A REFERENCE-SET APPROACH TO INFORMATION EXTRACTION FROM UNSTRUCTURED, UNGRAMMATICAL DATA SOURCES

by

Matthew Michelson

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

May 2009

Dedication

To Sara, whose patience and support

never waiver.

Acknowledgments

First and foremost, thanks to Sara, my wife. Her love, patience, and support made this thesis possible.

Of course, many thanks to my family for their patience. I know it took almost 90% of my life so far to finish school, but I promise this is my last degree! (Unless...)

I would especially like to thank my thesis adviser, Craig Knoblock, who taught me the mechanics of research, the elements of style and grammar, and the patience required to find the right solution. Craig was (and still is) always free to talk to me when I needed it, especially when I felt like I was floundering in the muck of trial-and-error (usually more error, less trial). Most importantly, he gave me enough freedom to try my own solutions, but also helped reign my ideas back to practicality. I am grateful for all his help.

Many thanks to the rest of my thesis committee: Kevin Knight, Cyrus Shahabi and Daniel O'Leary. My committee helped me focus my thesis during my qualification exam with their comments and questions. Their patience and kind demeanor made the practical aspects of my defense, such as scheduling, painless!

To my ISI-mates, both past (Marty, Snehal, Pipe, Mark, and Wesley) and present (Yao-yi, Anon, and Amon), thanks for all the laughs, games, Web-based time wasters,

and general harassment, without which the life of a graduate student would feel more like "real work!"

This research is based upon work supported in part by the National Science Foundation under award number IIS-0324955; in part by the Air Force Office of Scientific Research under grant number FA9550-07-1-0416; and in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004.

The United States Government is authorized to reproduce and distribute reports for Governmental purposes not withstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

Table of Contents

List Of Tables

List Of Figures

Abstract

This thesis investigates information extraction from unstructured, ungrammatical text on the Web such as classified ads, auction listings, and forum postings. Since the data is unstructured and ungrammatical, this information extraction precludes the use of rule-based methods that rely on consistent structures within the text or natural language processing techniques that rely on grammar. Instead, I describe extraction using a "reference set," which I define as a collection of known entities and their attributes. A reference set can be constructed from structured sources, such as databases, or scraped from semi-structured sources such as collections of Web pages. In some cases, as I shown in this thesis, a reference set can even be constructed automatically from the unstructured, ungrammatical text itself. This thesis presents methods to exploit reference sets for extraction using both automatic techniques and machine learning techniques. The automatic technique provides a scalable and accurate approach to extraction from unstructured, ungrammatical text. The machine learning approach provides even higher accuracy extractions and deals with ambiguous extractions, although at the cost of requiring human effort to label training data. The results demonstrate that reference-set based extraction outperforms the current state-of-the-art systems that rely on structural or grammatical clues, which is not appropriate for unstructured, ungrammatical text. Even the fully automatic case,

which constructs its own reference set for automatic extraction, is competitive with the current state-of-the-art techniques that require labeled data. Reference-set based extraction from unstructured, ungrammatical text allows for a whole category of sources to be queried, allowing for their inclusion in data integration systems that were previously limited to structured and semi-structured sources.

Chapter 1

Introduction

This chapter introduces the issue of information extraction from unstructured, ungrammatical text on the Web. In this chapter, I first motivate and define the problem explicitly. Then, I briefly outline my approach to the problem and describe the general framework for reference-set based information-extraction. I then describe my contributions to the field of information extraction, and then present the outline of this thesis.

1.1 Motivation and Problem Statement

As more and more information comes online, the ability to process and understand this information becomes more and more crucial. Data integration attacks this problem by letting users query heterogeneous data sources within a unified query framework, combining the results to ease understanding. However, while data integration can integrate data from structured sources such as databases, semi-structured sources such as that extracted from Web pages, and even Web Services [58], this leaves out a large class of useful information: unstructured and ungrammatical data sources. I call such unstructured, ungrammatical data "posts." Posts range in source from classified ads to auction

listings to forum postings to blog titles to paper references. **The goal of this thesis is to structure sources of posts, such that they can be queried and included in data integration systems.** Figure 1.1 shows an example integration system that combines a database of car safety information, semi-structured Web pages of car reviews, and posts of car classifieds. The box labeled "THESIS" is the mechanism by which to tie the posts of classified ads into the integration system via structured queries, and this is the motivation behind my research.

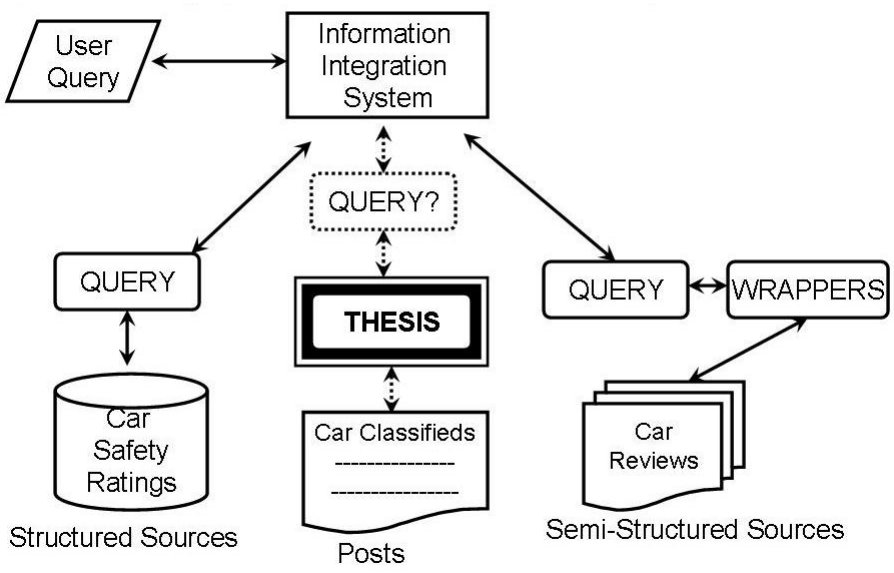

Figure 1.1: Fitting posts into information integration

Posts are full of useful information, as defined by the attributes that compose the entity within the post. For example, consider the posts about cars from the online classified service Craigslist,[1] shown in Table 1.1. Each used car for sale is composed of attributes that define this car; and if we could access the individual attributes we could include such sources in data integration systems, such as that of Figure 1.1, and answer

[1] www.craigslist.org

interesting queries such as "return the classified ads and car reviews for used cars that are made by Honda and have a 5 star crash rating." Such a query might requires combining the structured database of safety ratings with the posts of the classified ads and the car review websites. Clearly, there is a case to be made for the utility of structured queries such as joins and aggregate selections of posts.

Craig's List Post
93 civic 5speed runs great obo (ri) $1800
93- 4dr Honda Civc LX Stick Shift $1800
94 DEL SOL Si Vtec (Glendale) $3000

Table 1.1: Three posts for Honda Civics from Craig's List

However, currently accessing the data within posts does not go much beyond keyword search. This is precisely because the ungrammatical, unstructured nature of posts makes extraction difficult, so the attributes remain embedded within the posts. These data sources are ungrammatical, since they do not conform to the proper rules of written language (English in this case). Therefore, Natural Language Processing (NLP) based information extraction techniques are not appropriate [57; 9]. Further, the posts are unstructured since the structure can vary vastly between each listing. (That is, we can not assume that the ordering of the attributes across the posts are preserved.) So, wrapper based extraction techniques will not work either [49; 19] since the assumption that the structure can be exploited does not hold. Lastly, even if one can extract the data from within posts, you would need to assure that the extracted values map to the same value for accurate querying. Table 1.1 shows three example posts about Honda Civics, but each refers to the model differently. The first post mentions a "civic," the next writes "Civc" which is a spelling mistake, and the last calls the model "DEL SOL." Yet, if we do a

query for all "civic" cars, all of these posts should be returned. Therefore, not only is it important to extract the values from the posts, but we should clean the values in that all values for a given attribute should be transformed to a single, uniform value. This thesis presents an extraction methodology for posts that handles both the difficulties imposed by the lack of grammar and structure, and also can assign a single, uniform value for an attribute (called semantic annotation).

1.2 Proposed Approach

As stated, the method of extraction presented in this thesis involves exploiting structured, relational data to aid the extraction, rather than relying on NLP or structure based methods. I call this structured data a "reference set." A reference set consists of collections of known entities with the associated, common attributes. Examples of reference sets include online (or offline) reference documents, such as the CIA World Fact Book;[2] online (or offline) databases, such as the Comics Price Guide;[3] and with the Semantic Web one can envision building reference sets from the numerous ontologies that already exist.

These are all cases of manually constructed reference sets since they require user effort to construct the database, scrape the data off of Web pages, or create the ontology. As this thesis shows, however, in many cases we can automatically generate a reference set from the posts themselves, which can then be used for extraction. Nonetheless, manual reference sets provide a stronger intuition into the strengths of reference-set based information extraction.

[2] http://www.cia.gov/cia/publications/factbook/
[3] www.comicspriceguide.com

Using the car example, a manual reference set might be the Edmunds car buying guide,[4] which defines a schema for cars (the set of attributes such as make and trim) as well as standard values for attributes themselves. Manually constructing reference sets from Web sources such as the Edmunds car buying guide is done using wrapper technologies such as Agent Builder[5] to scrape data from the Web source using the schema the source defines for the car.

Regardless of whether the reference set is constructed manually or automatically, the high-level procedure for exploiting a reference set for extraction is the same. To use the reference sets, the methods in this thesis first find the best match between each post and the tuples of the reference set. By matching a post to a member of the reference set, the algorithm can use the reference set's schema and attribute values as semantic annotation for the post (a single, uniform value for that attribute). Further, and importantly, these reference set attributes provide clues for the information extraction step.

After the matching step, the methods perform information extraction to extract the actual values in the post that match the schema elements defined by the match from the reference set. This is the information extraction step. During the information extraction step, the parts of the post are extracted that best match the attribute values from the reference set member chosen during the matching step.

Figure 1.2 presents the basic methodology for reference-set based extraction, using the second post of Table 1.1 as an example, and assuming it matches the reference set tuple with the attributes {MAKE=HONDA, MODEL=CIVIC, TRIM=2 DR LX, YEAR=1993}. As shown, first, the algorithm selects the best match from a reference set.

[4] www.edmunds.com

[5] A product of Fetch Technologies http://www.fetch.com/products.asp

Next, it exploits the match to aid information extraction. While this basic framework stays consistent across the different methods described in this thesis, the details vary in accordance with the different algorithms.

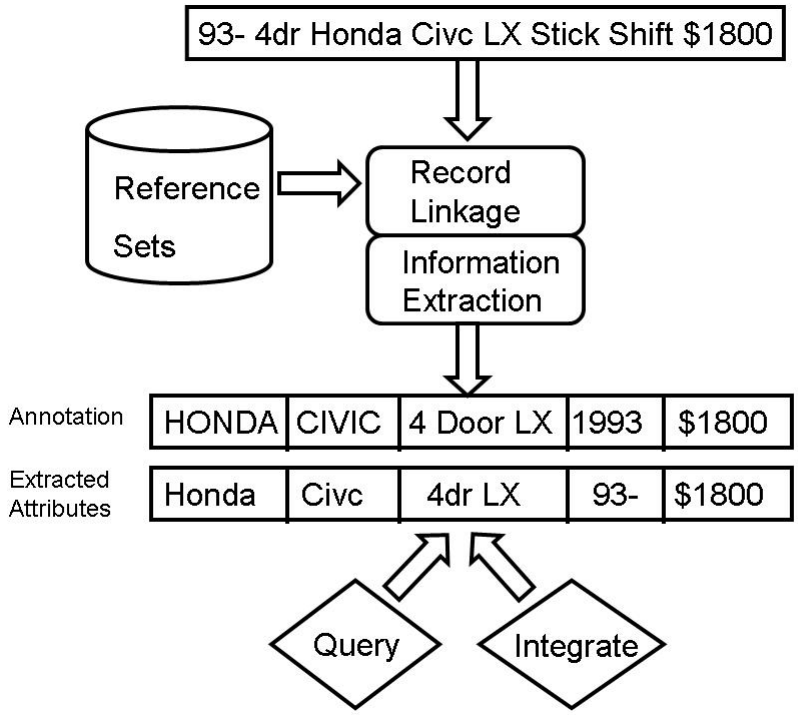

Figure 1.2: Framework for reference-set based extraction and annotation

This thesis presents three approaches to reference-set based extraction that are each suitable depending on certain circumstances. The main algorithm of this thesis, presented in Chapter 2 automatically matches a post to a reference set, and then uses the results of this automatic matching to aid automatic extraction. This method assumes that there exists a repository of manually created reference sets from which the system itself can choose the appropriate one(s) to for matching/extraction. By using a repository, this may lessen the burden on users who do not need to create new reference sets each time, but can reuse previously created ones.

However, requiring a manually created reference set is not always even necessary. As I show in Chapter 3, in many cases the algorithm I describe can construct the reference set automatically, directly from the posts. It can then use the automatically constructed reference set using the automatic matching and extraction techniques of Chapter 2.

There are two cases where a manually constructed reference set is preferred over one that is automatically constructed. First, it's possible that a user requires a certain attribute to be added as a semantic annotation to a post in support of future, structured queries (that is, after extraction). For example, suppose an auto-parts company has an internal database of parts for cars. This company might want to join their parts database with the classified ads about cars in a effort to advertise their mechanics service. Therefore, they could use reference-set based extraction to identify the cars for sale in the classified ads, which they can then join with their internal database. However, each car in their database might have its own internal identification and this attribute might be needed for their query. In this case, since none of the classified ads would already contain this company-specific attribute, the company would have to manually create their own reference set and include this attribute. So, in this case, the user would manually construct a reference set and then use it for extraction.

Second, the textual characteristics of the posts might make it difficult to automatically create a high quality reference set using the technique of Chapter 3. In fact, in Chapter 3 I present an analysis of the characteristics that define these domains where automatically creating a reference set is appropriate. Further, that chapter describes a method that can easily decide whether to use the automatic approach of Chapter 3, or suggest to the user to manually construct a reference set.

