examples. The human expert is presented with examples that fall near decision boundaries rather than any arbitrary examples. Thus the learning of rules and the tagging of training examples is alternated to minimize the amount of human involvement while at the same time maximizing its benefits. WHISK repeats this process to further refine the rules as new examples are encountered.

Kushmeric *et al.* [15] propose the ELIE algorithm. This approach also treats IE as a classification problem. Each token is described by a set of features including the token itself, the part-of-speech of the token, the values associated with the token in a gazetteer, and orthographic information. Furthermore, relational information is encoded as additional features, and thus, simulates relational learning. ELIE classifies tokens in two phases, L1 and L2. In L1, ELIE learns start and end of instances to be extracted from the set of training examples labeled either as a positive or negative example of a start or end tag. Two classifiers are used for this purpose, one for start tags and the other for end tags. During L1, some instances might be partially extracted, i.e., either the start or the end of the fragment might be extracted but not both. During the L2 phase, the system is trained to find the end given the beginning, or the beginning given the end.

The system Pinocchio [6] uses hybrid relational learning techniques for IE and has shown promising results. The input to the system is a collection of texts that have been preprocessed with a POS tagger. Pinocchio produces as output the text augmented with SGML tags which point out the locations from which information has been extracted, and a summary of the text content in the shape of a set of templates. The system uses a sequential covering algorithm. A unique property of

Pinocchio is that it does not attempt to learn the extraction rule for an entire slot. On the contrary, it recognizes the left delimiters separately from the right ones. The process of IE is three-fold. In the first phase, the best rule pool is induced using a bottom-up approach. In the second phase, more extraction rules are refined and added to the best rule pool to increase the recall. The final phase sees the correction rules being learned. This system can also be used to learn multiple extraction slots.

All these systems show comparable performance. However, their drawback is the annotation of training examples and user involvement even after the training phase. Thus, in this thesis, ClusTex is presented which does not require training examples. The next chapter describes the ClusTex model and its implementation in detail.

Chapter 4

ClusTex: The Idea and its Implementation

This chapter proposes a novel approach, ClusTex, that involves clustering to perform IE from HTML documents. IE has been traditionally viewed as a text classification problem. However, this thesis views it as a clustering problem. This chapter begins with the problem statement and assumptions. After that, the process that this approach uses is explained, i.e., starting from data preparation to clustering and the estimation of an extraction rule, and finally refinement of these clusters.

4.1 Problem Statement

As stated earlier, there are billions of documents on the Web from which we want to automatically extract information. There are various kinds of documents available: HTML files, XML files, various types of text documents, etc. In this thesis I concentrate on those HTML documents on the Web that contain a listing of some sort. This is semi-structured data because this information does not always follow the same format and may contain missing or multiple tuple values. Thus, formally, this model is used to solve the following problem:

Given a web page in HTML format containing a listing of semi-structured data, extract the information contained in the listing and consolidate the information pertaining to each individual listing.

4.2 Assumptions

To solve this problem, we first need to specify what sort of web pages this approach works on. In this section, I specify the requirements that a web page should fulfill for this approach to work. Generally, these requirement are on the type of the web page and the format of values that need to be extracted. Specifically, the web page containing the listings must fulfill the following criteria.

1. The web page must be written in HTML.
2. The document must be "well-formed" [2], i.e., it should show the following characteristics:
   - Text which is not a tag should not have either opening or closing brackets ("<" or ">"). Conversely, there should be no tags without opening and closing brackets.
   - All tags that usually occur in pairs, such as `<title>` and `<\title>`, must have matching starting and ending tags.
   - All nesting tags should be nested properly such as, `<td><b><\b><\td>`.

   However, if the document has imperfect structure, it may be easy to fix using a utility such as HTML Tidy.
3. All strings of text representing the tuple values must be separated by HTML tags.
4. If the information is listed in a table, the majority of values listed in each column must be in a similar format. For example, in the Computer Science course

listings domain, course codes usually have the same format for all courses, i.e., capital letters followed by numbers.

5. The values in different columns should have different formats so that they may be recognized that distinct from each other by the clustering algoritm.

6. The values should follow the same pattern in the page.

4.3 ClusTex - The Basic Idea

ClusTex is devised to perform IE from HTML documents containing semi-structured data using a clustering technique. It is a four-fold process which starts with data preparation. The data is then clustered, an extraction rule is estimated and clusters are refined using this rule. Finally, the data instances discovered in the data are reported.

Figure 4.1 shows an overview of this system. For three of the four stages, new algorithms are developed. These are as follows.

1. Algorithm: *ParseHTML* - To parse the HTML document and tokenize data

2. Algorithm: *RulExt* - To discover the extraction rule

3. Algorithm: *Refine* - To refine clusters and format final output

These algorithms are explained further in the next few sections. The clustering uses AutoClass [5, 19]. In the interest of completeness, details about AutoClass follow in the section on clustering.

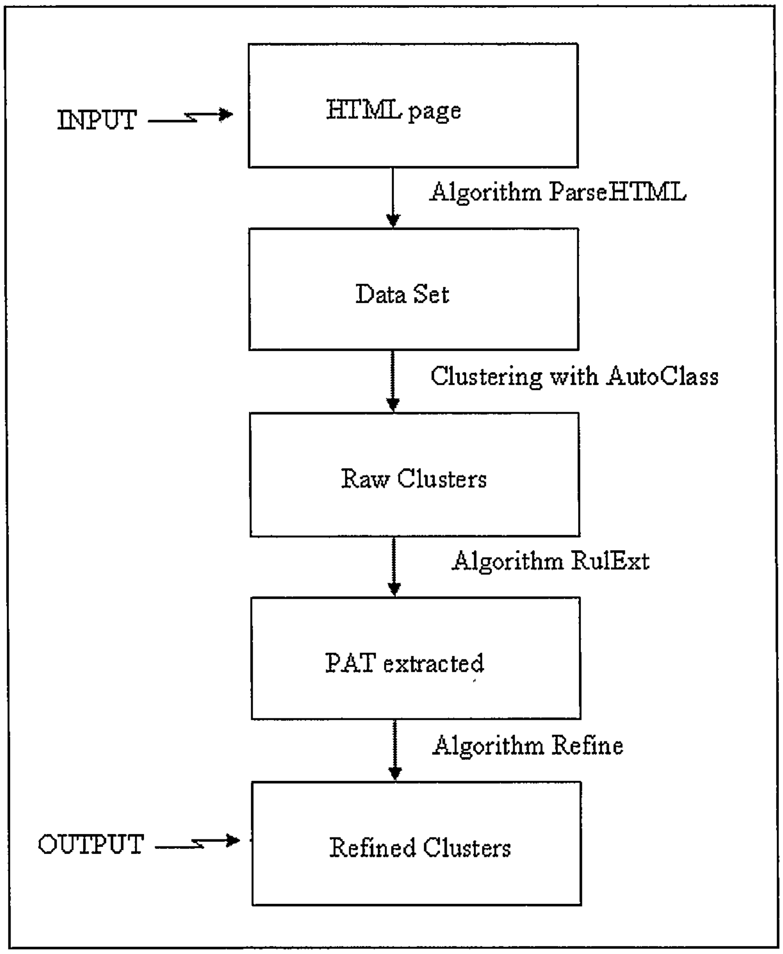


Figure 4.1: Overview of the proposed system

4.3.1 Implementation

ClusTex is implemented using Perl. Arrays were used to store text tokens and patterns. Each algorithm described above saved its output to a file, which was used by the next algorithm as its input.

4.4 Stages of Extraction

ClusTex has four stages of extraction as illustrated in Figure 4.1. In this section, these four stages are described.

1. **Data preparation**: In the first stage, data is prepared for clustering through tokenization and assignment of attribute values.
2. **Clustering**: In the second stage, the data tokens are separated into clusters based on their similarity to each other.
3. **Estimation of the Extraction Rule**: The third stage consists of the estimation of an extraction rule.
4. **Refinement**: The clusters are refined using the extraction rule in the fourth and final stage.

4.4.1 Stage 1: Data Preparation - Extracting Tokens and Assigning Attributes

As the first step in the process, the web document is parsed to extract all strings of text occurring between two HTML tags (hitherto known as *tokens*) and HTML tags that occur within the periphery of each token. These tokens are the instances of data which are "represented as ordered vectors of attribute values" [5]. Each attribute is the measurement of some property common to all data instances. For example, an attribute can be the presence or absence of an HTML tag in a token's range. These are the htmlAttributes. A second kind of attribute is defined on the format of a token, such as the presence or absence of an exclusively numeric token, or the presence or absence of punctuation, *etc.* These are termed orthographic features.

Instead of having simplistic attributes of the second kind (specified above), we can have a combination of these simplistic attributes to suit our needs and the nature of the items in a particular domain. These are the combiAttributes.

Taking the example of the Computer Science course listings domain, we see that various course attributes usually have a predictable format. Course codes usually have more than one capital letters followed by more than one numbers. So an attribute can be defined as whether a string does or does not start with one or more capital letters followed by more than one number. Course titles are *generally* comprised of more than eight letters and do not contain numbers. The attribute defined to account for this can be defined as the presence or absence in a string of the following properties:

- a length greater than eight
- starting with a capital letter followed by a mix of upper and lower case letters
- not containing any numbers

If a token has all these properties, a "1" will be assigned to this attribute, otherwise, a "0" will be assigned instead.

Table 4.1: Types of Attributes in the Course Listings Domain

htmlAttributes	CombiAttributes
IsTitle	IsCCode
IsBR	IsCTitle
IsA	IsProfName
IsInTable	IsGrade
IsTD	IsLocation

Table 4.1 gives some examples of these htmlAttributes and combiAttributes.

The algorithm ParseHTML is used at this stage (see Figure 4.2 for an abstract view of this algorithm). The following discussion gives a brief overview of how this algorithm works.

The input to ParseHTML is the web page that we want to extract information from, while the output is a .db2 file ready for clustering. ParseHTML reads all characters from the input file. Then it parses through the characters one by one. If a character is "<", it signifies that this is the beginning of a tag. Thus, the algorithm starts saving this character and the ones after it as an HTML tag. When a ">" is encountered, it signifies the end of the tag. At this point, the completed tag is saved into an array of tokens, Tok, and ParseHTML starts building a new text string (as opposed to a tag). All characters until the next "<" are saved as a string, and then it starts building a tag again. Once all tags and text are saved, ClusTex goes through this array once again. If a token from this array is a tag, it is checked against a list of tags. For each tag in that list, if there is a match, its presence is marked as a "1", otherwise its absence is marked by a "1". This generates HTMLattributes for the next text token in line. If the token is not a tag, the orthographic features are checked and combined to form CombiAttributes for the token. This process is repeated for

```
Algorithm: ParseHTML
Input: HTML file
Output: File containing tokenized text and assigned attributes for each
token
Begin
      While input_file is not empty
            Read character ch from input_file
            Save into char_array
            num_char = location of last character in char_array
      for i=0 to num_char
            Read ch at char_array[i]
            If (ch='<') /* This must be the start of an HTML tag
                  Start making a string for HTML tag
            Else if (ch='>') /* This must be the end of a tag
                  Complete and save tag in array Tok
                  num_tok=location of last token in Tok
                  Make an empty string to store text token.
            Else
                  Save ch in tag or text string.
                  If ch at char_array[i+1] is '<'
                        Save text string in array Tokens
                        num_tok=location of last token in Tok
      For j=0 to num_tok
            If token at Tok[j] is a tag
                  Check token against list of tags and assign 0 for
                  absence and 1 for presence for the next text token in
                  designated place in array Table
            Else if token at Tok[j] is not a tag i.e. is text
                  Check orthographic information and assign 0 or 1 for
                  combiAttributes
                  Save token at Table[j][0]
      Output Table into filename.db2
End ParseHTML
```

Figure 4.2: Algorithm ParseHTML: Parses HTML documents and tokenizes text

each token from the page. Finally, the tokens are listed with their attributes in the output file, ready to be clustered.

4.4.2 Stage 2: Clustering

A listings web page may have many kinds of tokens, but all of them might not be related to the listings themselves. For example, there may be an introductory paragraph, or contact information with a telephone number and an address at the end of the page which is unrelated to the listing itself. Furthermore, if a listing consists of multiple pages, these pages usually follow the same format with a side bar constantly displaying some standard links. We want to separate this information from the listing itself which we are interested in extracting.