The automatic extraction approach can be improved using machine learning techniques at the cost of labeling data, as I describe in Chapter 4. This is especially useful when very high quality extractions are required. Further, a user may require high-accuracy extraction of attributes that are not easily represented in a reference set, such as prices or dates. I call these "common attributes." In particular, common attributes are usually a very large set (such as prices or dates), but they contain characteristics that can be exploited for extraction. For instance, regular expressions can identify prices.

However, when extracting both reference-set and common attributes there are ambiguities that must be accounted for to extract common attributes accurately. For instance, while a regular expression for prices might pull out the token "626" from a classified ad for cars, it is more likely that the "626" in the post refers to the car model, since the Mazda company makes a car called the 626. By using a reference set to aid disambiguation, the system handles extraction from posts with high accuracy. For instance, the system can be trained that when a token matches both a regular expression for prices and the model attribute from the reference set, it should be labeled as a car model. Chapter 4 describes my machine learning approach to matching and extraction.

Table 1.2 summarizes and compares all three methods.

Table 1.2: Methods for extraction/annotation under various assumptions

	Summary	**Situation**
Chapter 2 ("ARX" approach)	Requires repository of manually constructed reference sets; Automatically selects appropriate reference set(s); Automatic extraction	Cannot use automatically constructed reference set because require specific, unattainable attribute or characteristics of text prevent automatic construction
Chapter 3 ("ILA" algorithm)	Automatically construct reference set; Use constructed reference set with automatic method from Chapter 2	Data set conforms to characteristics for constructing reference set, which can be tested
Chapter 4 ("Phoebus" approach)	Supervised machine learning; Requires manually constructed reference set and labeled training data	Require high-accuracy extraction including common attributes that may be ambiguous

1.3 Thesis Statement

I demonstrate that we can exploit reference sets for information extraction on unstructured, ungrammatical text, which overcomes the difficulties encountered by current extraction methods that rely on grammatical and structural clues in the text for extraction; furthermore, I show that we can automatically construct reference sets from the unstructured, ungrammatical text itself.

1.4 Contributions of the Research

The key contribution of the thesis is a *method for information extraction that exploits reference sets, rather than grammar or structure based techniques.* The research includes the following contributions:

- An automatic matching and extraction algorithm that exploits a given reference set (Chapter 2)
- A method that selects the appropriate reference sets from a repository and uses them for extraction and annotation without training data (Chapter 2)
- An automatic method for constructing reference sets from the posts themselves. This algorithm also is able to suggest the number of posts required to automatically construct the reference set sufficiently (Chapter 3)

- An automatic method whereby the machine can decide whether to construct its own reference set, or require a person to do it manually (Chapter 3)
- A method that uses supervised machine learning to perform high-accuracy extraction on posts, even in the face of ambiguity. (Chapter 4)

1.5 Outline of the thesis

The remainder of this thesis is organized as follows. Chapter 2 presents my automatic approach to matching posts to a reference set and then using those matches for automatic extraction. This technique requires no more manual effort beyond constructing a reference set (and even that can be avoided if the repository of reference sets is comprehensive enough). Chapter 3 describes the technique for automatically constructing a reference set, which can then be exploited using the methods of Chapter 2. Chapter 3 also describes an algorithm for deciding whether or not the reference set can be automatically constructed. Chapter 4 describes the approach to reference-set based extraction that uses machine learning. Chapter 5 reviews the work related to the research in this thesis. Chapter 6 finishes with a conclusion of the contributions of this thesis and points out future courses of research.

Chapter 2

An Automatic Approach to Extraction from Posts

Unsupervised Information Extraction (UIE) has recently witnessed significant progress [8; 28; 51]. However, current work on UIE relies on the redundancy of the extractions to learn patterns in order to make further extractions. Such patterns rely on the assumption that similar structures will occur again to make the learned extraction patterns useful. Initially, the approaches can be seeded with manual rules [8], example extractions [51], or sometimes nothing at all, relying solely on redundancy [28]. Regardless of how the extraction process starts, extractions are validated via redundancy, and they are then used to generalize patterns for extractions.

This approach to UIE differs from the one in this thesis in important ways. First, the use of patterns makes assumptions about the structure of the data. However, I cannot make any structural assumptions because my target data for extraction is defined by its lack of structure and grammar. Second, since these systems are not clued into what can be extracted, they rely on redundancy for their confidence in their extractions. In my case, the confidence in the extracted values comes from their similarity to the reference-set attributes exploited during extraction. Lastly, the goals differ. Previous UIE systems

seek to build knowledge bases through extraction. For instance, they aim to find all types of cars on the Web. Our extraction creates relational data, which allows us to classify and query posts. For this reason, we believe the previous work complements ours well because we could use their techniques to automatically build our reference sets.

This chapter presents my algorithm for unsupervised information extraction of unstructured, ungrammatical text. The algorithm has three distinct steps. As stated in the introduction, reference-set based extraction relies on exploiting relational data from reference sets, so the first step of the algorithm is to choose the applicable reference set from a repository of reference sets. This is done so that users can easily reuse reference sets they may have constructed in the past. Of course, a user can skip this step and just provide his or her reference set directly to the algorithm instead, but by having the machine select the appropriate reference sets, this eases some of the burden and guesswork from a human user. Once the reference sets are chosen, the second step in the algorithm is for the system to match each post to members of the reference set, which allows it to use these members' attributes as clues to aid in the extraction. Finally, the system exploits these reference-set attributes to perform the unsupervised extraction. It is important to note that the approach of this chapter is totally automated beyond the construction of the reference sets.

2.1 Automatically Choosing the Reference Sets

As stated above, the first step in the algorithm is to select the appropriate reference sets. Given that the repository of reference sets grows over time, the system should choose the

reference sets to exploit for a given set of posts. The algorithm chooses the reference sets based on the similarity between the set of posts and the reference sets in the repository. Intuitively, the most appropriate reference set is the one with the most useful tokens in common with the posts. For example, if we have a set of posts about cars, we expect a reference set with car makes, such as Honda or Toyota, to be more similar to the posts than a reference set of hotels.

To choose the reference sets, the system treats each reference set in the repository as a single document and the set of posts as a single document. This way, the algorithm calculates a similarity score between each reference set and the full set of posts. Next these similarity scores are sorted in descending order, and the system traverses this list, computing the percent difference between the current similarity score and the next. If this percent difference is above a threshold, and the score of the current reference set is greater than the average similarity score for all reference sets, the algorithm terminates. Upon termination, the algorithm returns as matches the current reference set and all reference sets that preceded it. If the algorithm traverses all of the reference sets without terminating, then no reference sets are relevant to the posts. Table 2.1 shows the algorithm.

I choose the percent difference as the splitting criterion between the relevant and irrelevant reference sets because it is a relative measure. Comparing only the actual similarity values might not capture how much better one reference set is as compared to another. Further, I require that the score at the splitting point be higher than the average score. This requirement captures cases where the scores are so small at the end of the list that their percent differences can suddenly increase, even with a small difference

Table 2.1: Automatically choosing a reference set

Given posts P, threshold T, and reference set repository R
$p \leftarrow$ *SingleDocument(P)*
For all reference sets $ref \in R$
  $r_i \leftarrow$ *SingleDocument(ref)*
  $SIM(r_i, p) \leftarrow$ *Similarity*(r_i, p)
For all $SIM(r_i, p)$ in descending order
  If *PercentDiff*$(SIM(r_i, p), SIM(r_{i+1}, p)) > T$ AND
  $SIM(r_i, p) > AVG(SIM(R, p))$
    Return $SIM(r_x, p), 1 > x > i$
Return nothing (No matching reference sets)

in score. This difference does not mean that the algorithm found relevant reference sets, rather it just means that the next reference set is that much worse than the current, bad one.

By treating each reference set as a single document, the algorithm of Table 2.1 scales linearly with the size of the repository. That is, each reference set in the repository is scored against the posts only once. Furthermore, as the number of reference sets increases, the percent difference still determines which reference sets are relevant. If an irrelevant reference set is added to the repository, it will score low, so it will still be relatively that much worse than the relevant one. If a new relevant set is added, the percent difference between the new one and the one already chosen will be small, but both of them will still be much better than the irrelevant sets in the repository. Thus, the percent difference remains a good splitting point.

I do not require a specific similarity measure for this algorithm. Instead, the experiments of Section 2.4 demonstrate that certain classes of similarity metrics perform

well. So, rather than picking just one as the best I try many different metrics and draw conclusions about what types of similarity scores should be used and why.

2.2 Matching Posts to the Reference Set

As outlined in Chapter 1, reference-set based extraction hinges upon exploiting the best matching member of a reference set for a given post. Therefore, after choosing the relevant reference sets, the algorithm matches each post to the best matching members of the reference set. When selecting multiple reference sets, the matching algorithm executes iteratively, matching the set of posts once to each chosen reference set. However, if two chosen reference sets have the same schema, it only selects the higher ranked one to prevent redundant matching.

To match the reference set records to the posts, I employ a vector-space model. By using a vector-space approach to matching, rather than machine learning, I can lessen the burden required by tasks such as labeling training matches. Furthermore, a vector-space model allows me to use information-retrieval techniques such as inverted indexes, which are fast and scalable.

In my model, I treat each post as a "query" and each record of the reference set as a "document," and I use token-based similarity metrics to define their likeness. Again, I do not tie this part of the algorithm to a specific similarity metric because I find a class of token-based similarity metrics works well, which I justify in the experiments.

However, for the matching algorithm I do modify the similarity metrics I use. This modification considers two tokens as matching if their Jaro-Winkler [62] similarity is

greater than a threshold. For example, consider the classic Dice similarity, defined over a post p and a record of the reference set r, as:

$$Dice(p, r) = \frac{2 * (p \cap r)}{|p| + |r|}$$

If the threshold is 0.95, two tokens are put into the intersection of the modified Dice similarity if those two tokens have a Jaro-Winkler similarity above 0.95. This Jaro-Winkler modification captures tokens that might be misspelled or abbreviated, which is common in posts. The underlying assumption of the Jaro-Winkler metric is that certain attributes are more likely similar if they share a certain prefix. This works particularly well for proper nouns, which many reference set attributes are, such as "Honda Accord" cars. Using the modified similarity metric, the algorithm compares each post, p_i, to each member of the reference set and returns the reference set record(s) with the maximum score, called r_{max_i}. In the experiments, I vary this threshold to test its effect on the performance of matching the posts. I also justify using the Jaro-Winkler metric to modify the Dice similarity, rather than another edit distance metric.

However, because more than one reference set record can have a maximum similarity score with a post (r_{max_i} is a set), an ambiguity problem exists with the attributes provided by the reference set records. For example, consider a post "Civic 2001 for sale, look!" If we have the following 3 matching reference records: {HONDA, CIVIC, 4 Dr LX, 2001}, {HONDA, CIVIC, 2 Dr LX, 2001} and {HONDA, CIVIC, EX, 2001}, then we have an ambiguity problem with the trim. We can confidently assign HONDA as the make, CIVIC as the model, and 2001 as the year, because all of the matching records agree on these

attributes. We say that these attributes are "in agreement." Yet, there is disagreement on the trim because we cannot determine which value is best for this attribute. All the reference records are equally acceptable from the vector-space perspective, but they differ in value for this attribute. Therefore, the algorithm removes all attributes that do not agree across all matching reference set records from the annotation (e.g., the trim in our example). Once this process executes, the system has all of the attributes from the reference set that it can use to aid extraction. The full automatic matching algorithm is shown in Table 2.2.

Table 2.2: Vector-space approach to finding attributes in agreement

Given posts P and reference set R
For all $p_i \in P$
  $r_{max_i} \leftarrow$ *MAX(SIM* (p_i,R))
  Remove attributes not *in agreement* from r_{max_i}

Selecting all the reference records with the maximum score, without pruning possible false positives, introduces noisy matches. These false positives occur because posts with small similarity scores still 'match' certain reference set records. For instance, the post "We pick up your used car" matches the Renault Le Car, Lincoln Town Car, and Isuzu Rodeo Pick-up, albeit with small similarity scores. However, since none of these attributes are in agreement, this post gets no annotation. Therefore, by using only the attributes in agreement, I essentially eliminate these false positive matches because no annotation will be returned for this post. That is, because no annotation is returned for such posts, it is as if there are no matching records for it. In this manner, by using only the attributes "in agreement," I separate the true matches from the false positives.

Once the system matches the posts to a reference set, users can query the posts structurally like a database. This is a tremendous advantage over the traditional keyword search approach to searching unstructured, ungrammatical text. For example, keyword search cannot return records for which an attribute is missing, whereas the approach in this thesis can. If a post were "2001 Accord for sale," and the keyword search was Honda, this record would not be returned. However, after matching posts to the reference set, if we select all records where the matched-reference-set attribute is "Honda" this record would be returned. Another benefit of matching the posts is that aggregate queries are possible. This mechanism is not supported by keyword search.

Perhaps one of the most useful aspects of matching posts to the reference set is that one can include the posts in an information-integration system (e.g., [38; 58]). In particular, my approach is well suited for Local-as-View (LAV) information integration systems [37], which allow for easy inclusion of new sources by defining their "source description" as a view over known domain relations. For example, assume a user has a reference set of cars from Edmunds car buying guide for the years 1990-2005 in his repository. If one includes this source in an LAV integration system, the source description is:

$$\text{Edmunds(make, model, year, trim) :-}$$
$$\text{Cars(make, model, year, trim)} \wedge$$
$$\text{year} \geq 1995 \wedge \text{year} \leq 2005$$

Now, assume a set of posts comes in and the system matches them to the records of this Edmunds source. Before matching, the user had no idea how to include this source in the integration system, but after matching the user can use the same source description

of the matching reference set, along with a variable to output the "post" attribute, to define a source description of the unstructured, ungrammatical source. This is possible because the algorithm appends records from the unstructured source with the attributes in agreement from the reference set matches. So, the new source description becomes:

$$\text{UnstructuredSource(post, make, model, year, trim) :-}$$

$$\text{Cars(make, model, year, trim)} \wedge$$

$$\text{year} \geq 1995 \wedge \text{year} \leq 2005$$

In this manner, one can collect and include new sources of unstructured, ungrammatical text in LAV information integration systems without human intervention.