A clustering software such as AutoClass aids us in attaining such an end. A good clustering separates all tokens from a page into different clusters, similar tokens falling into the same cluster: If the attributes are assigned correctly, most tokens describing the same thing will fall in one cluster, while most irrelevant tokens will fall in other clusters. Taking the example of the course listings domain, most course codes should fall in one cluster, most course titles will fall in another, and similarly for all the other values for each course attribute.

Clustering Model used by AutoClass

AutoClass [5, 19] is an example of the partitioning relocation clustering method that was discussed in Chapter 2. AutoClass, as an approach to unsupervised classification (clustering), is based on the classical mixture model and uses a Bayesian method which involves the discovery of optimal classes or clusters given the data and prior expectations. Instead of generating class descriptions from labeled examples, AutoClass deals with the problem of discovery of "natural" clusters in the data.

AutoClass works on data which can be represented as an ordered vector of attribute values. Attributes represent measurements of properties common to all data instances. These measurements can be discrete values (such as "true" or "false"), integer values, or real numbers. AutoClass does not deal with relational data, i.e. attributes which relate one data instance to another.

4.4.3 Stage 3: Discovery of Extraction Rules

Although clustering separates most of the information to be extracted, some tokens might fall in clusters where they do not belong due to their different format. For example, a concise course title such as UNIX might fall in the cluster for course codes, a professor's name might fall into the cluster for course titles, *etc.* Thus, after the tokens have been clustered, the clusters need to be refined.

We also need to find the order of these extracted values, i.e., the course code, title, and professor's name that belong together. In other words, we need to discover the extraction rules to answer this question: Which token fills which slot?

I use patterns of cluster numbers to represent extraction rules. The term *patterns* is taken to mean various concepts in the literature. I define this and other relevant terms below to tie them to the specific context of this thesis.

Definition 1: *Pattern*: A pattern of size n is a sequence of n numbers.

Definition 2: *Invalid Pattern*: A pattern is considered invalid if a number occurs more than once in it.

Definition 3: *Valid Pattern*: A pattern is considered valid if all numbers in it occur once only.

My contention is that since the listing is semi-structured data, i.e., *most* courses have the same attributes listed, and these values are usually in a similar format, the tokens should follow a certain sequence in the document. If the tokens themselves are replaced by the number of the cluster in which they occur, we are left with a list of cluster numbers in the order of the occurrence of tokens. We then look for a pattern that recurs many times in this list of cluster numbers. The algorithm that has been developed to achieve this end is RulExt and an abstract view of this algorithm is shown in Figure 4.3. The following paragraphs give a brief overview of this algorithm.

The input for RulExt is a file containing the cluster numbers of tokens in the order of occurrence of the tokens. RulExt outputs *PAT*, the estimated extraction rule which tells ClusTex how to related different tokens to each other.

RulExt starts with a higher than expected pattern size, P_S. Until there are more numbers in the input file, ClusTex makes a pattern *PT* of size P_S by reading cluster numbers one by one from the input file. If a cluster number occurs more than once in the pattern, it is rendered invalid. The first number in *PT* is then discarded and the next number from the file is added to *PT*. If *PT* is valid, it is saved into an array of patterns, PT_array, if it is not already there. The next step is to find matches for *PT* in the rest of the data list. RulExt makes a $TEST$ pattern from P_S cluster numbers occurring after the occurrence of *PT*, and compares *PT* to $TEST$. If they match, $TEST$ is deleted and is assembled again from P_S numbers occurring after the earlier pattern that was designated $TEST$. If, however, there is a mismatch, not only is $TEST$ deleted, but the first character in *PT* is also deleted and the first

```
Algorithm: RulExt
Input: File containing cluster number listing for tokens
Output: PAT - estimated extraction rule
Begin
      PT="", TEST=""
      For i=P_S to 1 /* P_S is higher than expected size for PAT
            while input_file is not empty
                  Read character ch from input_file
                  While size_of_PT < P_S
                        Add ch to PT
                  If PT is invalid
                        Delete first ch in PT
                  Else if PT is valid
                        Save PT into PT_array if not already there
                        While (size_of_PT = P_S)
                              While (size_of_TEST < P_S)
                                    Add ch to TEST
                        If (size_of_TEST = P_S)
                              If (PT == TEST)
                                    Empty TEST
                                    Save loc_TEST in PT_array
                                    Update occ_count in PT_array
                              Else
                                    Empty TEST
                                    Delete first ch in PT
            If (num_valid_patterns > 0.5 * num_total_patterns)
                  Output PT_array
                  Exit for loop
      If (one PT in PT_array with highest occurrence)
            Output PT as PAT
      Else if (there is a tie for highest occurrence)
            Use heuristics to break tie and output PAT
      Else
            Output the first PT with highest occurrence as PAT
End RulExt
```

Figure 4.3: Algorithm RulExt: Extracts the Extraction Rule - PAT

number after the earlier PT from the file is read in to rebuild PT, and the process is repeated to match a new $TEST$ pattern with PT.

After all patterns of size P_S have been accounted for, the number of valid and invalid patterns are counted. If more than half of total patterns are valid, then P_S is confirmed as the size for the extraction rule, PAT. If valid patterns form half or less than half of the total patterns, P_S is decreased by one, and the algorithm looks for patterns of lower size until the number of valid patterns exceeds the number of invalid patterns.

The next step is to find which valid pattern is PAT. RulExt goes through $PTarray$, the list of valid patterns. If there is one pattern that occurs the most times, this is taken to be PAT. If there is a tie, a rule of thumb is used to break the tie. This rule of thumb is that an identifying characteristic of an instance should occur towards its beginning. For example, in the case of a course entity, we expect the course code to occur towards the beginning of a course listing. If the rule of thumb is unable to break the tie, then RulExt picks the first most prolific item as PAT.

The first phase of RulExt in which patterns are formed from the data is best explained by a simple example. The estimation of PAT is discussed in further detail in the two subsequent sections.

Example

.This example shows the extraction of PAT from a very short course listing from the Computer Science domain. Suppose the listing (strings only, in order of occurrence) is as follows:

1. CPSC203

2. Introduction to Computer Science

3. CPSC457

4. Operating Systems

After parsing and clustering, we should get the clusters shown in Table 4.2.

Table 4.2: Example 1: Resulting clusters

Cluster Number	Member	Member
Cluster 0	CPSC203	CPSC457
Cluster 1	Introduction to Computer Science	Operating Systems

If we replace the tokens by the number of the cluster in which they occur, we get the following listing:

1. 0

2. 1

3. 0

4. 1

In this listing, we see three patterns of size 2: 01, 10 and 01. Note that trying to find patterns of size 3 would lead to two patterns being extracted, 010 and 101, both of which are invalid.

Finding the Optimal Extraction Rule

From domain knowledge, we have a general idea about how many tokens occur in one listing. For instance, eight tokens is the highest number seen in all of the course listings pages examined. Let this number be called P_S. We start the search with expected pattern size of $P_S + 1$ and try to find patterns of this size in the list of cluster numbers.

The idea is that all necessary parts of the listing must occur in separate clusters. Thus, in the case of a correct extraction rule, PAT should contain only one occurrence of a particular cluster number. If there are two or more occurrences of a cluster number, that signifies that either a token is incorrectly clustered, or the pattern contains tokens from different entities in the listing. As was previously stated, two or more occurrences of a cluster number in a pattern render it invalid. In the ongoing example, such a count is shown in Table 4.3.

Table 4.3: Example 1: Array containing valid patterns

Pattern	Total occurrences	Last occurrence	1st occurrence	2nd occurrence
01	2	3	1	3
10	1	2	2	

If the number of invalid patterns is close to 100% of all patterns, and very few valid patterns are revealed, it is assumed that we are working with the wrong pattern size. The pattern size P_S is decreased by one and the process is repeated until the number of valid patterns exceeds the number of invalid patterns.

In all cases, we find that the most frequently occurring valid pattern is the desired extraction rule PAT, i.e., the correct order of occurrence in the document. For each such search of patterns of size k, the details of all valid patterns, the number of occurrences, and the point of each occurrence are documented.

Tie for PAT

In case there is a tie between two patterns for PAT, the extraction rule to be estimated, a heuristic, or rule of thumb, is used to decide on the optimal pattern. More specifically, from the domain knowledge, we have certain expectations about order of occurrence of different types of tokens, i.e., we expect the identifying characteristic of an instance or the primary key to occur at the beginning of the instance. Thus, in the case of course listings, it is expected that the course code will occur close to the beginning of a pattern or rule. For example, if there is a tie between two patterns, P1 and P2, and P1 contains the number of the cluster containing mostly course codes towards its beginning, and P2 contains the same information towards its end, we can deduce that P1 is probably the correct extraction rule.

This heuristic is very effective in domains where we can expect certain tokens to occur towards the start of PAT. This is evident in Chapter 5 in which the results of my experiments are presented. It is seen that in the one web page where the start of the pattern cannot be anticipated, we have to resolve the tie by picking the first pattern with the highest number of occurrences.

4.4.4 Stage 4: Refinement

After deciding on PAT of size k (and in the process, the desired clusters), we need to cull out irrelevant information from these clusters, and also reassign tokens that had been incorrectly classified earlier to the right cluster. This reassignment only takes place for patterns that differ from PAT by no more than $log_2 k$ (rounded down) digits. This value is determined by trial and error. No distinction is made between valid or invalid patterns when it comes to reassignment.

The algorithm Refine is developed to re-assign tokens correctly in three phases. Figure 4.4 shows the pseudocode for this algorithm. An overview of the three phases of the algorithm, Refine, is described next.

```
Algorithm: Refinement
Input: File containing cluster number listing for tokens, PAT
Output: Text file containing refined output
Begin
      From start of input_file to 1st occurrence of PAT
            Check all patterns with PAT
            If (difference < $\log_2 k$ char)
                  Change cluster numbers for mismatches
      From 1st occurrence of PAT to last occurrence of PAT
            If (difference in location of 2 PAT occurrences>PAT_SIZE)
                  Compare patterns between those 2 occurrences with PAT
                  If (difference < $\log_2 k$ char)
                        Change cluster numbers for mismatches
      From last occurrence of PAT to end of input_file
            Check all patterns with PAT
            If (difference < $\log_2 k$ char)
                  Change cluster numbers for mismatches
      Delete cluster numbers not appearing in PAT
      Read Token text from array TOK and combine with cluster numbers
      Reconstruct data instances by printing all tokens in one pattern
      in one line
End Refinement
```

Figure 4.4: Algorithm Refine: Refines clusters and outputs result

1. The first phase starts from the beginning of the cluster number listing and ends at the first occurrence of PAT. In all mismatching patterns that fit the above criterion, the mismatching digits are changed to match digits from PAT. In essence, the incorrectly clustered token is taken out of the original cluster and reassigned to the correct one.

2. In the second phase, the incorrectly clustered tokens occurring within the bulk of the listings are refined. The interval between each occurrence of PAT is checked for mismatched tokens. If the interval between two occurrences is k (where k is the size of PAT), it tells us that two occurrences of PAT occurred side by side. If it is more than twice k, it tells us that there might be another pattern or patterns between the two occurrences that may be candidates for re-assignment. Thus, the same process from phase 1 is repeated for each interval between two occurrences of PAT where the interval is greater than twice k, and incorrectly clustered tokens are reassigned.

3. In the third phase, the patterns occurring *after* the last occurrence of PAT are checked in the above way and tokens are re-assigned to different clusters if the need arises.

After the refinement is complete, all clusters that do not figure in PAT are deleted. Data instances are recreated from the tokens using clusters and PAT, and are reported.

Chapter 5

Experimental Results

In this chapter, the experimental results are reported. The first section describes the testing environment. The next section presents the data sets on which ClusTex was tested. The third section presents details of experiments on each of the data sets. In the fourth section, results are discussed and compared with those reported in the literature.

5.1 The Testing Environment

To gauge the performance and efficiency of the proposed approach, experiments were conducted on computers with the following features:

- Architecture: 200 MHz Sun UltraSPARC-II
- Main Memory: 256 MB RAM
- Operating System: Solaris 8

5.2 The Data sets

ClusTex is designed to work on web pages containing semi-structured data listings, and is not restricted to any particular domain. However, to test my approach, the following three domains were selected:

1. Computer Science course listings from University web sites

2. Cell phone sales listings

3. Marathon Listings

Experiments are primarily reported on course listings web pages. However, one experiment on a cell phone sales web page and two experiments on marathon listing web pages are also shown. ClusTex was tested on the following web pages.