2.3 Unsupervised Extraction

Yet, the annotation that results from the matching step may not be enough. There are many cases where a user wants to see the actual extracted values for a given attribute, rather than the reference set attribute from the matching record. Therefore, once the system retrieves the attributes in agreement from the reference set matches, it exploits these attributes for information extraction. First, for each post, each token is compared to each of the attributes in agreement from the matches. The token is labeled as the attribute for which it has the maximum Jaro-Winkler similarity. I label a token as 'junk' (to be ignored) if it receives a score of zero against all reference-set attributes.

However, this initial labeling generates noise because the attributes are labeled in isolation. To remedy this, the algorithm uses the modified Dice similarity, and generates a baseline score between the extracted field and the reference-set field. Then, it goes

Table 2.3: Cleaning an extracted attribute

CLEAN-ATTRIBUTE(extracted attribute E, reference set attribute A)
ExtCands $\leftarrow$ {}
baseline $\leftarrow$ *DICE(E,A)*
For all tokens $e_i \in E$
  score $\leftarrow$ *DICE*(E/e_i,*A*)
  IF *score* > *baseline*
    ExtCands $\leftarrow$ *ExtCands* $\cup$ e_i
IF *ExtCands* is empty
  return E
ELSE
  select $c_i \in$ *ExtCands* that yieds max *score*
  CLEAN-ATTRIBUTE(E/c_i,*A*)

through the extracted attribute, removing one token at a time, and calculates a new Dice similarity value. If this new score is higher than the baseline, the removed token is a candidate for permanent removal. Once all tokens are processed in this way, the candidate for removal that yielded the highest new score is removed. Then, the algorithm updates the baseline to the new score, and repeats the process. When none of the tokens yield improved scores when removed, this process terminates. This cleaning is shown in Table 2.3. For this part of the algorithm, I use the modified Dice since our experiments in the annotation section show this to be one of the best performing similarity metrics. Note, my unsupervised extraction is $O(\|p\|^2)$ per post, where $\|p\|$ is the number of tokens in the post, because, at worst, all tokens are initially labeled, and then each one is removed, one at a time. However, because most posts are relatively short, this is an acceptable running time.

The whole multi-pass procedure for unsupervised extraction is given by Figure 2.1. By exploiting reference sets, the algorithm performs unsupervised information extraction without assumptions regarding repeated structure in the data.

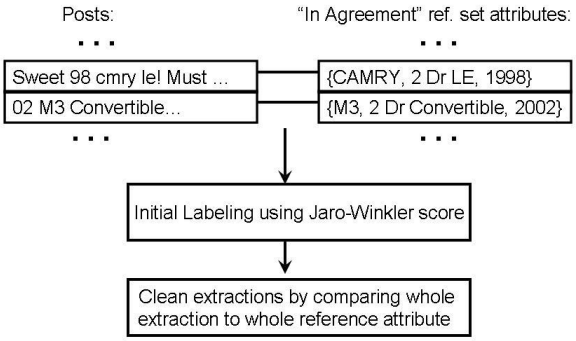

Figure 2.1: Unsupervised extraction with reference sets

2.4 Experimental Results

This section presents results for my unsupervised approach to selecting a reference set, finding the attributes in agreement, and exploiting these attributes for extraction.

2.4.1 Results: Selecting reference set(s)

The algorithm for choosing the reference sets is not tied to a particular similarity function. Rather, I apply the algorithm using many different similarity metrics and draw conclusions about which ones work best and why. This yields insight as to which types of metrics users can plug in and have the algorithm perform accurately. [1]

The experiments for selecting reference sets uses six reference sets and three sets posts to test out the various situations where the algorithm should work correctly: returning a single matching reference set, multiple matching reference sets, and no matching reference sets.

[1]Note that for the following metrics: Jensen-Shannon distance, Jaro-Winkler similarity, TF/IDF, and Jaro-Winkler TF/IDF, I use the SecondString package's implementation [17].

I use six reference sets, most of which have been used in past research. First, I use the *Hotels* reference set from [44], which consists of 132 hotels each with a star rating, a hotel name, and a local area. Another reference set from the same paper is the *Comics* reference set, which has 918 Fantastic Four and Incredible Hulk comics from the Comic Books Price guide, each with a title, issue number, and publisher. I also use two restaurant reference sets, which were both used previously for record linkage [6]. One I call *Fodors*, which contains 534 restaurant records, each with a name, address, city, and cuisine. The other is *Zagat*, with 330 records.

I also have two reference sets of cars. The first, called *Cars*, contains 20,076 cars from the Edmunds Car Buying Guide for 1990-2005. The attributes in this set are the make, model, year, and trim. I supplement this reference set with cars from before 1990, taken from the auto-accessories company, Super Lamb Auto. This supplemental list contains 6,930 cars from before 1990. I call this combined set of 27,006 records the *Cars* reference set. The other reference set of cars also has the attributes make, model, year, and trim. However, it is a subset of the cars covered by *Cars*. This data set comes from the Kelly Blue Book car pricing service containing 2,777 records for Japanese and Korean cars from 1990-2003. I call this set *KBBCars*. This data set has also been used in the record linkage community [46]. A summary of the reference sets is given in Table 2.4.

Table 2.4: Reference Set Descriptions

Name	**Source**	**Website**	**Attributes**	**Records**
Fodors	Fodors Travel Guide	www.fodors.com	name, address, city, cuisine	534
Zagat	Zagat Restaurant Guide	www.zagat.com	name, address, city, cuisine	330
Comics	Comics Price Guide	www.comicspriceguide.com	title, issue, publisher	918
Hotels	Bidding For Travel	www.biddingfortravel.com	star rating, name, local area	132
Cars	Edmunds and Super Lamb Auto	www.edmunds.com and www.superlambauto.com	make, model, trim, year	27,006
KBBCars	Kelly Blue Book Car Prices	www.kbb.com	make, model, trim, year	2,777

I chose sets of posts to test different cases that exist for finding the appropriate reference sets. One set of posts matches only a single reference set in the experimental repository. It contains 1,125 posts from the forum Bidding For Travel. These posts, called *BFT*, match only the *Hotels* reference set. This data set was used previously in [44].

Because my approach can also select multiple relevant reference sets, I use a set of posts that matches both reference sets of cars. This set, which I call *Craig's Cars*, contains 2,568 car posts from Craigslist classifieds. Note that while there may be multiple, appropriate reference sets, they also might have an internal ranking. In this case I expect that both the *Cars* and *KBBCars* reference sets are selected, but *Cars* should be ranked first (since $KBBCars \subset Cars$).

Lastly, I must examine whether the algorithm suggests that no relevant reference sets exist in the repository. To test this feature, I collected 1,099 posts about boats from Craigslist, called *Craig's Boats*. Boats are similar enough to cars to make this task non-trivial, because boats and cars are both made by Honda, for example, so that keyword appears in both sets of posts. However, boats also differ from each of the reference sets so that no reference set should be selected.

All three sets of posts are summarized in Table 2.5.

Table 2.5: Posts Set Descriptions

Name	**Source**	**Website**	**Reference Set Match**	**Records**
BFT	Bidding For Travel	www.biddingfortravel.com	Hotels	1,125
Craig's Cars	Craigslist Cars	www.craigslist.org	Cars, KBBCars	2,568
Craig's Boats	Craigslist Boats	www.craigslist.org		1,099

The first metric I test is the Jensen-Shannon distance (JSD) [39]. This information-theoretic metric quantifies the difference in probability distributions between the tokens in the reference set and those in the set of posts. Because JSD requires probability distributions, I define the distributions as the likelihood of tokens occurring in each document.

Table 2.6 shows the results for choosing relevant reference sets using JSD. The reference set names in bold reflect those that are selected for the given set of posts. (This means Craig's Boats should have no bold names.) The scores in bold are the similarity scores for the chosen reference sets. The percent difference in bold is the point at which the algorithm breaks out and returns the appropriate reference sets. In particular, using JSD the algorithm successfully identifies the multiple cases of a single appropriate reference set, multiple reference sets, or no reference set. Note that in all experiments I maintain the percent-difference splitting-threshold at 0.6.

Table 2.6: Using Jensen-Shannon distance as similarity measure

BFT Posts		
Ref. Set	Score	% Diff.
Hotels	**0.622**	**2.172**
Fodors	0.196	0.050
Cars	0.187	0.248
KBBCars	0.150	0.101
Zagat	0.136	0.161
Comics	0.117	
Average	0.234	

Craig's Cars		
Ref. Set	Score	% Diff
Cars	**0.520**	**0.161**
KBBCars	**0.447**	**1.193**
Fodors	0.204	0.144
Zagat	0.178	0.365
Hotels	0.131	0.153
Comics	0.113	
Average	0.266	

Craig's Boats		
Ref. Set	Score	% Diff.
Cars	0.251	0.513
Fodors	0.166	0.144
KBBCars	0.145	0.088
Comics	0.133	0.025
Zagat	0.130	0.544
Hotels	0.084	
Average	0.152	

The next metric I test is cosine similarity using TF/IDF for the weights. These results are shown in Table 2.7. Similarly to JSD, TF/IDF is able to identify all of the cases correctly. However, in choosing both car reference sets for the *Craig's Cars* posts, TF/IDF incorrectly determines the internal ranking, placing KBBCars ahead of Cars. This due to the IDF weights that are calculated for the reference sets. Although the algorithm treats the set of posts and the reference set as a single document for comparison, the IDF weights are based on individual records in the reference set. Therefore, since Cars is a superset of KBBCars, certain tokens are weighted more in KBBCars than in Cars, resulting in higher matching scores. These results also justify the need for a double stopping criterion. It is not sufficient to only consider the percent difference as an indicator of relative superiority amongst the reference sets. The scores must also be compared to an average to ensure that the algorithm does not errantly choose a bad reference set simply because it is relatively better than an even worse one. The last two rows of the *Craig's Boats* posts and the *BFT Posts* in Table 2.7 show this behavior.

I also tried a simpler bag-of-words metric that does not use any sort of token probabilities or weights. Namely, I use the Jaccard similarity, which is defined as the tokens in common divided by the union of the token sets. That is, given token set S and token set T, Jaccard is:

$$Jaccard(S,T) = \frac{S \cap T}{S \cup T}$$

The results using the Jaccard similarity are shown in Table 2.8. One of the most surprising aspects of these results is that the Jaccard similarity actually does pretty well. It gets all but one of the cases correct. It is able to link the *BFT Posts* to the Hotels

Table 2.7: Using TF/IDF as the similarity measure

BFT Posts		
Ref. Set	Score	% Diff.
Hotels	**0.500**	**1.743**
Fodors	0.182	0.318
Comics	0.134	0.029
Zagat	0.134	0.330
Cars	0.100	1.893
KBBCars	0.035	
Average	0.182	

Craig's Cars		
Ref. Set	Score	% Diff
KBBCars	**0.122**	**0.239**
Cars	**0.099**	**1.129**
Zagat	0.046	0.045
Fodors	0.044	0.093
Comics	0.041	0.442
Hotels	0.028	
Average	0.063	

Craig's Boats		
Ref. Set	Score	% Diff.
Cars	0.200	0.189
Comics	0.168	0.220
Fodors	0.138	0.296
Zagat	0.107	0.015
KBBCars	0.105	0.866
Hotels	0.056	
Average	0.129	

set only and it is able to determine that no reference set is appropriate for the *Craig's Boats* posts. However, with the *Craig's Cars* posts, it is only able to determine one of the reference sets. It ranks the Fodors and Zagat restaurants ahead of KBBCars because the restaurants have city names in common with some of the classified car listings. However, it is unable to determine that city name tokens are not as important as car makes and models. This is a problem if, for instance, the post is small and it contains a few more city name tokens than car tokens.

Lastly, I investigate whether requiring that tokens match strictly, as in the above metrics, is more useful than a soft matching technique that considers tokens matching based on string similarities. To test this idea I use a modified version of TF/IDF where tokens are considered a match when their Jaro-Winkler score is greater than 0.9. These results are shown in Table 2.9. This metric is able to correctly retrieve the reference set

Table 2.8: Using Jaccard as the similarity measure

BFT Posts		
Ref. Set	Score	% Diff.
Hotels	**0.272**	**1.489**
Comics	0.109	0.166
Fodors	0.094	0.004
Zagat	0.093	0.640
Cars	0.057	0.520
KBBCars	0.037	
Average	0.110	

Craig's Cars		
Ref. Set	Score	% Diff
Cars	**0.207**	**1.339**
Fodors	0.088	0.126
Zagat	0.078	0.005
KBBCars	0.078	0.089
Comics	0.072	2.536
Hotels	0.020	
Average	0.091	

Craig's Boats		
Ref. Set	Score	% Diff.
Cars	0.129	0.366
Comics	0.095	0.347
Fodors	0.070	0.112
Zagat	0.063	0.261
KBBCars	0.050	1.917
Hotels	0.017	
Average	0.071	

for the *BFT Posts*, and for the *Craig's Boats* posts this metric chooses no appropriate reference set. However, for the Craig's Boats case, this metric is close to returning many incorrect reference sets since with the Zagat reference set the score is very close to the average while the percent difference is huge. The largest failure is with the *Craig's Cars* posts. For this set of posts the ordering of the reference sets is correct because the Cars and KBBCars have the highest scores, but no reference set is chosen because the percent difference is never above the threshold with a similarity score above the average. The percent differences are low because of the soft nature of the token matching. For example, the Comics contains many matches because the issue number of the comics, such as #99, #199, #299, etc. match an abbreviated car year in a post such as '99. So, the similarity scores between the cars and comics reference sets are close. From these sets of results it becomes clear that it is better to have strict token matches, although they may be

less frequent. The strict nature of the matches ensures that the tokens particular to a reference set are used for matches, which helps differentiate reference sets.

Table 2.9: Using Jaro-Winkler TF/IDF as the similarity measure

BFT Posts		
Ref. Set	Score	% Diff.
Hotels	**0.593**	**1.232**
Fodors	0.266	0.173
Zagat	0.227	0.110
Cars	0.204	0.068
Comics	0.191	0.777
KBBCars	0.108	
Average	0.265	

Craig's Cars		
Ref. Set	Score	% Diff
Cars	0.699	0.174
KBBCars	0.595	0.174
Comics	0.505	0.050
Fodors	0.481	0.107
Zagat	0.435	0.948
Hotels	0.223	
Average	0.490	

Craig's Boats		
Ref. Set	Score	% Diff.
Cars	0.469	0.096
Comics	0.428	0.106
Fodors	0.387	0.110
KBBCars	0.349	0.082
Zagat	0.322	1.212
Hotels	0.146	
Average	0.350	

The performance of each metric is shown in Table 2.10. This table shows each metric I tested, and whether or not it correctly identified the reference set(s) for that set of posts. In the case of Craig's Boats, I consider it correct if no reference set is chosen. The last column shows whether the method was able to also rank the chosen reference sets correctly for the Craig's Cars posts.

Table 2.10: A summary of each method choosing reference sets

Method	**BFT**	**Craig's Boats**	**Craig's Cars**	**Craig's Cars rank**
JSD	√	√	√	√
TF/IDF	√	√	√	X
J-W TF/IDF	√	√	X	X
Jaccard	√	√	X	X

Based on these sets of results, I draw some conclusions about which metrics should be used and why. Comparing the Jaccard similarity results to the success of both JSD and TF/IDF, it is clear that it is necessary to include some notion of importance regarding the matching tokens. The results argue that probability distributions of the tokens as defined in JSD are a better metric, since TF/IDF can be overly sensitive, i.e., ignoring tokens that may be important, even though they are frequent. This claim is also justified by the fact that using JSD the algorithm does not run into the situation where it needs to use the average stopping criterion, although it does with the TF/IDF metric. However, since JSD and TF/IDF both do well, I can say that if a similarity metric can differentiate important tokens from those that are not, then it can be used successfully in the algorithm to choose reference sets. This is why I do not tie this algorithm to any particular metric, since many could work.

Another interesting aspect of these results is the poor performance of TF/IDF using the Jaro-Winkler modification. It seems that boosting the number of tokens that match by using a less strict token matching method actually harms the ability to differentiate between reference sets. This suggests that the tokens that define reference sets need to be emphasized by matching them exactly. Lastly, across different domains and even across different similarity metrics, the chosen threshold of 0.6 is appropriate for the cases where the algorithm chooses the correct reference set. In all of the cases where the correct reference set(s) is chosen, this threshold is exceeded.

2.4.2 Results: Matching posts to the reference set

Once the relevant reference sets are chosen, I use them to find the attributes in agreement for the different sets of posts, since the algorithm will use these attributes during extraction. For the set of posts with reference sets (i.e., BFT and Craig's Cars), I compare the attributes in agreement found by the vector-space model to the attributes in agreement for the true matches between the posts and the reference set. To evaluate the annotation, I use the traditional measures of precision, recall, and F-measure (i.e., the harmonic mean between precision and recall). A correct match occurs when the attributes match between the true matches and the predictions of the vector-space model. As stated previously, I try multiple modified token-similarity metrics to draw conclusions about what types of metrics work and why.

Table 2.11 shows the results using the modified Dice similarity for the BFT and Craig's Cars posts, varying the Jaro-Winkler token-match threshold from 0.85 to 0.99. That is, above this threshold two tokens will be considered a match for the modified Dice. In both domains there is an improvement in F-measure as the threshold increases. This is because more relevant tokens are being included when the threshold increases. If the threshold is low then many irrelevant tokens are considered matches, so the algorithm makes incorrect record level matches. For instance, as the threshold becomes low in the Craig's Cars domain, the results for the year attribute drop steeply because often the difference in years is a single digit, which in turn yields errantly matched tokens for a low threshold string similarity. Since errant matches are ignored (because they are not "in agreement") the scores are low. Note, however, that once the threshold becomes too

high, at 0.99, the results start to decrease. This happens because a very strict (high value) edit-distance is too restrictive, so does not capture some of the possible matching tokens that might be slightly misspelled.

The only attribute where the F-measure increases at 0.99 versus 0.95 is the year attribute of the Craig's Cars domain. In this case, the recall slightly increases at 0.99, but the precisions are almost the same, yielding a higher F-measure for the matching threshold of 0.99. This is due to more year values being "in agreement" with the higher threshold since there will be less variation in terms of which reference set values can match for this attribute, so those that do will likely be in agreement. For an example where the threshold of 0.95 includes years that are not in agreement with high Jaro-Winkler scores, consider a post with the token "19964" which might be a price or a year. If the reference set record's year attribute is "1994," the Jaro-Winkler score between "19964" and "1994" is 0.953. If the reference set record's year is "1996" the Jaro-Winkler score is 0.96. In both cases, a threshold of 0.95 includes both years, so if this post matches two reference set records with the same make, model and trim, but differing years of 1994 and 1996, then the year is discarded because it is not in agreement. We almost see the same behavior with the trim attribute as well. This is because with both of these attributes, a single difference in a character, say "LX" versus "DX" for a trim (or a digit for the year) yields a completely different attribute, which can then become not "in agreement."

Table 2.12 shows the results using the modified Jaccard similarity. As with the Dice similarity, the modification is such that two tokens are put into the intersection of the Jaccard similarity if their Jaro-Winkler score is above the threshold. The most striking

Table 2.11: Annotation results using modified Dice similarity

BFT posts				
Threshold	Attribute	Recall	Precision	F-Measure
0.85	Hotel name	76.46	78.13	77.29
	Star rating	80.74	77.86	79.27
	Local area	88.04	85.46	86.73
0.9	Hotel name	88.23	88.49	88.36
	Star rating	91.73	87.64	89.64
	Local area	93.09	89.36	91.19
0.95	Hotel name	88.42	88.51	**88.47**
	Star rating	92.32	87.79	**90.00**
	Local area	93.97	89.44	**91.65**
0.99	Hotel name	87.84	88.36	88.10
	Star rating	92.02	87.76	89.84
	Local area	93.39	89.39	91.34
Craig's Cars posts				
Threshold	Attribute	Recall	Precision	F-Measure
0.85	make	81.57	77.64	79.55
	model	57.61	61.15	59.33
	trim	38.76	29.80	33.70
	year	2.38	8.14	3.68
0.9	make	88.41	82.82	85.52
	model	76.28	77.16	76.72
	trim	65.57	48.12	55.51
	year	69.45	82.88	75.58
0.95	make	93.96	86.35	**89.99**
	model	82.62	81.35	**81.98**
	trim	71.62	51.95	**60.22**
	year	78.86	91.01	84.50
0.99	make	93.51	86.33	89.78
	model	81.29	81.25	81.27
	trim	71.75	51.85	60.20
	year	79.14	90.94	**84.63**

result is that the scores match exactly to those using the Dice similarity. The Dice similarity and Jaccard similarity can be used interchangeably. Further investigation reveals that the actual similarity scores between the posts and their reference set matches are different, which should be the case, but the resulting attributes that are "in agreement" are the same using either metric. Therefore, they yield the same annotation from the matches.

Table 2.12: Annotation results using modified Jaccard similarity

BFT posts				
Threshold	Attribute	Recall	Precision	F-Measure
0.85	Hotel name	76.46	78.13	77.29
	Star rating	80.74	77.86	79.27
	Local area	88.04	85.46	86.73
0.9	Hotel name	88.23	88.49	88.36
	Star rating	91.73	87.64	89.64
	Local area	93.09	89.36	91.19
0.95	Hotel name	88.42	88.51	**88.47**
	Star rating	92.32	87.79	**90.00**
	Local area	93.97	89.44	**91.65**
0.99	Hotel name	87.84	88.36	88.10
	Star rating	92.02	87.76	89.84
	Local area	93.39	89.39	91.34
Craig's Cars posts				
Threshold	Attribute	Recall	Precision	F-Measure
0.85	make	81.57	77.64	79.55
	model	57.61	61.15	59.33
	trim	38.76	29.80	33.70
	year	2.38	8.14	3.68
0.9	make	88.41	82.82	85.52
	model	76.28	77.16	76.72
	trim	65.57	48.12	55.51
	year	69.45	82.88	75.58
0.95	make	93.96	86.35	**89.99**
	model	82.62	81.35	**81.98**
	trim	71.62	51.95	**60.22**
	year	78.86	91.01	84.50
0.99	make	93.51	86.33	89.78
	model	81.29	81.25	81.27
	trim	71.75	51.85	60.20
	year	79.14	90.94	**84.63**

Table 2.13 shows results using the Jaro-Winkler TF/IDF similarity measure. Similarly to the other metrics, for the BFT domain as the threshold increases so does the F-measure, until the threshold peaks at 0.95 after which it decreases in accuracy. However, with the Craig's Cars domain the modified TF/IDF performs the best with a threshold of 0.99.

From these results, across all metrics, a threshold of 0.95 performs the best for the BFT domain. In the Cars domain, the 0.95 threshold works best for the modified Dice and Jaccard, and at this threshold both methods outperform the TF/IDF metric, even

when its threshold is at its best at 0.99. Therefore, the most meaningful comparisons between the different metrics can be made at the threshold 0.95. In the BFT domain, the modified TF/IDF outperforms the Dice and Jaccard metrics, until this threshold of 0.95. At this threshold level, the Jaccard and Dice metrics outperform the TF/IDF metric on the two harder attributes, the hotel name and the hotel area. In the Cars domain, the TF/IDF metric is outperformed at every threshold level, except on the year attribute. Interestingly, at the lowest threshold levels TF/IDF performs terribly because the tokens that match in the computation have very low IDF scores since they match so many other tokens in the corpus, resulting in very low TF/IDF scores. If the scores are low, then many records will be returned and almost no attributes will ever be in agreement, yielding very few correct annotations.

These three sets of results allows me to draw some conclusions about the utility of different metrics for the vector-space matching task. The biggest difference between the TF/IDF metric and the other two is that the TF/IDF metric uses term weights computed from the set of tokens in the reference set. The key insight of IDF weights are their ability to discern meaningful tokens from non-meaningful ones based on the inter-document frequency. The assumption is that more meaningful tokens occur less frequently. However, almost all tokens in a reference set are meaningful, and it is sometimes the case that very meaningful tokens in a reference set occur very often. The most glaring instances of this occur with the make and year attributes in the Cars reference set used for the Craig's Cars posts. Makes such as "Honda" occur quite frequently in the data set, and given that for 20,076 car records the years only range from 1990 to 2005, the re-occurrence of the same year tokens are very, very frequent. These attributes will be deemphasized

Table 2.13: Annotation results using modified TF/IDF similarity

BFT posts				
Threshold	Attribute	Recall	Precision	F-Measure
0.85	Hotel name	85.89	78.91	82.25
	Star rating	92.32	84.81	88.40
	Local area	94.55	86.86	90.54
0.9	Hotel name	90.76	83.83	87.16
	Star rating	95.33	88.05	91.55
	Local area	94.36	87.15	90.61
0.95	Hotel name	90.47	83.63	**86.92**
	Star rating	97.18	89.84	**93.36**
	Local area	94.55	87.41	**90.84**
0.99	Hotel name	89.69	82.91	86.17
	Star rating	96.89	89.57	93.08
	Local area	93.68	86.60	90.00
Craig's Cars posts				
Threshold	Attribute	Recall	Precision	F-Measure
0.85	make	51.14	44.51	47.60
	model	41.93	35.54	38.47
	trim	43.10	15.50	22.80
	year	35.58	29.18	32.06
0.9	make	67.07	58.53	62.51
	model	61.79	52.48	56.76
	trim	67.81	22.90	34.24
	year	58.48	48.24	52.87
0.95	make	88.55	77.30	82.54
	model	84.55	71.84	77.68
	trim	81.21	26.86	**40.37**
	year	76.29	62.78	68.88
0.99	make	88.90	77.65	**82.90**
	model	84.74	72.02	**77.86**
	trim	80.29	26.52	39.87
	year	76.67	63.12	**69.24**

significantly because of their frequency. If the matching token metric ignores the year, this attribute will often not be in agreement since multiple records of the same car for different years will be returned. Thus, TF/IDF has low scores for the year attribute. TF/IDF also creates problems by overemphasizing unimportant tokens that occur rarely. Consider the following post, *"Near New Ford Expedition XLT 4WD with Brand New 22 Wheels!!! (Redwood City - Sale This Weekend !!!) $26850"* which TF/IDF matches to the reference set record

{VOLKSWAGEN, JETTA, 4 Dr City Sedan, 1995}. In this case, the very rare token "City" causes an errant match because it is weighted so heavily. In the case of the BFT posts, since the Hotel reference set has few commonly occurring tokens amongst a small set of records, this phenomena is not as observable. Since weights, whether based on TF/IDF or probabilities, rely on frequencies, such an issue will likely occur in most matching methods that rely on the frequencies of tokens to determine their importance.

Therefore, I draw the following conclusions. In the matching step, an edit-distance should be used to make soft matches between the tokens of the post and the reference set records. If the Jaro-Winkler metric is used, the threshold should be set to 0.95, since that yields the highest improvement using the best metrics. Lastly, and most importantly, reference sets do not adhere to the assumptions made by weighting schemes, so only metrics that do not use such schemes, such as the Dice and Jaccard similarities, should be used, rather than TF/IDF.

Earlier I stated that the Jaro-Winkler metric emphasizes matching proper nouns, rather than more common words, because it considers the prefix of words to be an important indicator of matching. This is in contrast to traditional edit distances that define a transformation over a whole string, which are better for generic words where the prefix might not indicate matching. Table 2.14 compares using Dice similarity modified with the Jaro-Winkler metric to Dice modified with the Smith-Waterman distance [56]. (Smith-Waterman is a classic edit distance originally developed to align DNA sequences.) As with the Jaro-Winkler score, if two tokens have a Smith-Waterman distance above 0.95 they are considered a match in the modified Dice similarity. As the table shows, the Jaro-Winkler Dice score outperforms the Smith-Waterman variant. Since many of the

reference set attributes are proper nouns, the Jaro-Winkler is better suited for matching, which is especially apparent when using the Cars reference set.