1. Computer Science course listings from University of Calgary, Canada

2. Computer Science course listings from New York University, U.S.A.

3. Computer Science course listings from Duke University, U.S.A.

4. Computer Science course listings from Columbia University, U.S.A.

5. Buy.com Cell Phone Sales web page

6. The Running Page - 1999 Marathon Schedule

7. MarathonGuide.com - US Marathons Races Directory and Schedule

These web pages are chosen because they meet the minimum requirement for being processed by this approach. These pages were written in HTML, and displayed semi-structured data, while the features of the data instances were displayed in the same order, there were also some missing features and some multi-valued features in addition to information irrelevant to the listings.

Another reason to choose these pages was that they exhibited differences in the format and content of the presented information as well. For example, in the course listings domain, the page from University of Calgary had data instances with two attributes, and the data was in an HTML table. The page from New York University had data instances with three attributes, and the data was in an HTML table as well. The page from Duke University had data instances with two, and in some cases three, attributes but the data was not in a table. Finally, the Columbia University data had eight attributes and the data was presented in an HTML table.

As mentioned earlier, the approach is tested on one web page from the domain of cell phone sales listings. Most of the information presented in this page was structured. The data in this page was presented in an HTML table and had a maximum of three attributes per instance.

Two pages were tested from the marathon listings domain. Both of these pages were also presented in tables. However, the page from the web site "The Running Page" had many missing data tokens and tokens belonging to the same category have varying formats. In addition, many tokens belonging to different categories had similar formats. These characteristics helped highlight the limitations of the system.

For the course of these experiments, all tokens were considered both as separate entities and as part of a single data instance. Results are reported for both scenarios.

5.3 Evaluation Criteria

Researchers in the IE field commonly report their results by using metrics such as Precision, Recall, and the geometrical average of these two, the F value.

In simple words, *precision* is the general correctness of the output. Buttler *et al.* [2] defines it as the percentage of correct extractions. Precision is formally defined as follows.

$$Precision\ (P) = \frac{Total\ number\ of\ correct\ values\ extracted}{Total\ number\ of\ values\ extracted}$$

P is a value between 0 and 1, 0 signifying all false positives and 1 signalling no false positives.

Recall, simply stated, is the prediction of correct values. Buttler *et al.* [2] defines it as "the percentage of positive instances of the target concept that are correctly identified". It is formally defined as follows.

$$Recall\ (R) = \frac{Number\ of\ correct\ values\ extracted}{Total\ number\ of\ possible\ correct\ values}$$

R is also a value between 0 and 1, where 0 means no correct values were predicted and 1 means all correct values were predicted.

The *F value* is defined as follows.

$$F = \frac{(\beta^2 + 1)P * R}{\beta^2 P + R}$$

where β is the weight of R over P, a value of $\beta = 1$ means that recall and precision are weighted equally. Researchers usually report the F1 value where β is taken to be 1.

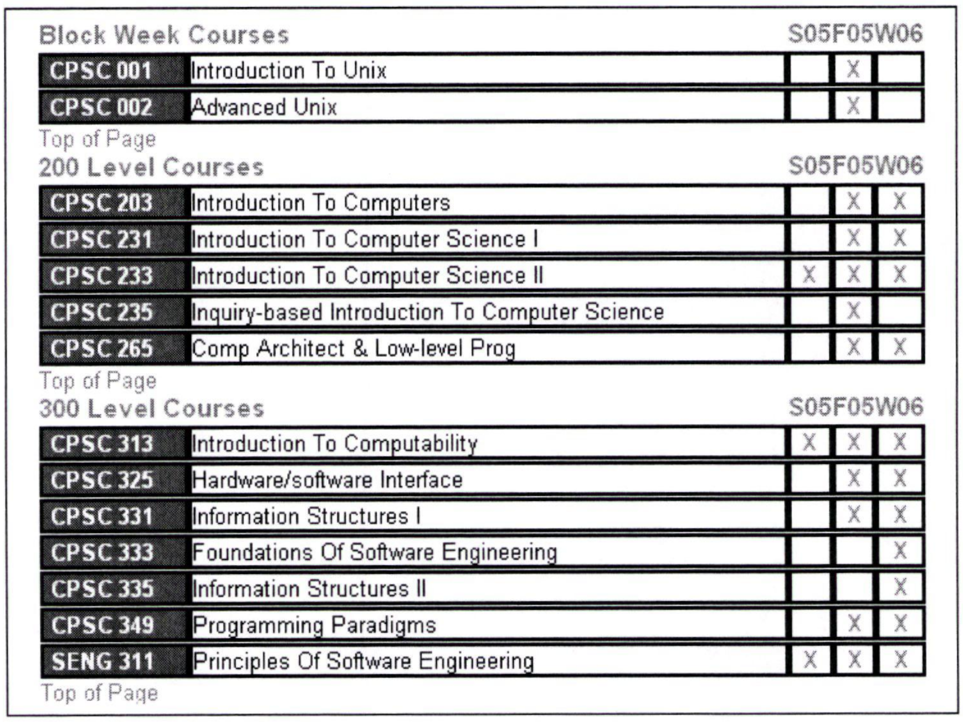
These values are usually reported as percentages. To present my results and compare these with other reported in the literature, I use *P, R* and *F1* as well.

5.4 Experiments and Results

5.4.1 University of Calgary web page

The Computer Science course listings from the University of Calgary [33] contain course codes and their respective course titles. This information is presented in an

HTML table. A screen shot of this page is shown in Figure 5.1.



Block Week Courses

		S05	F05	W06
CPSC 001	Introduction To Unix		X	
CPSC 002	Advanced Unix		X	

Top of Page

200 Level Courses

		S05	F05	W06
CPSC 203	Introduction To Computers		X	X
CPSC 231	Introduction To Computer Science I		X	X
CPSC 233	Introduction To Computer Science II	X	X	X
CPSC 235	Inquiry-based Introduction To Computer Science		X	
CPSC 265	Comp Architect & Low-level Prog		X	X

Top of Page

300 Level Courses

		S05	F05	W06
CPSC 313	Introduction To Computability	X	X	X
CPSC 325	Hardware/software Interface		X	X
CPSC 331	Information Structures I		X	X
CPSC 333	Foundations Of Software Engineering			X
CPSC 335	Information Structures II			X
CPSC 349	Programming Paradigms		X	X
SENG 311	Principles Of Software Engineering	X	X	X

Top of Page

Figure 5.1: Web Page with Course Information from University of Calgary

After the tokenization of the input HTML file and the assignment of attribute values to each token, AutoClass was run on the data set and the result contained three clusters. Cluster 0 contained primarily course codes, Cluster 1 contained mostly course titles, and Cluster 2 contained some course names interspersed with some irrelevant text from the page (text from navigation bars, introductory paragraph, *etc.*). The cluster distribution of the tokens from this page is shown in Table 5.1.

The patterns discovered when tokens were replaced by the number of the cluster in which they occurred, are shown in the screen shot in Figure 5.2. Attempts to find

Table 5.1: University of Calgary: Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	223
1	215
2	56

patterns of size greater than 2 led to all resulting patterns being invalid.

```
We have 6 patterns of size 2 now
0: 2 1 has occured 7 times with first occurence at location: 6
1: 1 2 has occured 12 times with first occurence at location: 7
2: 2 0 has occured 20 times with first occurence at location: 15
3: 0 1 has occured 208 times with first occurence at location: 16
4: 1 0 has occured 203 times with first occurence at location: 17
5: 0 2 has occured 15 times with first occurence at location: 20
There were 28 invalid patterns also
```

Figure 5.2: Patterns discovered in the University of Calgary Input using Cluster numbers

Since the pattern "0 1" occured most prolifically, it was taken to be PAT, the sequence of tokens that we are seeking. All patterns that differ by $log_2 2$ or one digit such as "0 2" and "2 1" or any invalid patterns were converted to match "0 1" by reassigning the mismatched token to the correct cluster. After refinement, the contents of the clusters were as follows.

Table 5.2: University of Calgary: Post-Refinement Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	225
1	227
2	33

Since cluster 2 does not occur in PAT, it was discarded and the clusters 0 and 1 were reported.

Results

Looking at the cluster contents, it was seen that cluster 0 contained all course codes, and cluster 1 contained course titles. Referring back to the web page, it was confirmed that “0 1” was indeed the right pattern, as the information in the page was laid out with course codes followed by course titles.

Table 5.3 shows us the cluster-wise results. In this and all subsequent results tables, TE denotes total number of tokens extracted, CE denotes the total number of correctly extracted tokens, and TC is the number of all possible correct tokens that could be extracted from this page. P is precision, R is recall, and F1 is the F1 value.

Table 5.3: University of Calgary: Cluster-wise Results

Cluster#	Info Type	TE	CE	TC	P	R	F1
0	Course Codes	225	220	220	97.78%	100%	98.77%
1	Course Titles	227	220	220	96.92%	100%	98.44%

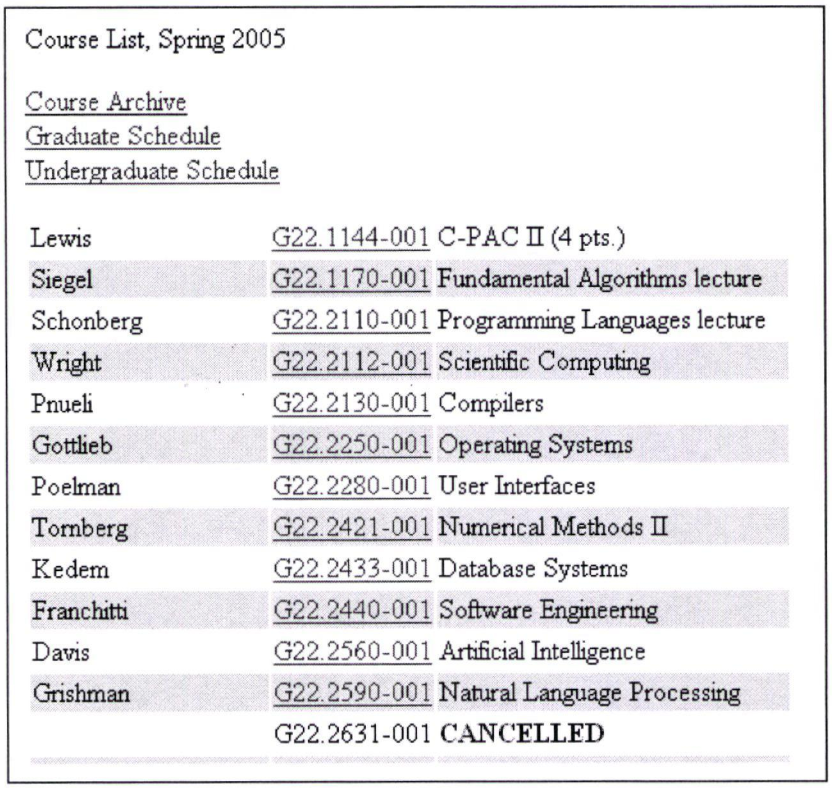
It is evident from the information shown in this table that we were able to achieve highly accurate results in this experiment. The 100% recall for both clusters shows that the system was able to extract all possible correct values from this web page. The slightly lower precision values signify that some false positives were extracted as well. Averaging the above values gives us $P = 97.35\%$, $R = 100\%$, and $F1 = 98.48\%$.

Rebuilding course entities using PAT, 227 output pairs were extracted. Since there were a total of 220 possible course entities in the page, seven of the extracted pairs were false positives. For the re-constructed courses entities, the following values were calculated: $P = 96.92\%$, $R = 100\%$, and $F1 = 98.43\%$. Thus, once again, the

100% recall signifies that all the course code-course title pairs were extracted correctly while the lower precision values shows the existence of seven false positives as well.

5.4.2 New York University web page

The Computer Science course listings from New York University [40] contain course codes, their respective course titles, and the last names of faculty numbers teaching those courses. This information is presented in an HTML table. Figure 5.3 shows a screen shot of this page.