Table 2.14: Modified Dice using Jaro-Winkler versus Smith-Waterman

BFT posts				
Method	Attribute	Recall	Precision	F-Measure
Jaro-Winkler	Hotel name	88.42	88.51	**88.47**
	Star rating	92.32	87.79	**90.00**
	Local area	93.97	89.44	**91.65**
Smith-Waterman	Hotel name	67.80	69.56	68.67
	Star rating	75.39	73.88	74.63
	Local area	84.34	81.72	83.01
Craig's Cars posts				
Method	Attribute	Recall	Precision	F-Measure
Jaro-Winkler	make	93.96	86.35	**89.99**
	model	82.62	81.35	**81.98**
	trim	71.62	51.95	**60.22**
	year	78.86	91.01	**84.50**
Smith-Waterman	make	24.12	27.18	25.56
	model	14.71	18.82	16.52
	trim	11.17	5.81	7.64
	year	29.17	52.30	37.45

2.4.3 Results: Extraction using reference sets

The main research exercise of the this thesis is information extraction from unstructured, ungrammatical text. Therefore, Table 2.15 shows our "field-level" results for performing information extraction exploiting the attributes in agreement. Field-level extractions are counted as correct only if *all* tokens that compromise that field in the post are correctly labeled. Although this is a much stricter rubric of correctness (versus token-level correctness), it more accurately models how useful an extraction system would be. The results from my system are presented as ARX[2]

I compare ARX's results to three systems that rely on supervised machine learning for extraction. I compare against two different Conditional Random Field (CRF)

[2]**A**utomatic **R**eference-set based e**X**traction

[32] extractors built using MALLET [42], since CRFs are a comparison against both state-of-the-art extraction and methods that rely on structure. The first method, called "CRF-Orth" includes only orthographic features, such as whether a token is capitalized or contains punctuation. The second CRF method, called "CRF-Win" contains orthographic features and sliding window features such that it considers both the two-tokens preceding the current token, and two-tokens afterward for labeling. CRF-Win in particular will demonstrate how the unstructured nature of posts makes extraction difficult. I trained these extractors using 10% of the labeled posts for training, since this is small enough amount to make it a fair comparison to the automatic method, but is also large enough such that the extractor can learn well enough.

I also compare ARX to Amilcare [14], which relies shallow NLP parsing for extraction. Amilcare also provides a good comparison because it can be supplied with "gazetteers" (lists of nouns) to aid extraction, so it was provided with the unique set of attribute values (such as Hotel areas) for each attribute. Note, however, that Amilcare was trained using 30% of the labeled posts for training so that it could learn a good model (however, even this large amount of training did not seem to help as it only outperformed ARX on two attributes). All supervised methods are run 10 times, and the results reported are averages.

Although the results are not directly comparable since I compare systems that require labeled data (supervised machine learning) to ARX, they still point to the fact that ARX performs very well. Although the ARX system is completely automatic from selecting its own reference set all the way through to extraction, it still performs the best on 5/7 attributes. The two attributes where it is not the best, (star rating of BFT and year of

Table 2.15: Extraction results

Craig's Cars Posts				
Attribute		Recall	Prec.	F-Mes.
Make	ARX	95.99	100.00	**97.95**
	CRF-Orth	82.03	85.40	83.66
	CRF-Win	80.09	77.38	78.67
	Amilcare	97.58	91.76	94.57
Model	ARX	83.02	95.01	**88.61**
	CRF-Orth	71.04	77.81	74.25
	CRF-Win	67.87	69.67	68.72
	Amilcare	78.44	84.31	81.24
Trim	ARX	39.52	66.94	**49.70**
	CRF-Orth	47.94	47.90	47.88
	CRF-Win	38.77	39.35	38.75
	Amilcare	27.21	53.99	35.94
Year	ARX	76.28	99.80	86.47
	CRF-Orth	84.99	91.33	88.04
	CRF-Win	83.03	86.12	84.52
	Amilcare	86.32	91.92	**88.97**
BFT Posts				
Attribute		Recall	Prec.	F-Mes.
Star Rating	ARX	83.94	99.44	91.03
	CRF-Orth	94.37	95.17	94.77
	CRF-Win	93.77	94.67	94.21
	Amilcare	95.58	97.35	**96.46**
Hotel Name	ARX	70.09	77.16	**73.46**
	CRF-Orth	66.90	68.07	67.47
	CRF-Win	39.84	42.95	41.33
	Amilcare	58.96	67.44	62.91
Local Area	ARX	62.23	85.36	**71.98**
	CRF-Orth	64.51	77.14	70.19
	CRF-Win	28.51	39.79	33.07
	Amilcare	64.78	71.59	68.01

Craig's Cars), it is competitive (within 5% of the F-measure) of the best method, which in these two cases is Amilcare, which was supplied with 30% of the labeled data for training. Further, for these two attributes, all of the methods were competitive indicating that there is some regularity (such as the fact that these are the two attributes with numbers in them) that can be easily exploited making extraction less difficult for these two attributes than the others.

In fact, lending credibility to this fact is that the CRF-Win method is actually competitive for these two attributes, although it performs the worst on 6/7 of the attributes.

The fact that CRF-Win does so poorly indicates that the sliding window actually confused the extraction method, implying that there is not a regular structure (i.e. neighborhood window) that can be used to aid extraction. This is precisely the point of this thesis, that structure and grammar based methods will not do well for extraction of data that does not conform to structural assumptions, and therefore new methods are required, such as reference-set based extraction. The results of Table 2.15 support this claim.

One interesting point about these results involves the comparison of the recall and precision for the ARX method. In 5/7 attributes, the difference between the precision and recall is at least 10% in favor of the precision, yet the ARX method does well. This suggests that if the algorithm can increase the discovery of the extractions, it will increase the recall, and get even better results. For example, one of the attributes where this occurs is the Local Area of the BFT posts. In this attribute, one often sees acronyms such as "AP" for airport or "DT" for downtown. Supervised systems can be trained to identify such cases, but my approach would need some sort of acronym and synonym discovery method to find those. However, it could find these, it could bridge this gap between precision and recall.

Chapter 3

Automatically Constructing Reference Sets for Extraction

Chapter 2 described a method for automatically extracting data from unstructured, ungrammatical text. The process was automatic except for the necessity in manually constructing the reference set (which can be kept in the repository). However, as this Chapter will show, even this reference set creation process can be automated.

In fact, in many cases, manually constructing a reference set is not necessary, and instead the machine can construct its own reference set which it can then use for extraction and matching using the algorithms of Chapter 2. Further, in the cases where it might be unclear whether to manually create a reference set or let the machine create one, an algorithm of this Chapter can determine the answer on behalf of the user.

However, before I delve into the details for automatically constructing a reference set, I will first describe the cases where a manually constructed reference set is necessary. I call the first situation where a manual reference set is necessary the "SKU" problem, since SKUs are usually internal identifiers used by companies to identify products. The generalized version of the SKU problem is that it is possible that a user requires that a certain attribute must be added as semantic annotation to a post in support of future,

structured queries (that is, after extraction). For example, suppose an auto-parts company has an internal database of parts for cars. This company might want to join their parts database with the classified ads about cars in a effort to advertise their mechanics service. Therefore, they could use reference-set based extraction to identify the cars for sale in the classified ads, which they can then join with their internal database. Yet, their internal database might have its own internal identification attribute (SKU), and since this attribute would be needed to join the cars identified in the classifieds with their database, it would need to be included as annotation for the post. In this case, since none of the classified ads would already contain this attribute since it's specific to the company, it would not be possible to include it as an automatically constructed attribute, and therefore the company would have to manually create their own reference set to include this attribute. So, in this case, the user would manually construct a reference set and then use it for extraction.

There is a second scenario that requires a manual reference set. Specifically, in some cases the textual characteristics of the posts are such that the automatic method of constructing a reference set encounters difficulty. As stated above, not only does this chapter analyze these textual characteristics that define the domains where automatically constructing a reference set is appropriate, but further, I describe a method that can determine whether to use the automatic approach of this chapter for reference set construction, or suggest to the user to manually construct a reference set. This determination algorithm uses just a single, labeled example post.

3.1 Reference Set Construction

This chapter presents a method to automatically create a reference set with the intention of using that reference set for extraction. The process starts solely with just a set of posts, with no a priori knowledge or labeled data. In order to construct a reference set from a set of posts, I borrow from the work on concept discovery. The concept discovery task takes Web pages as input, and outputs a hierarchy of concepts, in tree form, such that some of the concepts are subsumed (children) of others. This task generally breaks into two distinct steps. First, the set of candidate terms (concepts) is selected from the corpus of Web pages. Second, these candidate terms are turned into a hierarchy. The techniques for the second step are varied. Examples include methods that range from conditional probabilities over the terms [54] to Principal Component Analysis over a term matrix [23] to smoothed Formal Concept Analysis [13]. Regardless, however, the general structure to solve the problem is clear: first find the terms to construct the hierarchy, then construct it.

I borrow from this process by noting that reference sets often form a type of hierarchy. Each column of a reference set will typically subsume the columns that come after it. This "subsumption" happens because of a specificity definition for certain attributes. For example, consider the reference set shown in Figure 3.1. In this reference set, the Civic and the Accord model are specific versions of a Honda make, and the Focus model is a specific Ford make. Therefore, one can think of Honda as a node in the hierarchy that subsumes the Civic and the Accord, while Ford subsumes Focus. Note, however, I do not claim that a reference set truly conforms to a conceptual hierarchy. Rather it is

hierarchy-like in that certain attributes may relate to others in such a way that it can be represented as a compact tree.

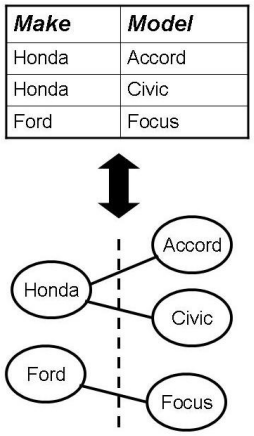

Figure 3.1: Hierarchies to reference sets

Consider the hierarchies for cars shown in Figure 3.1. Note that although these hierarchies are independent, I can combine them to form a single reference set by traversing each tree and setting each node as a column value. The columns of each hierarchy are aligned in the reference set by their position in the tree. Therefore, my task becomes constructing independent hierarchies for each entity represented by the posts, which I can then combine into a single reference set. This whole process is shown in Figure 3.2.

Mirroring concept discovery, the first step in my algorithm is to consider which tokens in my set of posts might represent actual attributes for the reference set. For this step, the algorithm simply breaks up the set of posts into bigrams, keeping only those bigrams that occur in more than one post. The condition that a bigram must occur in at least two documents is similar to condition on candidate tokens in Sanderson and Croft [54].

As an example, the second post of Table 1.1 becomes the set of bigrams: {"93- 4dr," "4dr Honda," "Honda Civc," ...}

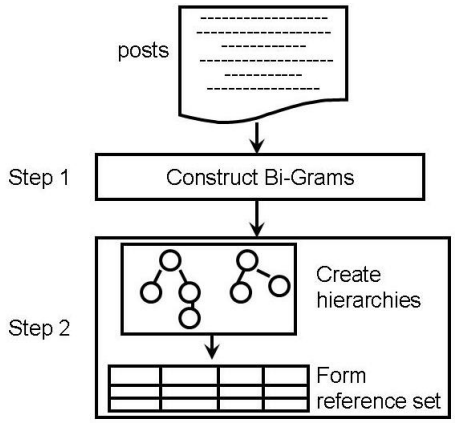

Figure 3.2: Creating a reference set from posts

Next, the method uses these bigrams to discover possible terms subsumptions for the hierarchies. I use bigrams because in many cases, in posts, users list the attributes of the entities. That is, because of the compactness of posts, users tend to use mostly attribute terms to fill in their posts. Further, they sometimes order the attributes in a natural way. For instance, in the second car classified ad of Table 1.1 the car make (Honda) is followed by the model (Civc) followed by the trim (LX). These are useful clues for building the Honda hierarchy. Note, however, that this structure does not always exist, otherwise it could always be exploited for extraction.

Although the algorithm exploits the structure of *some* of the posts in order to discover the reference set, this does not imply that *all* of the posts maintain this structure. The point of the algorithm is to construct a reference set, without labeled data or a priori knowledge, which can then be used for information extraction without any assumption

on the structure of the posts. So, by exploiting a little of the structure, the algorithm performs information extraction from the posts that do not have structure.

Once the first step completes, the algorithm has a set of bigrams from which to construct the independent hierarchies which will be translated into a reference set. To construct the hierarchies, I use a slightly modified version of the conditional probability test of Sanderson and Croft [54], rewritten later [33] as[1]:

$$x \text{ SUBSUMES } y \text{ IF } P(x|y) \geq 0.75 \text{ AND } P(y|x) \leq P(x|y)$$

For example, using the bigram "Honda Civc" I would test if $P(\text{Honda}|\text{Civc}) \geq 0.75$ and $P(\text{Civc}|\text{Honda}) \leq P(\text{Honda}|\text{Civc})$. However, unlike the concept hierarchy work, the attributes are not necessarily single token terms. So, I modify the conditional probability heuristic to account for the fact that I may need to merge multiple terms to form a single attribute.

Intuitively, if two terms almost always co-occur with each other, but only with each other, then they are likely to be part of the same attribute. Therefore, I create the heuristic that if x subsumes y, and $P(y|x)$ is also greater than the threshold, then I should merge the terms.[2] So, if both terms are subsuming each other, and therefore likely co-occur frequently, but only with each other (otherwise $P(x|y)$ would be greater but not $P(y|x)$), they should be merged. So, the merging heuristic is defined as:

$$\text{MERGE}(x,y) \text{ IF } x \text{ SUBSUMES } y \text{ AND } P(y|x) \geq 0.75$$

Once this process finishes, I have multiple, independent hierarchies representing the entities described by the posts. The algorithm flattens these hierarchies into the reference

[1]The threshold was originally set as 0.8 by Sanderson and Croft [54], but I found that 0.75 worked better empirically for my problem.

[2]This requires that relax the assumption that $P(y|x) < P(x|y)$, since they could be equal as well.

set by aligning columns along the tree depth, as shown in Figure 3.1. The whole algorithm in shown in Table 3.1. Since this algorithm constructs a reference set using a single pass over the posts, I call it the "Single Pass" algorithm.

Table 3.1: Constructing a reference set

```
BuildReferenceSet(Posts P)
# The Single Pass algorithm
Bigrams B ← {}
Hierarchies H ← {}
∀ posts p ∈ P
   ∀ tokens t_i ∈ p, 1 ≤ i < NumToks(p)
      B ← B ∪(t_i, t_{i+1})
For all bigrams b ∈ B
   t_1, t_2 ← SplitBigram(b)
   If BigramCount(b) > 1 and
   P(t_1|t_2) ≥ 0.75 and P(t_2|t_1) ≤ P(t_1|t_2)
      t_1 subsumes t_2
         H ← UpdateHierarchies(t_1,t_2)
      If t_1 subsumes t_2 and P(t_2|t_1) ≥ 0.75
         H ← MergeTerms(t_1,t_2,H)
For all hierarchies h ∈ H
   FlattenToReferenceTuples(h)
```

Although the above approach works well for finding certain attributes and their relationship, one limitation stems from what I call the "general token" effect. Since subsumption is determined by the conditional probabilities of the tokens, when the second token is frequent (more general) than the first token, the conditional probability will be low and not yield a subsumption.

An example of this general token effect can be seen with the trim attribute of cars. For instance, consider the posts in Table 3.2 which show the general token effect for the trim value of "LE." These posts show the "LE" trim occurring with a Corolla model, a Camry model, a Grand AM model, and a Pathfinder. However, in the labeled data (used for experiments), although the "CAMRY LE" has the highest conditional probability for

an 'LE' bigram, it is only 49%. This is due to the fact that so many other bigrams share the LE value as their second token (e.g., Corolla, Pathfinder, etc.). Therefore, since LE happens across many different posts in many varying bigrams, I call it a "general" token, and its conditional probability will never be high enough for subsumption. Thus it is never inserted into the reference set.

Table 3.2: Posts with a general trim token: LE

2001 Nissan Pathfinder LE - $15000
Toyota Camry LE 2003 —- 20000 $15250
98 Corolla LE 145K, Remote entry w/ alarm, $4600
1995 Pontiac Grand AM LE (Northern NJ) $700

To compensate for this "general token" peculiarity, I iteratively run the Single Pass algorithm, where for each iteration after the first, I consider the conditional probability using a set of the first tokens from bigrams that all share the common second token in the bigram. Note, however, this set only contains bigrams whose first token is already a node in a hierarchy, otherwise the algorithm may be counting noise in the conditional probability. This is the reason the algorithm can only iterate after the first pass. The algorithm iterates until it no longer generates new attribute values. I call this version of the approach the "Iterative" algorithm.

As an example, consider again the 'LE' trim. The iterative approach considers the following conditional probability for subsumption, assuming the models of Camry, Corolla and Pathfinder have already been discovered:

$$P(\{CAMRY \cup COROLLA \cup PATHFINDER\}|LE)$$

Now, if this conditional probability fits the heuristic for subsumption (or merging), then I will add LE as a child to the nodes CAMRY, COROLLA and PATHFINDER in their own respective hierarchies.

The algorithm is now equipped to deal with the "common token" effect. However, there are still two aspects of this approach that need clarification. First, how many posts should be supplied for constructing the reference set? There are literally millions of car classified ads, but it is unclear how many should be used for construction. Intuitively, it seems there is redundancy (an assumption I use to build the reference sets), so one would not need to see all of the classified ads. However, can one determine this stopping point algorithmically? Further, given that in order to overcome the general token issue the algorithm iteratively generates reference set tuples, some noisy attributes are introduced. However, manually pruning away this noise takes away from the automatic nature of my approach.

To deal with both of these issues simultaneously, I introduce a "locking" mechanism into the algorithm. Since many of the attributes the algorithm discovers are specifications of more general attributes (such as car models specify makes), there is a point at which although the algorithm may be discovering new values for the more specific attributes (car models), it has saturated what it can discover for the parent attribute (car makes). Further, once it saturates the parent attributes, if the algorithm does introduce further values, they are likely noise. So, the algorithm should "lock" the parent attribute at the saturation point, and only allow the discovery new attributes that are below the level of locked attribute in the hierarchies.

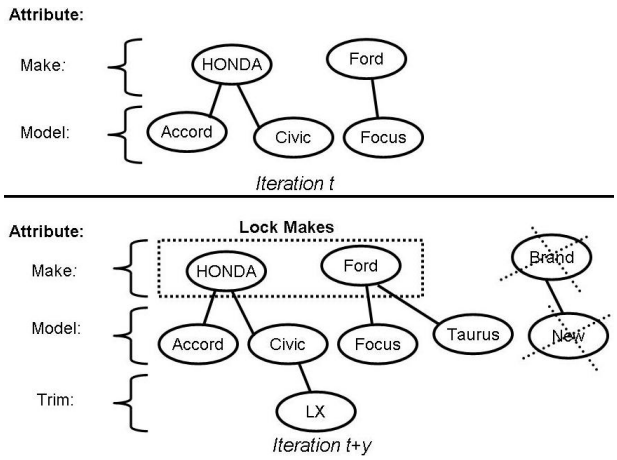

Figure 3.3: Locking car attributes

Consider the example shown in Figure 3.3. At iteration t the algorithm has discovered two hierarchies, one rooted with the car make Ford and one rooted with the car make Honda. Each of these makes also has a few models associated with them (their children). The bottom of the figure shows some future time (iteration $t + y$), at which point the system decides to lock the make attribute, but not the model and trim. Therefore, the algorithm can still add car models (as it does with the Taurus model to the Ford make) and car trims (as with the LX trim for the Civic model). However, since the make attribute is locked, no more make attributes are allowed, and therefore the hierarchy that would have been rooted on the make "Brand" with model "New" (which is noise) is not allowed. This is why it is shown on the right as being crossed out.

In this manner, the locking acts like a pre-pruner of attribute values. Rather than having to prune away errant attributes after discovering the reference set tuples, the intent is that the algorithm will lock the attributes at the right time so as to minimize the number of noisy attributes that may be introduced at later iterations. This works because noise is often introduced as the algorithm iterates, but not in the beginning. In

my example, instead of post-pruning away the hierarchy rooted on "Brand," the algorithm instead prevented it from being added by locking the make attribute.

The locking mechanism also provides an algorithmic method to stop processing posts. Simply, if all attributes (i.e. all levels of the hierarchies) become locked, there is no point in seeing any more posts, so that would be the appropriate number of posts at which the algorithm can stop creating reference set tuples. So, this introduces a way to determine the number of posts needed for constructing the reference set. In practice, a user can send posts into the algorithm, which in turn constructs and locks, and if attributes remain unlocked, the machine requests more posts from the user. This would repeat until all attributes are locked. Of course, given the prevalence of such technologies as RSS feeds, a user would not even have to supply the algorithm with posts. It could simply request them itself via RSS until it does not require any more.

Therefore, I leverage the notion that the locking process requests more posts from the user until it locks all attributes. Essentially, at each request, the machine compares what it can discover using the current set of given posts to what it can discover using the last set of given posts (note that the newly requested set supersedes the previous set). For example, the user starts by giving the algorithm 100 posts, and then the algorithm requests more, at which point the person supplies 100 more posts. The algorithm compares what it can discover by using the first 100 posts as compared to what it can discover using the combined 200 posts. If the algorithm thinks it can't discover more from the 200 than the 100, then it locks certain attributes (Again, note $\text{Posts}_{100} \subset \text{Posts}_{200}$).

To do this comparison for locking the algorithm compares the entropies for generating a given attribute, based upon the likelihood of a token being labeled as the given attribute.

So, in the cars domain it calculates the entropy of a token being labeled a make, model or a trim (note that these label names are given post-hoc, and the machine simply regards them as attribute$_1$, attribute$_2$, etc.). For clarity, I calculate the entropy H for the make as:

$$H(make) = -\sum_{x \in tokens} p_x * \log(p_x)$$

The entropy of labeling a particular attribute can be interpreted as the uncertainty of labeling tokens with the given attribute. So, as the algorithm sees more and more posts, if the entropy does not change from seeing 100 posts to seeing 200 posts, then the uncertainty in labeling that attribute is steady so there is no need to keep discovering attribute values for that attribute in more posts. However, I cannot directly compare the entropies across runs, since the underlying amounts of data are different. So, I use normalized entropy instead. That is, for attribute X, across N posts, I define:

$$H(X)_{Norm} = \frac{H(X)}{logN}$$

Although entropy provides a measure of uncertainly for token labels, it does not provide a sufficient comparison between runs over varying numbers of posts. To provide an explicit comparison, I use the "maximum redundancy" metric, which uses the entropies across runs to provide a normalized measure of scaled information. For a given attribute X, if we are comparing runs across 100 posts and 200 posts, denoted X_{100} and X_{200} respectively, the maximum redundancy, R_{Max} is defined as:

$$R_{Max} = \frac{\min(H(X_{100}), H(X_{200}))}{H(X_{100}) + H(X_{200})}$$

Maximum redundancy is a useful stopping criteria. When the maximum redundancy is zero, the variables are independent, and it can only reach a maximum value, (R_{Max}). Reaching R_{Max} means that one variable is completely redundant if you have knowledge about the other variable. So, if the algorithm is given some set of posts, (i.e. my 100 and 200 posts), and it finds the maximum value for R_{Max} for a given attribute between these posts then it knows it can lock that attribute at 100 posts, since the entropy using the 200 did not yield more information. So, the algorithm locks an attribute when it finds the maximum value for R_{Max} for that attribute.

Table 3.3 summarizes the above technique for locking the attributes when constructing a reference set.

Table 3.3: Locking attributes

LockingAttributes(Attributes a, Posts p_i, Posts p_j)
# p_i are the first set of posts
# p_j are the second set, such that
# $p_i \subset p_j$
for each $a \in$ Attributes
if a is not locked
$H_{Norm}(a_i) \leftarrow$ Entropy(p_i, a)
$H_{Norm}(a_j) \leftarrow$ Entropy(p_j, a)
If $R_{Max}(H_{Norm}(a_i), H_{Norm}(a_j))$ is maximum
AND Parent(a) is locked
Lock(a)
if all $a \in$ Attributes are locked
return locked

There are two small things to note. First, I add a heuristic that an attribute may only lock if its parent attribute is already locked, to prevent children from locking before parents. Second, although the above discussion repeatedly mentions the machine requesting more posts from a user, note that the algorithm can easily automatically request its

own posts from the sources using technology such as RSS feeds, alleviating any human involvement beyond supplying the URL to the source.

Therefore, by using the above technique to lock the attributes, even if the algorithm generates new children attribute values, the parents will be locked, which acts as an automatic pruning. Further, the algorithm now knows how many posts are required to automatically construct a reference set from the source of posts.

Table 3.4 ties all of the aspects together (constructing reference set tuples, iterating for the general tokens, and locking) yielding my algorithm for automatically building reference set tuples from posts, which I call the "Iterative Locking Algorithm" (ILA).

Table 3.4: ILA method for building reference sets

```
BuildReferenceSet(Posts, x, y)
# x is the number of posts to start with
# y is the number of posts to add each iteration
locked ← false
while(locked is false)
   Posts p ← GetPosts(x, y)
   ReferenceSet rs ← BuildReferenceSet(p) # BuildReferenceSet as in Table 3.1

   while(continue)
      Updates ← FindGeneralTokens(rs) # Find general tokens using combined probabilities

      if |Updates| i== 0 # No updates found
         continue ← false # Stop iterating: No more general tokens found
      else
         rs ← rs ∪ Updates # Found new general tokens, keep iterating

   Attributes a ← GetFoundAttributes(rs)
   x ← y
   Posts q ← GetPosts(x, y)
   repeat for q # Need generate attributes for q
   locked ← LockingAttributes(a, p, q) # Defined in Table 3.3
```

Figures 3.4, 3.5, and 3.6 show hierarchies that ILA constructed for different domains using real-world data. Specifically, Figure 3.4 shows an entity hierarchy constructed from

car posts from the classified website Craigslist rooted on "Honda." Figure 3.5 shows an entity hierarchy for laptop computers from Craiglist rooted on "HP," and Figure 3.6 shows a hierarchy for skis, rooted on the "Stockli" ski brand. Note the ski posts come from eBay. These examples show that across domains many attribute values are discovered correctly. However, they also demonstrate two common types of errors that occur. First, there are cases where noisy nodes are included in the hierarchies. For example, consider the node "Autos" under "Honda" in Figure 3.4 and the node "Fun" under the ski model "Sinox" in Figure 3.6, which are both noise. Another common error is placing a node in the wrong level of the tree. For example, in Figure 3.5, the "ZT3000" node is actually a specific model number for a "Pavilion" laptop, so it should be a child of "Pavilion" rather than put in its current location. This misplacement generally occurs when people leave out an implied attribute. That is, for the posts about ZT3000 laptops, the sellers assumed that the buyers would know it's a Pavilion, so in their posts they include the attributes HP and ZT3000, but leave out Pavilion. Despite these errors, as the results show, my method can use these automatically constructed reference sets for successful extraction.