Course List, Spring 2005

Course Archive
Graduate Schedule
Undergraduate Schedule

Lewis	G22.1144-001	C-PAC II (4 pts.)
Siegel	G22.1170-001	Fundamental Algorithms lecture
Schonberg	G22.2110-001	Programming Languages lecture
Wright	G22.2112-001	Scientific Computing
Pnueli	G22.2130-001	Compilers
Gottlieb	G22.2250-001	Operating Systems
Poelman	G22.2280-001	User Interfaces
Tornberg	G22.2421-001	Numerical Methods II
Kedem	G22.2433-001	Database Systems
Franchitti	G22.2440-001	Software Engineering
Davis	G22.2560-001	Artificial Intelligence
Grishman	G22.2590-001	Natural Language Processing
	G22.2631-001	**CANCELLED**

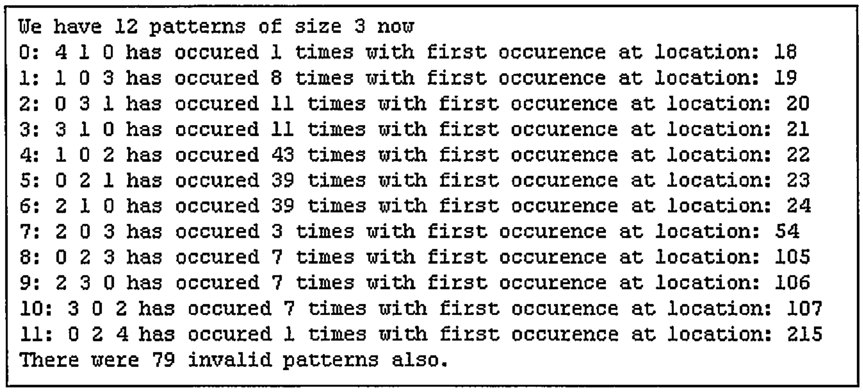
Figure 5.3: Web Page showing Course Information from New York University

After the clustering was performed by AutoClass, the results showed the existence of five clusters. Table 5.4 shows the number of tokens contained in each cluster.

Table 5.4: New York University: Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	68
1	60
2	54
3	23
4	14

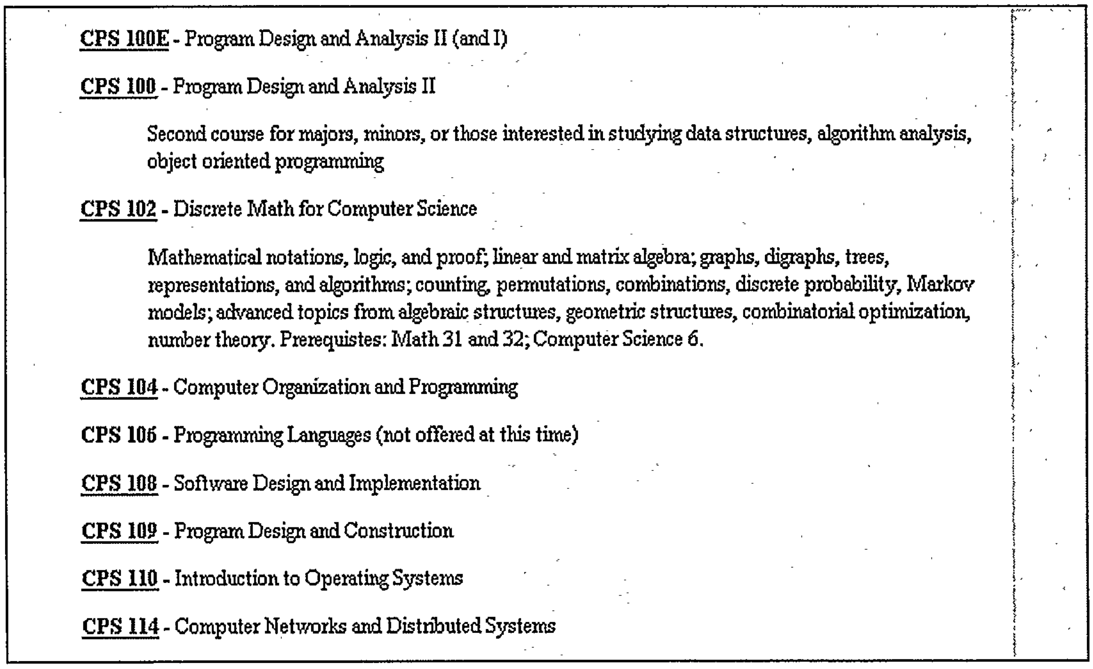
Figure 5.4 shows the patterns of size 3 found in the output when the token text was replaced by the number of the cluster in which the token occurred. Attempts to find patterns of a size larger than 3 led to almost all patterns being invalid i.e. containing more than one occurrence of a cluster number.



```
We have 12 patterns of size 3 now
0: 4 1 0 has occured 1 times with first occurence at location: 18
1: 1 0 3 has occured 8 times with first occurence at location: 19
2: 0 3 1 has occured 11 times with first occurence at location: 20
3: 3 1 0 has occured 11 times with first occurence at location: 21
4: 1 0 2 has occured 43 times with first occurence at location: 22
5: 0 2 1 has occured 39 times with first occurence at location: 23
6: 2 1 0 has occured 39 times with first occurence at location: 24
7: 2 0 3 has occured 3 times with first occurence at location: 54
8: 0 2 3 has occured 7 times with first occurence at location: 105
9: 2 3 0 has occured 7 times with first occurence at location: 106
10: 3 0 2 has occured 7 times with first occurence at location: 107
11: 0 2 4 has occured 1 times with first occurence at location: 215
There were 79 invalid patterns also.
```

Figure 5.4: Patterns discovered in the New York University Input using Cluster numbers

Pattern 4, "1 0 2" occured most prolifically and was taken to be *PAT*. All those patterns that differed from PAT by $log_2 3$ or one (rounded down) digit, such as "1 0 3"



CPS 100E - Program Design and Analysis II (and I)

CPS 100 - Program Design and Analysis II

Second course for majors, minors, or those interested in studying data structures, algorithm analysis, object oriented programming

CPS 102 - Discrete Math for Computer Science

Mathematical notations, logic, and proof; linear and matrix algebra; graphs, digraphs, trees, representations, and algorithms; counting, permutations, combinations, discrete probability, Markov models; advanced topics from algebraic structures, geometric structures, combinatorial optimization, number theory. Prerequistes: Math 31 and 32; Computer Science 6.

CPS 104 - Computer Organization and Programming

CPS 106 - Programming Languages (not offered at this time)

CPS 108 - Software Design and Implementation

CPS 109 - Program Design and Construction

CPS 110 - Introduction to Operating Systems

CPS 114 - Computer Networks and Distributed Systems

Figure 5.5: Web Page showing Course Information from Duke University

Table 5.7: Duke University: Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	103
1	91
2	34
3	31

any false positives. For cluster 1, two correct instructors' names were missed and, hence, led to a recall of 96.83%. The seven false positives lowered the precision to 92.65%. In cluster 2, no false positives were reported which leads to a perfect precision. However, the system failed to extract six course titles which lowered the recall. Counting all token extractions together gave us $P = 97.55\%$, $R = 95.51\%$, and $F1 = 96.75\%$.

Rebuilding course entities using PAT, 61 output tuples were extracted. There were a total of 68 possible course entities in the page. There were no false positives in this case. For the re-constructed course entities, the following values were calculated: $P = 100\%$, $R = 89.71\%$, and $F1 = 94.57\%$. The lower recall shows the seven missed instructor-code-title tuples.

5.4.3 Duke University web page

The Computer Science course listings from Duke University [39] contain course codes and their respective course titles. This example is different from that of the University of Calgary in that the information is not presented in an HTML table. Figure 5.5 shows a screen shot of this page.

After the clustering was performed by AutoClass, the results showed the existence of four clusters. The following table shows the number of tokens contained in each cluster.

Figure 5.6 shows the patterns of size 2 found in the output when the token text was replaced by the number of the cluster in which the token occurred. Attempts to find patterns of a size larger than 2 led to almost all patterns being invalid i.e. containing more than one occurrence of a cluster number.

and "3 0 2" or any invalid patterns were re-classified to match it. After refinement, the token distribution through the clusters is shown in Table 5.5.

Table 5.5: New York University: Post-Refinement Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	68
1	68
2	61
3	8
4	13

Clusters 3 and 4 do not occur in PAT and were discarded. Clusters 1, 0, and 2 were reported.

Results

Looking at the cluster contents, it was seen that cluster 0 contained all course codes, cluster 1 contained names of faculty members, and cluster 2 contained course titles. Referring back to the web page, it was confirmed that "1 0 2" was correctly estimated to be *PAT*, as the information in the page was laid out with the faculty members' names followed by the codes and titles of courses that they were teaching. The results for each cluster are shown in Table 5.6.

Table 5.6: New York University:Cluster-wise Results

Cluster#	Info Type	TE	CE	TC	P	R	F1
0	Course Codes	68	68	68	100%	100%	100%
1	Instructors' Names	68	61	63	92.65%	96.83%	95.69%
2	Course Titles	61	61	68	100%	89.7%	94.57%

In this table, we see that for cluster 0 which contained mostly course codes, the system achieved the extraction of all possible correct course codes and did not report

```
We have 11 patterns of size 2 now
0: 0 2 has occured 9 times with first occurrence at 0
1: 2 3 has occured 9 times with first occurrence at 2
2: 3 2 has occured 9 times with first occurrence at 4
3: 3 0 has occured 6 times with first occurrence at 34
4: 0 3 has occured 5 times with first occurrence at 35
5: 2 0 has occured 7 times with first occurrence at 43
6: 2 1 has occured 3 times with first occurrence at 57
7: 1 0 has occured 89 times with first occurrence at 58
8: 0 1 has occured 88 times with first occurrence at 59
9: 1 2 has occured 1 times with first occurrence at 118
10: 1 3 has occured 1 times with first occurrence at 186
There were 31 invalid patterns also.
```

Figure 5.6: Patterns discovered in the Duke University Input using Cluster numbers

Pattern 7, "1 0" occured most prolifically and was designated to be *PAT*, the pattern of tokens that we are looking for. All those patterns that differed from PAT by $log_2 2$ or one digit, such as "3 0", "2 0", "1 2", and "1 3", or any invalid patterns that fit the criteria, were re-classified to match it. The token distribution through clusters after refinement is shown in Table 5.8.

Table 5.8: Duke University: Post-Refinement Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	105
1	94
2	31
3	29

Clusters 2 and 3 do not figure in PAT and were discarded. Clusters 1 and 0 were reported.

Results

Looking at the cluster contents, it was seen that cluster 1 contained all course codes and cluster 0 contained course titles.

This example brings to our attention one of the limitations of this approach. Upon referring to the web page, it was noticed that the listing not only contained the codes and titles for 93 courses, it also contained descriptions for 8 of these courses. However, the system was unable to extract these descriptions. There are two reasons for this.

1. Some descriptions were discarded during the tokenization process. The reason for this is that AutoClass is unable to process tokens whose length is longer than 200 characters and is liable to segmentation faults. Hence, my system routinely discards tokens containing more than 200 characters to make sure the program runs smoothly on the data. Some of these descriptions contained more than 200 characters and were taken out of the data set to make sure that the program runs correctly on the rest of the data set. Under the usual circumstances, the performance of the system is not effected by this because long tokens are usually introductory sentences in the web page and do not offer much information.

2. Another reason that the shorter descriptions were ignored for patterns is that they occurred very infrequently, as mentioned above. Therefore, AutoClass was not able to find a significant correlation between course descriptions, codes, and titles.

If we ignore the matter of missed course descriptions, PAT matched the rest of the information, because, in the page, the information was laid out such that course codes were followed by course titles. The results for separate extracted clusters are given in Table 5.9.

Table 5.9: Duke University:Cluster-wise Results

Cluster#	Info Type	TE	CE	TC	P	R	F1
0	Course Titles	105	93	93	88.57%	100%	93.94%
1	Course Codes	94	92	93	97.87%	98.92%	98.4%
-	Course Descriptions	0	0	8	-	0%	-

The information in this table shows the results of the system when applied on a data set with certain limitations, i.e., not enough tokens reported together for the system to associate them with each other. In cluster 0, mostly course titles were reported with twelve false positives. However, all course titles available in the data set were correctly extracted in cluster 0. Thus, for this cluster, we had a perfect recall but lower precision since all of the output was not correct. In cluster 1, course codes were reported. There were two false positives, a fact that lowered the precision slightly. One course code present in the data set was missed resulting in a lowered recall for this particular cluster. Since course descriptions were not reported in the output, the third cluster is empty to show the cluster that should have contained descriptions. Since none were extracted, we have an undefined precision (division by 0) and the recall is 0.

For this example, two sets of results are presented for the case when we average results for all tokens extracted. If we consider the course descriptions to be part of the information that should have been extracted, we get $P = 92.96\%$, $R = 95.36\%$, and $F1 = 94.14\%$.

If, on the other hand, we decide that descriptions were not the factual information that we wanted to extract, we get $P = 92.96\%$, $R = 99.46\%$, and $F1 = 96.1\%$ instead.

Rebuilding course entities using PAT, 104 output pairs were extracted. Since there were a total of 93 possible course entities in the page, eleven of the extracted pairs were false positives. Out of the 93 correct pairs, a further 8 were considered incorrect because the description was not taken into account. For the re-constructed courses entities, the following values were calculated: $P = 80.77\%$, $R = 91.3\%$, and $F1 = 85.72\%$. From these values we deduce that 80.77% of the extracted pairs were correct while the system was able to extract 91.3% of all possible correct tuples in the data set.

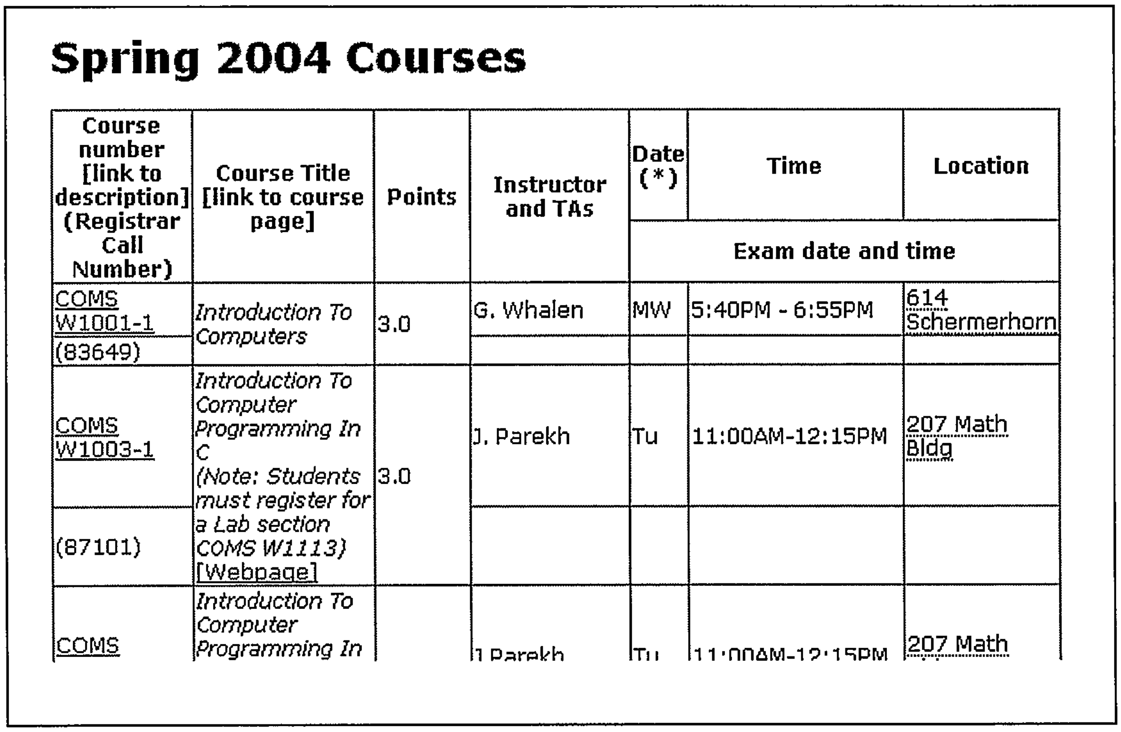
It should be noted that I have been strict about declaring the correctness or incorrectness of extracted tuples. In many other existing approaches in the literature, correct and partially correct (take the example of the eight tuples with missing descriptions above) are both taken to mean correct. Here, partially correct tuples are considered incorrect.

5.4.4 Columbia University web page

The Computer Science course listings from Columbia University [38] contain course codes, their respective course titles, credit points, instructor's name, day and time when the course is taught, location and a reference number. This information is presented in an HTML table and is shown in Figure 5.7.

After the clustering was performed by AutoClass, the results showed the existence of nine clusters. Table 5.10 shows the number of tokens contained in each cluster.

Figure 5.8 shows the patterns of size 8 found in the output when the token text was replaced by the number of the cluster in which the token occurred. Attempts to find patterns of a size larger than 8 led to almost all patterns being invalid.



Spring 2004 Courses

Course number [link to description] (Registrar Call Number)	Course Title [link to course page]	Points	Instructor and TAs	Date (*)	Time	Location
				Exam date and time		
COMS W1001-1	*Introduction To Computers*	3.0	G. Whalen	MW	5:40PM - 6:55PM	614 Schermerhorn
(83649)						
COMS W1003-1	*Introduction To Computer Programming In C (Note: Students must register for a Lab section COMS W1113)* [Webpage]	3.0	J. Parekh	Tu	11:00AM-12:15PM	207 Math Bldg
(87101)						
COMS	*Introduction To Computer Programming In*		J. Parekh	Tu	11:00AM-12:15PM	207 Math

Figure 5.7: Web Page showing Course Information from Columbia University

Table 5.10: Columbia University: Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	107
1	62
2	59
3	57
4	57
5	57
6	46
7	44
8	16

```
We have 9 patterns of size 8 now
0: 8 1 7 2 6 5 3 0 has occurred 1 times with first occurrence at location: 17
1: 1 7 2 6 5 3 0 4 has occurred 32 times with first occurrence at location: 18
2: 7 2 6 5 3 0 4 1 has occurred 32 times with first occurrence at location: 19
3: 2 6 5 3 0 4 1 7 has occurred 28 times with first occurrence at location: 36
4: 6 5 3 0 4 1 7 2 has occurred 29 times with first occurrence at location: 37
5: 5 3 0 4 1 7 2 6 has occurred 30 times with first occurrence at location: 38
6: 3 0 4 1 7 2 6 5 has occurred 30 times with first occurrence at location: 39
7: 0 4 1 7 2 6 5 3 has occurred 30 times with first occurrence at location: 40
8: 4 1 7 2 6 5 3 0 has occurred 30 times with first occurrence at location: 41
There were 88 invalid patterns also
```

Figure 5.8: Patterns discovered in the Columbia University Input using Cluster numbers

A tie was found between two patterns: "1 7 2 6 5 3 0 4" and "7 2 6 5 3 0 4 1". The tie was broken using the heuristic that an identifying characteristic of an entity (in this case the course code which belongs to cluster 1) should occur towards the beginning of PAT. The former pattern was assumed to be *PAT*, the pattern of tokens that we are seeking. All those patterns that differ from PAT by up to $log_2 8$ or three digits, were re-classified to match it. The post-refinement cluster distribution is shown in table 5.11.

Cluster 8 does not occur in PAT and was discarded. Clusters 0, 1, 2, 3, 4, 5, 6, and 7 were reported in the order "1 7 2 6 5 3 0 4".

Results

Looking at the cluster contents, it was seen that cluster 0 contained mostly locations, cluster 1 contained course codes, cluster 2 contained credit points, cluster 3 contained time slots, and cluster 4 contained reference numbers for courses. Furthermore, cluster 5 contained days of lectures, cluster 6 contained instructors' names, and

Table 5.11: Columbia University: Post-Refinement Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	78
1	62
2	59
3	57
4	57
5	57
6	57
7	62
8	16

cluster 7 contained course titles. Referring back to the web page, it was confirmed that "1 7 2 6 5 3 0 4" was the right pattern, as the information in the page was laid out with the course code followed by course title, credit points, instructor's name, day and time, location, and the reference number. Table 5.12 shows the results for each cluster separately.

Table 5.12: Columbia University:Cluster-wise Results

Cluster#	Info Type	TE	CE	TC	P	R	F1
0	Locations	78	57	57	73.08%	100%	84.44%
1	Course Codes	62	62	64	100%	96.88%	98.41%
2	Credit Points	59	59	64	100%	92.19%	95.93%
3	Time	57	57	57	100%	100%	100%
4	Course Reference	57	57	64	100%	89.06%	94.22%
5	Days of the week	57	57	57	100%	100%	100%
6	Instructors' names	57	57	57	100%	100%	100%
7	Course Titles	62	59	64	95.16%	92.19%	93.65%

Cluster 0 shows the extraction of locations. The results show that all locations reported in the data set were correctly extracted with eleven false positives as well so that 73.08% of the reported tokens in cluster 0 were correctly identified as locations.

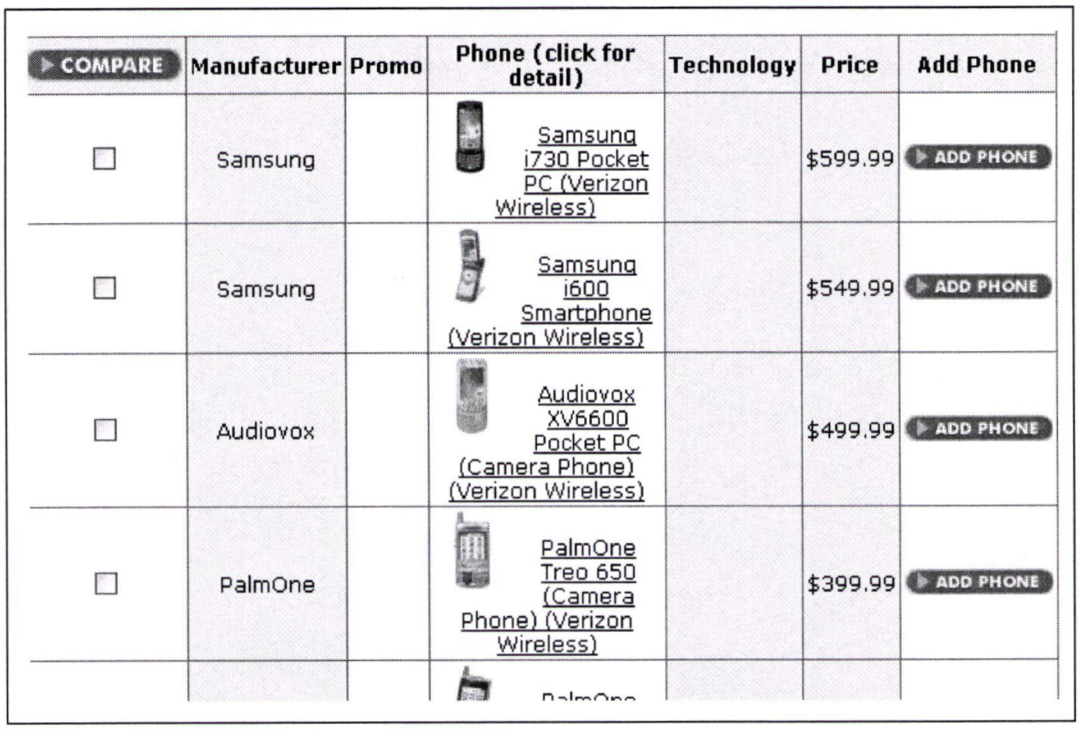
Cluster 1 shows the extraction of course codes where all tokens reported were correctly identified as codes. The system failed to extract 2 course codes which led to a lower recall. Cluster 2 shows that 92.19% of all correct credit points were extracted, and that all tokens in cluster 2 were correctly identified, i.e., no false positives were included. Cluster 3 shows the correct extraction of time slots. In cluster 4, all reported tokens were correctly classified while 89.06% of all reference numbers were extracted. In clusters 5 and 6, all correct days of the week and instructors' names respectively were correctly identified with no false positives. Finally, in cluster 7, 95.16% of reported tokens were correctly identified as titles, and the system failed to extract 7.81% of all possible correct titles. Averaging results for all tokens extracted, the cumulative results are $P = 95.09\%$, $R = 96.07\%$, and $F1 = 95.597\%$.

Rebuilding course entities using PAT, 62 output tuples were extracted. There were a total of 64 possible course entities in the page. There were five false positives. For the re-constructed courses entities, the following values were calculated: $P = 91.94\%$, $R = 89.06\%$, and $F1 = 90.48\%$. Thus, from these values, we understand that 91.94% of all extracted tuples were correct, while 89.06% of all possible correct tuples were extracted by the system.

5.4.5 Cell Phone Sales on Buy.com

I tested my approach on a cell phone sales domain as well, on a page from the web site buy.com [3]. This page contains phone listings with the manufacturer, the cell phone model, and the price listed for each phone. This information is presented in an HTML table and is shown in Figure 5.9.