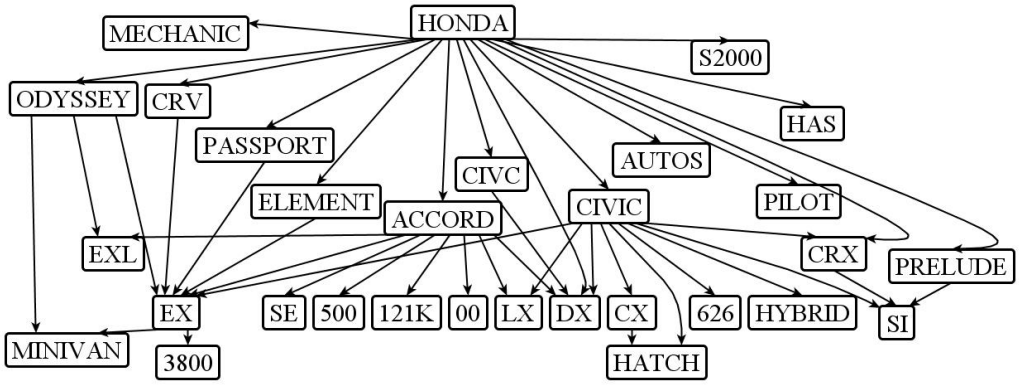

Figure 3.4: Hierarchy constructed by ILA from car classified ads

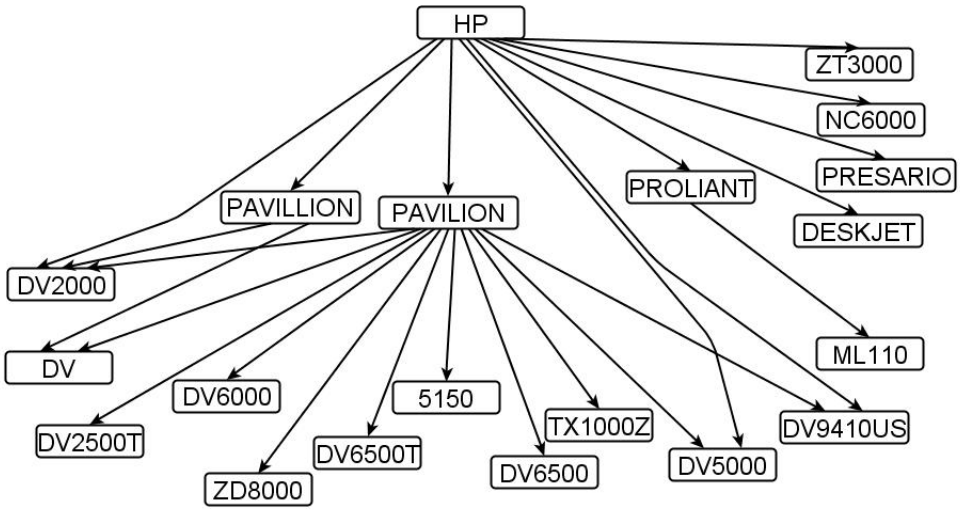

Figure 3.5: Hierarchy constructed by ILA from laptop classified ads

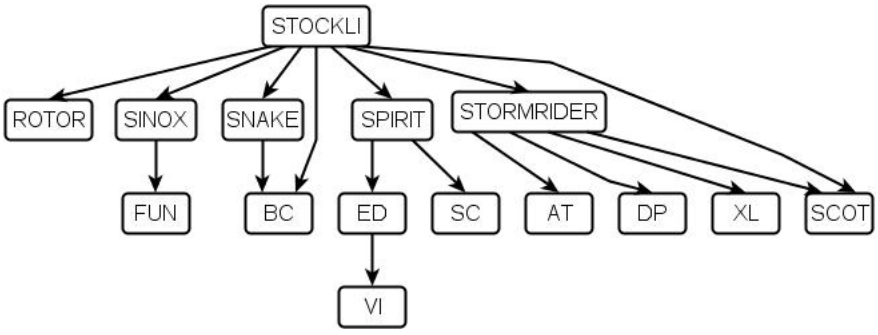

Figure 3.6: Hierarchy constructed by ILA from ski auction listings

As described above, once ILA constructs the entity hierarchies, it flattens them into reference set tuples by starting at the root node of the hierarchy and tracing each path, assigning each node along the path to an attribute. The flattened reference sets for Figures 3.4, 3.5, and 3.6 are give in Tables 3.5, 3.6, and 3.7 respectively. Note, the attribute names in the tables are provided for ease of reading. The system outputs the attribute names as "Attribute$_0$," "Attribute$_1$," etc. since it does not know the schema.

Further, the correct tuples in each table are shown in **bold**. Interestingly, in some cases the users incorrectly spelled the attribute value, but ILA still produces the correct tuple using this misspelled value. The "Honda Civc" tuples of Table 3.5 show this behavior. Lastly, determining the "correctness" of a tuple is not easy. For instance, the "CRX" car by Honda is known both as its own model, and as a type of Civic. I choose to consider it its own model.

These tables make some of ILA's mistakes clearer than the hierarchies. For example, it is much clearer that certain attributes that were placed in the wrong tree level end up in the wrong attribute slot. Another interesting mistake that sometimes occurs, and which is made clear by the tables, is that an attribute can appear in different locations for different tuples. For instance, in Table 3.6, the value "DV2000" occurs as both a model and a model number. This is due to the fact that sometimes posters include the term Pavilion when selling this laptop, and sometimes they do not.

Table 3.5: Reference set constructed by flattening the hierarchy in Figure 3.4

Make	*Model*	*Trim*	*Trim Spec*
HONDA	**ACCORD**		
HONDA	ACCORD	00	
HONDA	ACCORD	121K	
HONDA	ACCORD	500	
HONDA	**ACCORD**	**DX**	
HONDA	**ACCORD**	**EX**	
HONDA	**ACCORD**	**EXL**	
HONDA	**ACCORD**	**LX**	
HONDA	**ACCORD**	**SE**	
HONDA	AUTOS		
HONDA	**CIVC**		
HONDA	**CIVC**	**DX**	
HONDA	CIVIC	626	
HONDA	CIVIC	CRX	
HONDA	**CIVIC**	**CX**	
HONDA	**CIVIC**	**DX**	
HONDA	**CIVIC**	**EX**	
HONDA	CIVIC	HATCH	
HONDA	**CIVIC**	**HYBRID**	
HONDA	**CIVIC**	**LX**	
HONDA	**CIVIC**	**SI**	
HONDA	CIVIC	CX	HATCH

Make	*Model*	*Trim*	*Trim Spec*
HONDA	**CRV**		
HONDA	**CRV**	**EX**	
HONDA	**CRX**		
HONDA	**CRX**	**SI**	
HONDA	DX		
HONDA	**ELEMENT**		
HONDA	**ELEMENT**	**EX**	
HONDA	HAS		
HONDA	MECHANIC		
HONDA	**ODYSSEY**		
HONDA	**ODYSSEY**	**EX**	
HONDA	**ODYSSEY**	**EXL**	
HONDA	ODYSSEY	MINIVAN	
HONDA	ODYSSEY	EX	3800
HONDA	**ODYSSEY**	**EX**	**MINIVAN**
HONDA	**PASSPORT**		
HONDA	**PASSPORT**	**EX**	
HONDA	**PILOT**		
HONDA	**PRELUDE**		
HONDA	**PRELUDE**	**SI**	
HONDA	**S2000**		

Table 3.6: Reference set constructed by flattening the hierarchy in Figure 3.5

Brand	*Model*	*Model Num.*
HP	DESKJET	
HP	DV2000	
HP	DV5000	
HP	DV9410US	
HP	NC6000	
HP	**PAVILION**	
HP	**PAVILION**	**5150**
HP	PAVILION	DV
HP	**PAVILION**	**DV2000**
HP	**PAVILION**	**DV2500T**
HP	**PAVILION**	**DV5000**
HP	**PAVILION**	**DV6000**
HP	**PAVILION**	**DV6500**
HP	**PAVILION**	**DV6500T**
HP	**PAVILION**	**DV9410US**
HP	**PAVILION**	**TX1000Z**
HP	**PAVILION**	**ZD8000**
HP	**PAVILLION**	
HP	PAVILLION	DV
HP	**PAVILLION**	**DV2000**
HP	**PRESARIO**	
HP	**PROLIANT**	**ML110**
HP	ZT3000	

Table 3.7: Reference set constructed by flattening the hierarchy in Figure 3.6

Brand	*Model*	*Model Spec.*	*Other*
STOCKLI	BC		
STOCKLI	**ROTOR**		
STOCKLI	SCOT		
STOCKLI	**SINOX**		
STOCKLI	SINOX	FUN	
STOCKLI	**SNAKE**		
STOCKLI	**SNAKE**	**BC**	
STOCKLI	**SPIRIT**	**ED**	
STOCKLI	**SPIRIT**	**ED**	**VI**
STOCKLI	**SPIRIT**	**SC**	
STOCKLI	**STORMRIDER**	**AT**	
STOCKLI	**STORMRIDER**	**DP**	
STOCKLI	STORMRIDER	SCOT	
STOCKLI	**STORMRIDER**	**XL**	

One useful aspect of constructing the reference set directly from the posts, versus manually constructing the reference set from a source on the Web has to do with the nature of using reference sets for extraction. It is not enough to simply construct a reference set from data on the Web and expect it to be useful across sets of posts. For example, one might be able to construct a reference set of cars from Edmunds, but their data only goes back a few decades. What if the cars in the posts are all classic cars? However, by building the reference set directly from the posts themselves, users gain assurance that the reference set will be useful. This is evident in my experiments where I find that sometimes the reference set collected from freely available data on the Web does not always have enough useful coverage for certain attributes.

As I mention above, there are alternative methods for building term hierarchies (subsumptions) from text. However, these methods are not as well suited as the Sanderson and Croft method [54] for my problem. First, since my data is ungrammatical, I cannot use Formal Concept Analysis which relates verbs to nouns in the text to discover subsumptions [13]. I have plenty of nouns in posts, but almost no verbs. Further, since my algorithm runs iteratively due to the "general token" problem, I need a method that runs efficiently. Using the Sanderson and Croft method, my algorithm runs in $O(n)$ time where n is the number of tokens from the posts, since my process scans the posts to create the bigrams and calculate the probabilities, and then considers only those with high enough probabilities. However, methods such as Principal Component Analysis [23] run in $O(n^3)$ with respect to the token by token matrix (which may be sparse), and so this method is not suitable for a large number of tokens and more than one iteration.

3.2 Experiments

My main purpose for discovering reference sets is to use them for reference-set based information extraction from unstructured, ungrammatical data. Therefore, as my experimental procedure, I first discover the reference set for a given set of posts, and then I perform information extraction using the discovered reference set. If I have discovered a useful reference set from the posts, the extraction results reflect that. For the extraction itself, I use the automatic method of the previous chapter.[3]

I present a number of experiments using varied data sources to provide insight into different aspects of my approach. However, note that all results are reported as field-level extraction results using the standard measures of precision, recall and F_1-measure for each of the data sets.

Table 3.8: Post Sets

Name	**Source**	**Attributes**
Craig's Cars	Craigslist	make, model, trim
Laptops	Craigslist	manufacturer, model, model num.
Skis	eBay	brand, model, model spec.

The three sets of posts used in the experiments are outlined in Table 3.5. The first set of posts is the same Craig's cars set used in the previous chapter. The second set of posts are laptops for sale from Craigslist, and the third set are skis from eBay. Again, I use a variety of source types (classified ads and auction listings) and a variety of domains. For each domain I labeled 1,000 posts fully, which I use to generate the field level extraction results shown below. I concentrate on the particular attributes outlined above for each

[3]Note, however, that the ILA method sometimes includes common terms, such as "Free Shipping" in the built reference set, and so to compensate for this, I use TF/IDF for finding reference-set matches, rather than the Dice similarity, since TF/IDF will discount the commonly occurring terms.

domain because those are the common attributes found in online sources that I use to manually create reference sets for extraction to compare the ILA approach against.

The first set of experiments demonstrate the utility of using my iterative method to deal with the "general token" effect, versus doing a single pass for building the reference set. These experiments also show that the locking mechanism of the ILA approach provides a level of cleaning when using the iterative method, in that by locking certain attributes generates better results because less noisy tuples are introduced to the constructed reference set.

Table 3.9 shows the comparative results for the different domains. For each domain, the number of posts required by the system is shown next to the domain name. Although this number is determined by the locking mechanism of the ILA algorithm, it is also used for the iterative and single pass method, because without it, I would not know how many posts the algorithms should examine. Each method's results are referred to by their corresponding name (Single Pass, Iterative, and Iterative-Locking (ILA)), and the number of reference set tuples generated by each method is shown in parentheses next to the method's name.

As Table 3.9 shows, my approach can effectively deal with the general token issue. This is especially apparent for the more specific attributes, such as the Car trims, the Laptop model numbers, and the Ski model-specifications where the improvement using the iterative and locked (ILA) methods are dramatic. Further, by adding the locking mechanism I can avoid some of the noise introduced by iterating multiple times over the set of posts. This is most dramatically apparent in the Laptop domain where the brand

Table 3.9: Field-level extraction results

Craig's Cars: 4,400 posts			
Make	Recall	Prec.	F_1-Meas.
Single Pass (227)	79.19	85.30	82.13
Iterative (606)	79.31	84.30	81.73
ILA (580)	78.19	84.52	81.23
Model	Recall	Prec.	F_1-Meas.
Single Pass (227)	64.14	85.67	73.36
Iterative (606)	64.77	84.62	73.38
ILA (580)	64.25	82.79	72.35
Trim	Recall	Prec.	F_1-Meas.
Single Pass (227)	3.91	66.67	7.38
Iterative (606)	23.45	54.10	32.71
ILA (580)	23.45	52.17	32.35

Laptops: 2,400 posts			
Make	Recall	Prec.	F_1-Meas.
Single Pass (148)	49.63	50.77	50.19
Iterative (531)	51.27	46.22	48.61
ILA (295)	60.42	74.35	66.67
Model	Recall	Prec.	F_1-Meas.
Single Pass (148)	51.49	61.11	55.89
Iterative (531)	54.47	49.52	51.87
ILA (295)	61.91	76.18	68.31
Model Num.	Recall	Prec.	F_1-Meas.
Single Pass (148)	11.16	97.96	20.04
Iterative (531)	25.58	77.46	38.46
ILA (295)	27.91	81.08	41.52

Skis: 4,600 posts			
Brand	Recall	Prec.	F_1-Meas.
Single Pass (455)	53.69	50.58	52.09
Iterative (1392)	60.59	55.03	57.68
ILA (1392)	60.84	55.26	57.91
Model	Recall	Prec.	F_1-Meas.
Single Pass (455)	43.24	53.88	47.98
Iterative (1392)	51.86	51.25	51.55
ILA (1392)	51.33	48.93	50.10
Model Spec.	Recall	Prec.	F_1-Meas.
Single Pass (455)	20.22	78.99	32.19
Iterative (1392)	42.37	63.55	50.84
ILA (1392)	39.14	56.35	46.29

and model are clearly noisy in the iterative method, but cleaned up by using the locking mechanism (ILA).