After the clustering was performed by AutoClass, the results showed the existence



COMPARE	Manufacturer	Promo	Phone (click for detail)	Technology	Price	Add Phone
☐	Samsung		Samsung i730 Pocket PC (Verizon Wireless)		$599.99	ADD PHONE
☐	Samsung		Samsung i600 Smartphone (Verizon Wireless)		$549.99	ADD PHONE
☐	Audiovox		Audiovox XV6600 Pocket PC (Camera Phone) (Verizon Wireless)		$499.99	ADD PHONE
☐	PalmOne		PalmOne Treo 650 (Camera Phone) (Verizon Wireless)		$399.99	ADD PHONE
			PalmOne			

Figure 5.9: Web Page showing Cell Phones listings from Buy.com

of three clusters. Table 5.13 shows the number of tokens contained in each cluster.

Table 5.13: Cell Phone Listings: Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	144
1	126
2	82

Figure 5.10 shows the patterns of size 3 found in the output when the token text was replaced by the number of the cluster in which the token occurred. As in the previous experiments, all attempts to find patterns of a size larger than 3 led to all patterns being invalid i.e. containing more than one occurrence of a cluster number.

```
We have 3 patterns of size 3 now
0: 1 0 2 has occured 76 times with first occurence at location: 22
1: 0 2 1 has occured 76 times with first occurence at location: 23
2: 2 1 0 has occured 76 times with first occurence at location: 24
There were 123 invalid patterns also.
```

Figure 5.10: Patterns discovered in the Buy.com Input using Cluster numbers

A tie was found between three patterns: “1 0 2”, “0 2 1” and “2 1 0”. Since all cell phone sale sites use a different pattern of tokens, the system had to rely on the prolific occurrence of the pattern to discover PAT. When there was a tie and no heuristics were able to break this tie, “1 0 2” was chosen as PAT arbitrarily because it is the first most prolific pattern. All those patterns that differ from PAT by $log_2 3$ or 1 (rounded down) digit were re-classified to match it. The post-refinement cluster distribution is shown in Table 5.14.

Table 5.14: Cell Phone Listings: Post-Refinement Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	136
1	110
2	106

Results

Looking at the cluster contents, it was seen that the results showed perfect recall. Cluster 0 contained mostly cell phone models, cluster 1 contained the manufacturers' names, and cluster 2 contained prices. Referring back to the web page, it was confirmed that "1 0 2" was the right pattern, as the information in the page was laid out with the manufacturers' names followed by the phone model and its price. Table 5.15 shows the results for each refined cluster separately.

Table 5.15: Cell Phone Sales:Cluster-wise Results

Cluster#	Info Type	TE	CE	TC	P	R	F1
0	Cell Phone Models	136	104	104	76.47%	100%	86.67%
1	Manufacturers' Names	110	104	104	94.55%	100%	97.19%
2	Prices	106	104	104	98.11%	100%	99.05%

Cluster 0 shows extraction of cell phone models. The results show that 76.47% of all extracted tokens in cluster 0 were correct. The 100% recall shows that these correctly extracted tokens were all the possible tokens that could be extracted. Cluster 1 shows the extraction of manufacturers' names. According to the results, all names reported in the data set were correctly extracted. These formed 94.55% of the total tokens reported for this cluster. Cluster 2 shows the extraction of prices. Once again, all prices reported in the data set were extracted along with two false positives which meant that 98.11% of the output for this cluster was correct. Averaging results for all tokens extracted, the cumulative results are $P = 89.71\%$, $R = 100\%$, and $F1 = 94.30\%$.

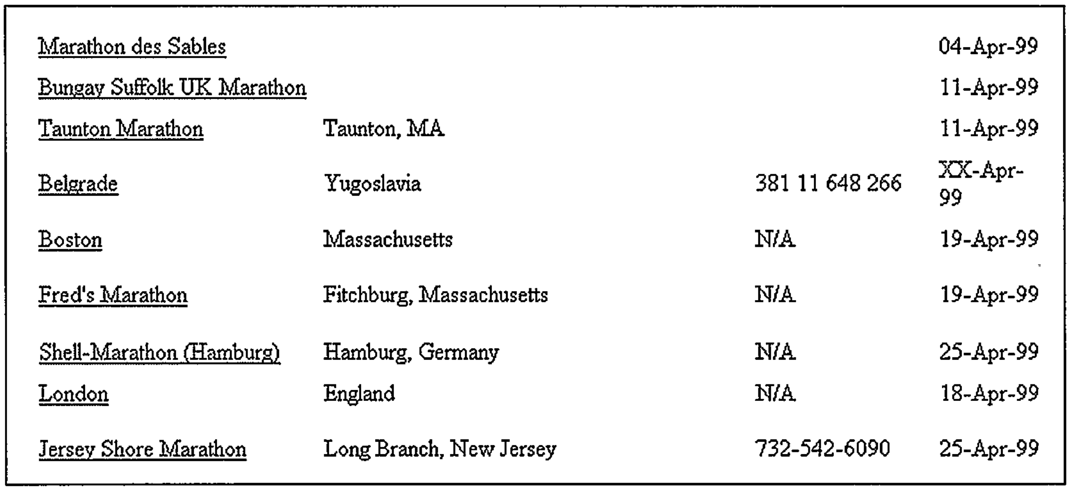
Rebuilding phone entities using PAT, 110 output tuples were extracted. There were a total of 104 possible phone entities in the page. There were six false positives. For the re-constructed phone entities, the following values were calculated: $P = 94.55\%$, $R = 100\%$, and $F1 = 97.19\%$. These values signify that out of all the output tuples, 94.55% were correct. Furthermore, the system was able to extract all possible correct tuples from the data set.

Although this seems like a good result, the correct PAT was extracted by chance and, in this particular case, does not highlight the power of this system. However, this example, like the earlier ones, showcases the suitability of clustering for a task such as this.

5.4.6 1999 Marathon Schedule from The Running Page

The approach is tested on two marathon listings web pages as well. This page from the website RunningPage.com [34] reports marathon listings containing the marathon name, the place where it is run, a contact phone number, and finally a date when it is scheduled. This information is presented in an HTML table and is shown in Figure 5.11.

This example is primarily included to show the limitations of this system. ClusTex depends on most of the tokens belonging to different categories to have different formats. If this condition is not fulfilled, the results worsen quickly. In this web page, the formats of marathon names and venues are similar in many cases. The following analysis shows the impact of this similarity on the output.



Marathon des Sables			04-Apr-99
Bungay Suffolk UK Marathon			11-Apr-99
Taunton Marathon	Taunton, MA		11-Apr-99
Belgrade	Yugoslavia	381 11 648 266	XX-Apr-99
Boston	Massachusetts	N/A	19-Apr-99
Fred's Marathon	Fitchburg, Massachusetts	N/A	19-Apr-99
Shell-Marathon (Hamburg)	Hamburg, Germany	N/A	25-Apr-99
London	England	N/A	18-Apr-99
Jersey Shore Marathon	Long Branch, New Jersey	732-542-6090	25-Apr-99

Figure 5.11: Web Page showing Marathon Listings from The Running Page

After the clustering was performed by AutoClass, the results showed the existence of eight clusters. Table 5.16 shows the number of tokens contained in each cluster.

Table 5.16: 1999 Marathon Listings: Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	49
1	45
2	33
3	24
4	21
5	15
6	7
7	2

Figure 5.12 shows the patterns of size 4 found in the output when the token text was replaced by the number of the cluster in which the token occurred. As in the previous experiments, all attempts to find patterns of a size larger than 4 led to all patterns being invalid.

```
We have 46 patterns of size 4 now
0: 6 2 4 3 has occurred 1 times with first occurrence at location: 9
1: 2 4 3 0 has occurred 3 times with first occurrence at location: 10
2: 4 3 0 1 has occurred 3 times with first occurrence at location: 11
3: 3 0 1 2 has occurred 2 times with first occurrence at location: 12
4: 1 2 0 5 has occurred 1 times with first occurrence at location: 14
5: 0 4 1 2 has occurred 1 times with first occurrence at location: 23
6: 4 1 2 0 has occurred 1 times with first occurrence at location: 24
7: 1 2 0 4 has occurred 1 times with first occurrence at location: 25
8: 2 0 4 1 has occurred 1 times with first occurrence at location: 26
9: 4 1 0 5 has occurred 1 times with first occurrence at location: 41
10: 1 0 5 2 has occurred 1 times with first occurrence at location: 42
11: 0 5 2 3 has occurred 3 times with first occurrence at location: 43
12: 5 2 3 0 has occurred 3 times with first occurrence at location: 44
13: 2 3 0 5 has occurred 4 times with first occurrence at location: 45
14: 3 0 5 1 has occurred 4 times with first occurrence at location: 46
15: 0 5 1 3 has occurred 4 times with first occurrence at location: 50
16: 5 1 3 0 has occurred 4 times with first occurrence at location: 51
17: 1 3 0 2 has occurred 4 times with first occurrence at location: 52
18: 3 0 2 4 has occurred 4 times with first occurrence at location: 53
19: 0 2 4 3 has occurred 2 times with first occurrence at location: 54
20: 0 1 2 3 has occurred 2 times with first occurrence at location: 58
21: 1 2 3 0 has occurred 2 times with first occurrence at location: 59
22: 1 3 0 4 has occurred 3 times with first occurrence at location: 71
23: 4 0 2 1 has occurred 1 times with first occurrence at location: 75
24: 0 2 1 3 has occurred 4 times with first occurrence at location: 76
25: 2 1 3 0 has occurred 4 times with first occurrence at location: 77
26: 1 3 0 5 has occurred 2 times with first occurrence at location: 78
27: 3 0 5 2 has occurred 3 times with first occurrence at location: 83
28: 0 2 4 1 has occurred 1 times with first occurrence at location: 91
29: 2 4 1 0 has occurred 1 times with first occurrence at location: 92
30: 4 1 0 2 has occurred 1 times with first occurrence at location: 93
31: 3 0 2 1 has occurred 2 times with first occurrence at location: 98
32: 2 3 0 1 has occurred 2 times with first occurrence at location: 104
33: 3 0 1 4 has occurred 2 times with first occurrence at location: 105
34: 0 1 4 3 has occurred 1 times with first occurrence at location: 106
35: 1 4 3 0 has occurred 1 times with first occurrence at location: 107
36: 0 1 4 2 has occurred 1 times with first occurrence at location: 110
37: 1 4 2 0 has occurred 1 times with first occurrence at location: 111
38: 4 2 0 5 has occurred 1 times with first occurrence at location: 112
39: 2 0 5 1 has occurred 1 times with first occurrence at location: 113
40: 4 3 0 5 has occurred 1 times with first occurrence at location: 124
41: 4 3 0 2 has occurred 2 times with first occurrence at location: 136
42: 0 4 1 3 has occurred 1 times with first occurrence at location: 161
43: 4 1 3 0 has occurred 1 times with first occurrence at location: 162
44: 1 5 2 7 has occurred 1 times with first occurrence at location: 186
45: 5 2 7 6 has occurred 1 times with first occurrence at location: 187
There were 74 invalid patterns also
```

Figure 5.12: Patterns discovered in the Running Page Input using Cluster numbers

A tie was found between eight patterns. The tie was broken using the heuristic that an identifying characteristic of an entity, in this case a marathon's name, close to the beginning of a pattern. Consequently, the pattern "5 1 3 0" was picked. All those patterns that differ from PAT by up to $log_2 4$ or two digits, were re-classified to match it. The post-refinement cluster distribution is shown in Table 5.17.

Table 5.17: 1999 Marathon Listings: Post-Refinement Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	48
1	49
2	19
3	27
4	13
5	32
6	7
7	2

Clusters 2, 4, 6 and 7 were discarded because they do not appear in PAT.

Results

Looking at the cluster contents, we see that the missing values and varying formats of the input test the limits of the data and reduce the precision and recall of the output. Cluster 0 contained mostly dates, cluster 1 contained mostly race venues, cluster 3 contained contact telephone numbers, and cluster 5 contained marathon names. Referring back to the web page, it was confirmed that "5 1 3 0" was the right pattern, as the information in the page was laid out with the marathons' names, followed by the venues, contact telephones, and dates. Table 5.18 shows the results for each refined cluster separately.

Table 5.18: The Running Page Listings: Cluster-wise Results

Cluster#	Info Type	TE	CE	TC	P	R	F1
0	Dates	48	47	47	97.92%	100%	98.95%
1	Venues	49	35	45	71.43%	100%	83.33%
3	Telephone Numbers	27	27	36	100%	75%	85.71%
5	Marathon Names	32	30	47	93.75%	63.83%	75.95%

Cluster 0 shows the extraction of dates. The results show that all dates in the page were extracted correctly leading to a 100% recall. However, one false positive was also reported decreasing the precision. Venues were extracted in cluster 1. Out of all venues reported in the page, the system was able to extract 35 leading to a recall of 83.33%. Cluster 1 also included 14 incorrectly clustered tokens that could not be refined by the system. Cluster 3 contained mostly telephone numbers. Although the system was able to correctly identify all actual telephone numbers, it was unable to extract values referring to telephone numbers such as"N/A". Out of 36 tokens reported under the telephone number column, only 27 were extracted and were reported as telephone numbers leading to 100% precision for cluster 3. Finally, cluster 5 reported marathon names. Out of 47, only 30 were extracted in addition to two false positives. Averaging results for all tokens extracted, the cumulative results are $P = 90.78\%$, $R = 84.71\%$, and $F1 = 85.99\%$.

From this example, it can be seen that it is necessary for tokens belonging to different categories to have different formats. Many marathon names were not completely reported (for example, the word "Boston" was substituted for the complete name "Boston Marathon") and ClusTex was unable to differentiate it from other tokens. In comparison, all dates had a similar format and were extracted correctly.

Rebuilding marathon entities using PAT, 32 output tuples were extracted. There were a total of 47 possible entities in the page. There were eight false positives. For the re-constructed entities, the following values were calculated: $P = 81.25\%$, $R = 55.32\%$, and $F1 = 65.82\%$. These values signify that out of all the output tuples, 81.25% were correct. Furthermore, owing to the similarly formatted tokens from different categories, the system was able to extract only a little more than half of all possible listings in the page.

5.4.7 US Marathon Listings from MarathonGuide.com

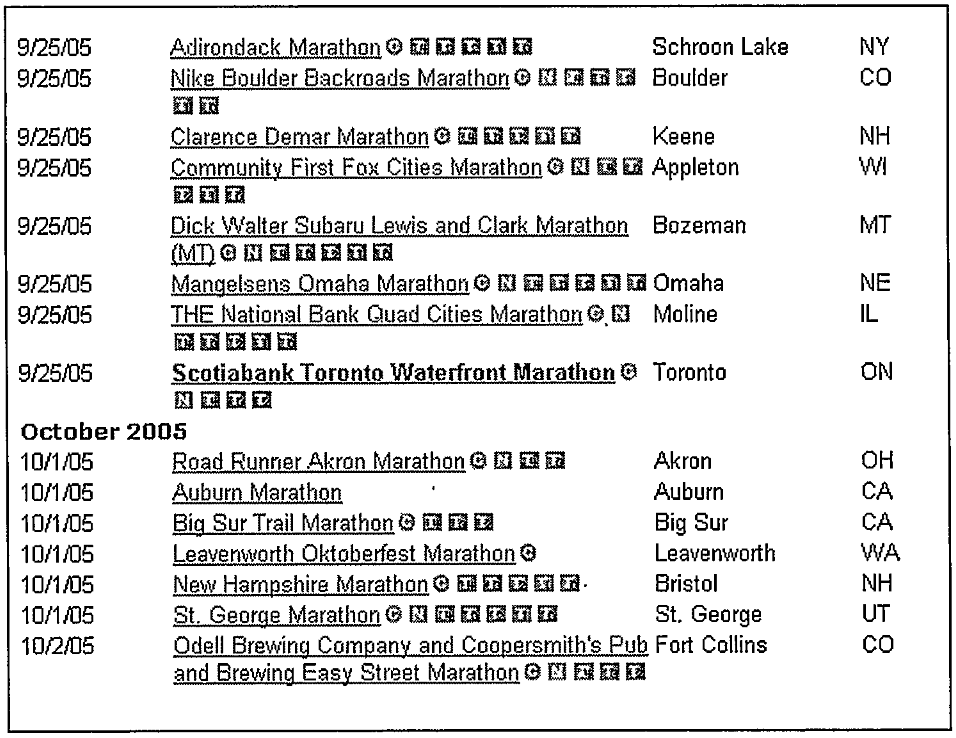
This page from MarathonGuide.com [31] contains marathon listings containing the date of the race, marathon name, the city where it is run and finally the state. This information is presented in an HTML table, and is shown in Figure 5.13.

After the clustering was performed by AutoClass, the results showed the existence of seven clusters. Table 5.19 shows the number of tokens contained in each cluster.

Table 5.19: MarathonGuide.com Listings: Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	132
1	115
2	76
3	66
4	42
5	36
6	3

Figure 5.14 shows the patterns of size 4 found in the output when the token text was replaced by the number of the cluster in which the token occurred. As in the previous experiments, all attempts to find patterns of a size larger than 4 led to all patterns being invalid, i.e., containing more than one occurrence of a cluster number.



9/25/05	Adirondack Marathon	Schroon Lake	NY
9/25/05	Nike Boulder Backroads Marathon	Boulder	CO
9/25/05	Clarence Demar Marathon	Keene	NH
9/25/05	Community First Fox Cities Marathon	Appleton	WI
9/25/05	Dick Walter Subaru Lewis and Clark Marathon (MT)	Bozeman	MT
9/25/05	Mangelsens Omaha Marathon	Omaha	NE
9/25/05	THE National Bank Quad Cities Marathon	Moline	IL
9/25/05	**Scotiabank Toronto Waterfront Marathon**	Toronto	ON
October 2005			
10/1/05	Road Runner Akron Marathon	Akron	OH
10/1/05	Auburn Marathon	Auburn	CA
10/1/05	Big Sur Trail Marathon	Big Sur	CA
10/1/05	Leavenworth Oktoberfest Marathon	Leavenworth	WA
10/1/05	New Hampshire Marathon	Bristol	NH
10/1/05	St. George Marathon	St. George	UT
10/2/05	Odell Brewing Company and Coopersmith's Pub and Brewing Easy Street Marathon	Fort Collins	CO

Figure 5.13: Web Page showing Marathon Listings from MarathonGuide.com

```
We have 29 patterns of size 4 now
0: 0 5 1 2 has occurred 3 times with first occurrence at location: 11
1: 5 1 2 3 has occurred 2 times with first occurrence at location: 12
2: 1 2 3 0 has occurred 42 times with first occurrence at location: 13
3: 2 3 0 1 has occurred 40 times with first occurrence at location: 14
4: 3 0 1 2 has occurred 37 times with first occurrence at location: 15
5: 0 1 2 3 has occurred 40 times with first occurrence at location: 16
6: 3 0 1 4 has occurred 26 times with first occurrence at location: 35
7: 0 1 4 5 has occurred 9 times with first occurrence at location: 36
8: 1 4 5 0 has occurred 9 times with first occurrence at location: 37
9: 4 5 0 1 has occurred 9 times with first occurrence at location: 38
10: 5 0 1 2 has occurred 23 times with first occurrence at location: 39
11: 0 1 2 5 has occurred 23 times with first occurrence at location: 48
12: 1 2 5 0 has occurred 23 times with first occurrence at location: 49
13: 2 5 0 1 has occurred 22 times with first occurrence at location: 50
14: 5 0 1 4 has occurred 9 times with first occurrence at location: 51
15: 0 1 4 3 has occurred 23 times with first occurrence at location: 52
16: 1 4 3 0 has occurred 23 times with first occurrence at location: 53
17: 4 3 0 1 has occurred 23 times with first occurrence at location: 54
18: 4 0 1 2 has occurred 2 times with first occurrence at location: 63
19: 2 3 0 5 has occurred 2 times with first occurrence at location: 170
20: 3 0 5 1 has occurred 2 times with first occurrence at location: 171
21: 0 1 2 4 has occurred 1 times with first occurrence at location: 413
22: 1 2 4 0 has occurred 1 times with first occurrence at location: 414
23: 2 4 0 1 has occurred 1 times with first occurrence at location: 415
24: 4 0 1 3 has occurred 1 times with first occurrence at location: 416
25: 0 1 3 5 has occurred 1 times with first occurrence at location: 417
26: 1 3 5 0 has occurred 1 times with first occurrence at location: 418
27: 3 5 0 1 has occurred 1 times with first occurrence at location: 419
28: 5 1 2 0 has occurred 1 times with first occurrence at location: 450
There were 25 invalid patterns also
```

Figure 5.14: Patterns discovered in the MarathonGuide.com Input using Cluster numbers

Pattern 2, "1 2 3 0" occurs most prolifically and is assumed to be *PAT*. All those patterns that differ from PAT by up to $log_2 4$ or two digits or any invalid patterns are re-classified to match it. After refinement, the token distribution through the clusters is shown in Table 5.20.

Table 5.20: MarathonGuide.com Listings: Post-Refinement Cluster-wise Distribution of Tokens

Cluster	Number of tokens
0	119
1	115
2	115
3	115
4	0
5	3
6	3

Since clusters 4, 5 and 6 do not figure in PAT, they were discarded.

Results

Looking at the cluster contents, we can see that cluster 0 contains mostly states, cluster 1 contains dates, cluster 2 contains marathon names and cluster 3 contains names of cities. Referring back to the web page, the sequence of tokens in PAT was confirmed as "1 2 3 0" since the information was laid out with the date followed by marathon name, the city and state. Table 5.21 shows the results for each refined cluster separately.

Cluster 0 shows extraction of states. The results show that 97.85% of all extracted tokens in cluster 0 were correct. The 100% recall shows that these correctly extracted tokens were all the possible tokens that could be extracted. Cluster 1 shows the extraction of dates. According to the results, all names reported in the data set

Table 5.21: MarathonGuide.com Listings: Cluster-wise Results

Cluster#	Info Type	TE	CE	TC	P	R	F1
0	State	119	114	114	95.80%	100%	97.85%
1	Date	115	114	114	99.13%	100%	99.56%
2	Marathon Names	115	114	114	99.13%	100%	99.56%
3	City	115	114	114	99.13%	100%	99.56%

were correctly extracted. These formed 99.13% of the total tokens reported for this cluster. Only one false positive was reported out of 115 tokens. Cluster 2 shows the extraction of marathon names. Once again, all names reported in the data set were extracted along with one false positive which meant that 99.13% of the output for this cluster was correct. Cluster 3 was found to contain cities and shows the same result as cluster 1 and 2. Averaging results for all tokens extracted, the cumulative results are $P = 98.298\%$, $R = 100\%$, and $F1 = 99.14\%$.

Rebuilding marathon entities using PAT, 115 output tuples were extracted. There were a total of 114 possible marathon entities in the page. There was one false positive. For the re-constructed marathon entities, the following values were calculated: $P = 99.13\%$, $R = 100\%$, and $F1 = 99.56\%$. These values signify that out of all the output tuples, 99.13% were correct. Furthermore, the system was able to extract all possible correct tuples from the data set.

5.5 Discussion

In this section, the results are discussed and compared with the results of other approaches. Table 5.22 shows the overall results from all web pages tested.

Thus, in summary, from the University of Calgary web page, all correct tu-

Table 5.22: Overall Extraction Results from all Web Pages

Web Page	P	R	F1
University of Calgary	96.92%	100%	98.43%
New York University	100%	89.71%	94.57%
Duke University	80.77%	91.3%	85.72%
Columbia University	91.94%	89.06%	90.48%
Buy.com Cell Phones	94.55%	100%	97.19%
1999 Marathon Schedule	81.25%	55.32%	65.82%
MarathonGuide.com US Listing	99.13%	100%	99.56%

ples were extracted with 3.08% of the output tuples being false positives. From the New York University web page, 89.71% of all correct tuples were extracted, all output being correct. From the Duke University web page, 91.33% correct tuples were extracted while 19.33% of total output tuples were false positives. In the case of Columbia University web page, 89.06% of all correct tuples were extracted and 91.94% of all extracted tuples were correct. In the cell phone domain, from the Buy.com page, all correct tuples in that page were extracted. The output also included 5.45% false positives. In the marathon domain, from the listing in the Running Page, only about 55.32% of all marathon tuples were extracted and the output contained 18.75% false positives as well. From the MarathonGuide.com page, all marathon tuples were correctly extracted in addition to 0.77% false positives.

This brings us to the discussion of how the page format would lead to a particular result. It appears that the best clusters are formed when the tokens belonging to a particular category follow the same format stringently. For example, in the University of Calgary web page, we see that the tokens belonging to each important category follow the same format, hence leading to a good clustering with very little or no need for refinement. On the other hand, as can be seen in the 1999 Marathon listing, similar formats lead to an incorrect clustering which cannot be fixed by refinement.

If the tokens of one kind differ from each other in format, then this would lead to an incorrect clustering of some tokens, and refinement needs to take place. Refinement is helped by tokens occurring in patterns. Thus, in the example of the Buy.com web page, we see that some prices were initially clustered incorrectly (because of failure in the format condition), but since all tokens follow a strict format, the refinement phase reassigned these tokens to the right cluster.

It should also be mentioned here that refinement works only for inaccurately clustered tokens in tuples with less than log_2k missing values. If there are any more missing values, then the system is unable to judge this as a pattern to be refined to match PAT. We see examples of this in the Columbia University and New York University web pages. Some tokens were incorrectly clustered, but since the tuple in which they occurred contained less than $k - log_2k$ values, refinement was not attempted.

The Duke University example brings to our attention the fact that extraction would fail if too many values are missing because the system would be unable to judge the correct pattern. In this example, out of a total of 93 course entities, only 8 contained course descriptions. Thus, there were 85 missing values in only one cluster. The system, due to the high number of missing values, was unable to detect the presence of a third attribute in the tuple.

ClusTex has a different premise than other existing approaches and works on semi-structured data rather than text. Hence, it has been tested on domains which are different than the ones used for testing by researchers who proposed classification based approaches. However, Embley *et al.* [13] tested their approach on the cell phone domain as well. For the cell phone sales page from Buy.com, they report $P = 85.4\%$, $R = 90.7\%$, and $F1 = 87.97\%$. Comparing this to my results of $P = 94.55\%$, $R = 100\%$, and $F1 = 97.19\%$, it is evident that my approach works on this data set quite well, and in any case, better than Embley's approach. On the other hand, Embley *et al.* test their approach on the car sales domain as well. They are able to extract information from a table that contains data on only one car. ClusTex fails in such a situation. Because it is based on clustering, it requires a large data set of tokens with different formats to be able to separate tokens from different categories.

Similarly, the works in which IE is seen as a text classification problem, usually test their approach on data sets such as the CMU seminar announcements corpus. This corpus contains a large number of documents, each containing one announcement. If ClusTex were to be applied on such a corpus, it would fail because it is unable to extract information about one entity from a page. However, one could imagine combining all documents into one large document. In that case, ClusTex might have some chance of success. This has not been tried, and is a possible future research direction.

ClusTex holds an edge over text classification systems in that they classify tokens from one page that contains one entity. They do not have to relate tokens belonging to the same entity with each other if there are other entities present as well. However, ClusTex achieves this by using patterns and shows good results.

Thus, judging from the evaluation criteria cited in Table 5.22, we can see that ClusTex shows very high precision and recall, and consequently F1, in the case of most listings tested. The lowest values are from tests on the 1999 marathon listings and Duke University listings, and pin-point the limitations of the approach that should be addressed rather than hidden.

Chapter 6

Conclusions and Future Work

In this thesis, ClusTex system which uses clustering techniques for IE from HTML web pages is developed. This chapter concludes this discussion by first presenting its advantages over and differences from other approaches. In the second section, the current limitations of this system are discussed and ideas for future work are explored.

6.1 Advantages of ClusTex and Differences from Existing Approaches

The most important point on which this approach differs from the others is that it uses clustering for data extraction, which is an unsupervised learning technique and does not require feedback from the user after domain features have been accounted for. All other similar works view IE as a text classification problems, and hence, use training examples and user feedback during the extraction process.

An advantage of this approach, at least when it is used on simple data sets with tokens of varying formats, is that it has a much simpler feature set than those of other systems. It only has HTML, semantic and orthographic features combined to better represent a particular domain. Thus, it requires very little pre-processing as compared to other systems but still gives comparable results.

If we decide to use ClusTex on an unknown domain, then only the HTML and orthographic features can be used to extract information, i.e., without any domain knowledge. In that case, after the information has been extracted, the user can match a cluster as a whole to its destination field in a database.

Another advantage of ClusTex is that instead of classifying tokens individually as in other systems, we can extract all tokens in one go.

6.2 Limitations and Possibilities for Future Work

The approach presented in this thesis suffers from several limitations that should be considered in future research. In the following paragraphs I suggest several directions in which further work can prove beneficial.

First of all, ClusTex requires that the web page be well-formed, i.e., have complete sets of start and end tags, and proper nestings etc. However, hand-coded pages with mistakes such as missing tags, especially in legacy data, are very common. On the bright side, this is easy to remedy with a utility such as HTML Tidy which can correct these mistakes and bring others to the attention of the user that it is unable to fix by itself.

Another limitation of this approach is its reliability on patterns of tokens for discovery of extraction rules. This effectively limits the use of this system for semi-structured data as opposed to semi-structured text in which tokens may occur without any pattern. It would be interesting to see an alternate way to discover extraction rules that does not use patterns, and consequently, can extract from semi-structured text as well.

Due to the very nature of clustering, ClusTex works only on data in which different tokens making up the data instances are different from each other, but the value for any particular attribute always occurs in more or less the same format. However, in some cases, the simplistic attributes may fail to account for finer details or properties of tokens. In a future work, this could be dealt with by adding more kinds of attributes in the tokenization process, using a POS tagger and a gazetteer. Right now, this system has a simplistic feature set which is potent enough to deal with many domains, especially ones containing semi-structured data. Making the attribute set more complex will also allow the system to work on semi-structured texts.

In this approach, during the refinement phase, tokens are only reassigned to other clusters in the pattern in which they occur differs from PAT by log_2k digits. If the pattern differs from PAT more than this value, reassignment is not attempted. If there is a valid pattern with more than log_2k values clustered incorrectly, it would not stand a chance of being rectified. A future work could deal with this problem possibly by striking a balance between a realistic limit for allowing re-assignment and getting better clustering results, which would decrease the need for refinement in the first place.

Another major limitation of ClusTex is its inability to deal with tokens that are longer that 200 characters in length. This is actually a limitation of AutoClass, the software I use for clustering. To get around this challenge, future research could concentrate on either using another clustering software or implementing a specialized clustering component for this system.

As we have seen in the testing phase, a case may arise when the system is unable to estimate an extraction rule based on heuristics and has to choose the first most prolific pattern for the rule. It would be interesting to see how a user feedback component could be added to get around this problem.

Existing systems based on text classification take the input pages one by one and classify the tokens into classes. This system requires all input to be on one page. In the future, one could attempt to make the system workable on many documents, each containing one instance, as suggested by Masterson and Kushmerick [32].

In its current state, the user has to supply all web pages individually to the system for extraction. An integration of this system with an information retrieval component resulting in an information integration system should be considered at a later stage. This would make the entire process truly automatic.

In conclusion, this thesis reports the creation of a new and efficient approach for IE using probabilistic clustering which should prove beneficial for the purposes of IE.

Bibliography

[1] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[2] D. Buttler, L. Liu, and C. Pu. A fully automated object extraction system for the world wide web. In *Proceedings of the 2001 International Conference on Distrubuted Computing Systems (ICDCS'01)*, pages 361–370, Phoenix, Arizona, May 2001.

[3] Buy.com. Cell phones and service plans. `http://www.buy.com/retail/wireless/product.asp?sku=&loc=12435&a=&z=90066&s=22&p=1&w=1`, 2005.

[4] Michael Cafarella, Doug Downey, Stephen Soderland, and Oren Etzioni. Knowitnow: Fast, scalable information extraction from the web. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2005.

[5] Peter Cheeseman and John Stutz. Bayesian classification (autoclass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180. 1996.

[6] Fabio Ciravegna. Learning to tag for information extraction from text. *In Workshop Machine Learning for Information Extraction, European Conference on Artifical Intelligence ECCAI*, August 2000.

[7] Fabio Ciravegna, Alexiei Dingli, David Guthrie, and Yorick Wilks. Integrating information to bootstrap information extraction from web sites. In *Proceedings*

of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico, 2003.

[8] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.

[9] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, and Paolo Missier. An automatic data grabber for large web sites. In *VLDB*, pages 1321–1324, 2004.

[10] Line Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norweigan Computing Center, 1999.

[11] Tina Eliassi-Rad and Jude Shavlik. *Intelligent exploration of the web*, chapter Intelligent Web agents that learn to retrieve and extract information, pages 255–274. Physica-Verlag GmbH, Heidelberg, Germany, Germany, 2003.

[12] David W. Embley, Douglas M. Campbell, Y. S. Jiang, Stephen W. Liddle, Yiu-Kai Ng, Dallan Quass, and Randy D. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data Knowledge Engineering*, 31(3):227–251, 1999.

[13] David W. Embley, Cui Tao, and Stephen W. Liddle. Automating the extraction of data from html tables with unknown structure. *Data & Knowledge Engineering*, 54(1):3–28, 2005.

[14] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel Weld, and Alexander Yates.

Web-scale information extraction in knowitall (preliminary results). In *Proceedings International WWW Conference*, 2004.

[15] Aidan Finn and Nicholas Kushmerick. Multi-level boundary classification for information extraction. In *Proceedings of the European Conference on Machine Learning, Pisa*, 2004.

[16] Dayne Freitag. Information extraction from HTML: Application of a general machine learning approach. In *Proceedings of the Fifteenth Conference on Artificial Intelligence AAAI-98*, pages 517–523, 1998.

[17] O. Glickman and R. Jones. Examining machine learning for adaptable end-to-end information extraction systems. *In Workshop on Machine Learning for Information Extraction, AAAI*, 1999.

[18] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 902–903, New York, NY, USA, 2005. ACM Press.

[19] Robin Hanson, John Stutz, and Peter Cheeseman. Bayesian classification theory. Technical Report FIA90 -12-7-01, NASA, 1991.

[20] Theodore W. Hong and Keith L. Clark. Using grammatical inference to automate information extraction from the Web. *Lecture Notes in Computer Science*, 2168, 2001.

[21] Yunhua Hu, Guomao Xin, Ruihua Song, Guoping Hu, Shuming Shi, Yunbo

Cao, and Hang Li. Title extraction from bodies of html documents and its application to web page retrieval. In *ACM-SIGIR'05*, 2005.

[22] A.K. Jain, M.N. Murty, and P.J. FLYNN. Data clustering: A review. *ACM Computing Surveys*, 31:264–323, 1999.

[23] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, 1988.

[24] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper induction for information extraction, 1997.

[25] Nicholas Kushmerick. Gleaning the web. *IEEE Intelligent Systems*, 14(2):20–22, 1999.

[26] Martin Labsky and Vojtech Svatek. Information extraction from web product catalogues. Working Paper, 2004.

[27] Martin Labsky, Vojtech Svatek, Pavel Praks, and Ondrej Svab. Information extraction from html product catalogues: Coupling quantitative and knowledge-based approaches. In *Dagstuhl Seminar on Machine Learning for the Semantic Web*, 2005.

[28] A. Laender, B. Ribeiro-Neto, A. Silva, and J. Teixeira. A brief survey of web data extraction tools. In *SIGMOD Record*, volume 31, June 2002.

[29] Ling Liu, Wei Han, David Buttler, Calton Pu, and Wei Tang. An xml-based wrapper generator for web information extraction. In *Proceedings ACM SIGMOD International Conference on Management of Data*, 1999.

[30] Yimin Liu. Multi-objective genetic algorithms based approach to clustering and its application to microarray data analysis. Master's thesis, University of Calgary, 2004.

[31] MarathonGuide.com. Us marathons races directory and schedule. `http://www.marathonguide.com/races/races.cfm`, 2005.

[32] David Masterson and Nicholas Kushmerick. Information extraction from multi-document threads. In *Proceedings Workshop on Adaptive Text Extraction and Mining*, pages 34–41, 2003.

[33] University of Calgary. Computer science and software engineering courses. `http://www.cpsc.ucalgary.ca/Undergrads/Courses`, 2005.

[34] The Running Page. 1999 marathon schedule. `http://www.runningpage.com/races/marathon.htm`, 2000.

[35] Leonid Peshkin and Avi Pfefer. Bayesian information extraction network. In *Proceedings of the Eighteenth International Joint Conf. on Artificial Intelligence*, 2003.

[36] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

[37] R. H. Turi. *Clustering-Based Colour Image Segmentation.* PhD thesis, Monash University, Australia, 2001.

[38] Columbia University. Computer science at columbia university - fall 2004 courses. ⟨`http://www.cs.columbia.edu/education/courses/list?`

yearterm=20043⟩, 2005.

[39] Duke University. Computer science courses. http://cs.duke.edu/students/courses, 2005.

[40] New York University. Computer science course list. http://cs.nyu.edu/web/Academic/courses, 2005.