Note that although the locking only seems to dramatically provide pruning for the Laptop domain, it also serves the very important function of letting us know how many posts to examine. So, even if it does not provide tremendous pruning (in some cases, such as the car domain the pruning is not needed), without this locking I would have no idea how many posts I need to examine in order to build the reference set.

My next set of experiments compare extraction results using the automatically constructed reference set (using the ILA algorithm) against both a manually constructed reference set using the ARX method of the previous chapter, and a supervised machine learning approach. By comparing to the ARX method using a manually constructed

reference set, I can show the benefits of automatically building the reference set for extraction, versus the tedious work of building it manually. Further, this justifies my claim that in many cases, building a reference set manually is not necessary since competitive results can be achieved by automatically building the reference set. Also, by comparing against supervised machine learning approaches I can show that the automatic method is competitive with expensive supervised methods, and in some cases, outperforms them due to the unstructured nature of the posts.

For the manually constructed reference sets, I built the reference sets from freely available data on the Web. For the Craig's Cars domain, I again use the reference set from Edmunds, outlined in the previous chapter. For the Laptops domain, I constructed a reference set of laptops using the online store Overstock.com. Collecting new laptops for sale provides an interesting reference set because while the set of laptop makers is static, the models and model numbers of the laptops that are for sale as new might not cover the used laptops for sale on Craigslist, which are generally older. Lastly, for the Skis domain I found a list of skis and ski apparel online from the website skis.com.[4] From this list, I parsed out the skis, and increased the reference set's utility by removing tokens in the attributes that could cause noisy extractions, such as the common token "Men's." The three manually constructed reference sets are described in Table 3.10.

For the machine learning comparison, I again use the CRF-Orth and CRF-Win implementations of Conditional Random Fields, described in the previous chapter. Again, it is important to note that CRF-Win in particular demonstrates how the unstructured nature of posts makes extraction difficult. I trained these extractors using 10% of the posts, since

[4]http://www.skis.com/affiliates/skis.xls

Table 3.10: Manually constructed reference-sets

Name	**Source**	**Attributes**	**Number of Tuples**
Cars	Edmunds and Super-Lamb Auto	make, model trim, year	27,006
Laptops	Overstock.com	manufacturer, model, model num.	279
Skis	Skis.com	brand, model, model spec.	213

this is a small enough amount to make it a fair comparison to automatic methods, but is also large enough such that the extractor can learn well. Also, since these are supervised methods, each was run 10 times and I report the averages. Table 3.11 shows the results comparing the ARX method using the ILA method to build the reference set, to these supervised methods and the ARX method using a manually constructed reference set (denoted by its source).

The results comparing the fully automatic method to the other methods are very encouraging. The fully automatic ILA method is competitive with the other methods across all of the domains. In fact, the ILA method's F_1-measure either outperforms or is within 10% of the F_1-measure for four out of the nine attributes as compared to both the CRF-Orth and manually constructed reference-set methods. This is the case for seven out of the nine attributes when compared against the CRF-Win. So, clearly the automatic method is competitive with previous appraoches that required a significant amount of human effort in either labeling and training a supervised method, or else in manually constructing a reference set to use for extraction.

Further, the fact that the CRF-Win method is often the worst performing method also points to the difficulty of using structure as an aid when performing extraction on posts. If the structure were very consistent, one would expect this method to greatly

outperform all of the other methods all of the time, since its key feature is the structural relation of the tokens.

Table 3.11: Comparing field-level results

Cars			
Make	Recall	Prec.	F_1-Meas.
ILA (580)	78.19	84.52	81.23
Edmunds (27,006)	92.51	99.52	95.68
CRF-Win (10%)	80.09	77.38	78.67
CRF-Orth (10%)	82.03	85.40	83.66
Model	Recall	Prec.	F_1-Meas.
ILA (580)	64.25	82.79	72.35
Edmunds (27,006)	79.50	91.86	85.23
CRF-Win (10%)	67.87	69.67	68.72
CRF-Orth (10%)	71.04	77.81	74.25
Trim	Recall	Prec.	F_1-Meas.
ILA (580)	23.45	52.17	32.35
Edmunds (27,006)	38.01	63.69	47.61
CRF-Win (10%)	38.77	39.35	38.75
CRF-Orth (10%)	47.94	47.90	47.88

Laptops			
Make	Recall	Prec.	F_1-Meas.
ILA (295)	60.42	74.35	66.67
Overstock (279)	84.41	95.59	89.65
CRF-Win (10%)	52.35	72.84	60.40
CRF-Orth (10%)	64.59	81.68	71.93
Model	Recall	Prec.	F_1-Meas.
ILA (295)	61.91	76.18	68.31
Overstock (279)	43.19	80.88	56.31
CRF-Win (10%)	44.51	68.32	53.54
CRF-Orth (10%)	58.07	78.29	66.54
Model Num.	Recall	Prec.	F_1-Meas.
ILA (295)	27.91	81.08	41.52
Overstock (279)	6.05	78.79	11.23
CRF-Win (10%)	42.15	65.22	50.66
CRF-Orth (10%)	53.18	68.58	59.48

Skis			
Brand	Recall	Prec.	F_1-Meas.
ILA (1392)	60.84	55.26	57.91
Skis.com (213)	83.62	87.05	85.30
CRF-Win (10%)	63.84	87.62	73.58
CRF-Orth (10%)	74.37	87.46	80.22
Model	Recall	Prec.	F_1-Meas.
ILA (1392)	51.33	48.93	50.10
Skis.com (213)	28.12	67.95	39.77
CRF-Win (10%)	51.08	74.83	60.15
CRF-Orth (10%)	61.36	73.15	66.59
Model Spec.	Recall	Prec.	F_1-Meas.
ILA (1392)	39.14	56.35	46.29
Skis.com (213)	18.28	59.44	27.96
CRF-Win (10%)	46.98	71.71	56.25
CRF-Orth (10%)	59.10	65.84	61.89

The lone attribute where the ILA method is greatly outperformed by the other methods is the brand attribute for Skis. While there are only 38 distinct brands to extract in the labeled data, the automatically built reference set contains 158 distinct brands. This discrepancy occurred for two reasons. First, some of the constructed "brands" were common words such as "Free" (as in Free Shipping). This causes problems because a post that says "ORG RETAIL: $925.00! FAST, FREE SHIPPING." will match the {Free, Shipping} tuple, creating a false positive. The other major reason that false brands are

put into the reference set happens because certain ski models were included without the appropriate brand (such as the "Mojo" ski which is pushed into the reference set without the "Head" brand). When this occurs, the correct reference set tuple is built, but the extracted attribute is given the incorrect label (i.e. the "Mojo" token is labeled as a brand, rather than a model). This confluence of circumstances led the ILA method to poor results. However, given that the ILA method is fully automatic, it is important to note that that this is the only attribute where ILA was greatly outperformed by all of the other methods.

Table 3.12: Comparing the ILA method

ILA vs. CRF-Win	
Outperforms	Within 10%
4/9	7/9

ILA vs. CRF-Ortho	
Outperforms	Within 10%
1/9	4/9

ILA vs. ARX	
Outperforms	Within 10%
4/9	4/9

It is also interesting to specifically examine the differences between the automatically constructed and manually constructed reference sets. The ARX method using manual reference-sets does a better job for the more general attributes, such as car makes and models, but for the more specific attributes (laptop model numbers, ski models, and ski model specifications) it is greatly outperformed by the automatic ILA method. What is particularly encouraging about this result is that these are the attributes that are very difficult to include in a manually constructed reference set because they are difficult to manually discover and it is not clear whether their coverage will hold. For example, it is

hard to enumerate the laptop model numbers and ski model specifications because there are so many, they are so varied, and they are hard to locate in a single place, so it requires substantial effort to construct such an inclusive reference set. Further, when comparing the ILA results to the Overstock results for the Laptop models and model numbers, it is clear that the coverage of the Overstock laptops (which are new) does not cover the posts (which are older, used laptops), since the results for these attributes using Overstock have such low recall. Table 3.12 summarizes the comparisons by showing the number of times the ILA method outperforms the given method and the number of times the ILA method's F_1-measure is within 10% of the given method.

3.3 Applicability of the ILA Method

This chapter presents an automatic algorithm for building reference sets from posts and then using them for extraction. However, the construction algorithm makes a few assumptions about the data it is processing. For one, the algorithm works well for hierarchical data. Although this is an enormous set of categories (e.g., all items for sale, descriptive categories such as geographical and person data, etc.) there are some categories that lack this characteristic (e.g. personal ads). More importantly, there are particular cases (as I will show) where the structure of the reference set attributes make it difficult to automatically construct a reference set. In this section I examine these cases and present a simple algorithm that can determine whether the fully automatic method can be used to build a reference set for extraction. If the system determines that it cannot use the automatic method of this chapter, then the user should construct a reference set manually, which

he or she can then exploit using the methods in this thesis. In this manner, the special cases requiring a manually constructed reference set can be determined easily.

3.3.1 Reference-Set Properties

There are certain properties of the underlying entities represented in the posts that allow my algorithm to construct the reference set well. Specifically, in the above experiments I show three domains (Cars, Laptops and Skis) where my algorithm performed well, even in some cases outperforming the state-of-the-art machine learning techniques. Yet, there are other domains where my technique has trouble building a reference set, and hence extracting the attributes. However, as stated above, if one is confronted with a domain that does not conform well to my automatic technique, a user can still tackle the problem by manually creating a reference set and using the extraction methods presented in this thesis.

To motivate this idea I present two domains where the automatic method does not construct a coherent reference set for extraction, but, given a manual reference set for these domains, the automatic, reference-set based extraction method outperforms the CRF-Win approach, which is supervised. The first domain, is the "BFT Posts" domain of the previous chapter. The second domain, called "eBay Comics," contains 776 eBay auction listings for Fantastic Four and Incredible Hulk comic books and comic-themed items. These domains are described in Table 3.13.

Table 3.13: Two domains where manual reference sets outperform

Name	**Source**	**Website**	**Attributes**	**Records**
BFT Posts	Bidding for Travel	www.biddingfortravel.com	star rating, area, name	1,125
eBay Comics	eBay comics: Fantastic Four & Incredible Hulk	www.ebay.com	title, issue number, description, publisher	776

These are particularly interesting domains because not only do they present problems for automatically building a reference set, these domains are also very unstructured and present problems to the CRF-Win method. Yet, as the results of Table 3.14 show, exploiting a manually constructed reference-set, using the ARX method, yields good results. For the BFT Posts domain, the reference set is the same 132 hotels described in the previous chapter as the "Hotels" reference set. For the eBay Comic domain, the reference set is the 918 tuples from the Comic Book Price Guide, called "Comics" in the previous chapter. To reiterate, each tuple of the Comics dataset contains a title, issue number, description (a one line of text such as "1st appearance of Wolverine!"), and publisher. As before, I ran CRF-Win 10 times, giving it 10% of the data for training, and present average values for the results using this method.

Table 3.14: Results on the BFT Posts and eBay Comics domains

eBay Comics			
Title	Recall	Prec.	F_1-Meas.
Comics	90.08	90.78	90.43
CRF-Win	77.69	77.85	77.77
ILA	30.93	38.83	34.43
Issue Num.	Recall	Prec.	F_1-Meas.
Comics	46.21	87.21	60.41
CRF-Win	80.67	80.52	80.59
ILA	8.40	22.55	12.24
Description	Recall	Prec.	F_1-Meas.
Comics	24.80	15.90	19.38
CRF-Win	6.08	14.56	8.78
ILA	0.00	0.00	0.00
Publisher	Recall	Prec.	F_1-Meas.
Comics	100.00	84.16	91.40
CRF-Win	38.42	86.84	52.48
ILA	0.00	0.00	0.00

BFT Posts			
Area	Recall	Prec.	F_1-Meas.
Hotels	62.23	85.36	71.98
CRF-Win	28.51	39.79	33.07
ILA	20.14	16.94	18.40
Star Rating	Recall	Prec.	F_1-Meas.
Hotels	83.94	99.44	91.03
CRF-Win	93.77	94.67	94.22
ILA	0.00	0.00	0.00
Name	Recall	Prec.	F_1-Meas.
Hotels	70.09	77.16	73.46
CRF-Win	39.84	42.95	41.33
ILA	0.47	1.12	0.66

The results validate that in some cases, a manually constructed reference set using the ARX method is the best option. CRF-Win only outperforms the other methods for two of the attributes (Hotel's star rating and Comic's issue number), and only the issue number

is dramatically different. Both the Hotel's star rating and the Comic's issue number have a numeric component, and since a number recognizer is one of the orthographic features used by CRF-Win, this explains the CRF method's high performance for those attributes. However, this is in stark contrast to the other 5/7 attributes where CRF-Win is drastically outperformed. The smallest difference between CRF-Win and ARX is roughly 11%, and for 3 of the 5 cases where CRF-Win is outperformed, it is done so by more than 30%. Clearly the data is unstructured (otherwise one would expect CRF-Win to do better). Yet, one can sucessfully perform extraction on this data without labeling training data (as in the CRF-Win) by manually constructing a reference set.

The results also indicate that the automatically constructed reference sets from the ILA approach did not work for these domains. Clearly the ILA method is not discovering useful reference set tuples. For some the attributes, such as the Hotel star rating and Comic description, the algorithm did not discover a single attribute, resulting in no extractions. Part of the issue for ILA is the small number of posts used for construction for each domain, but there are other properties of these data sets that make it difficult to automatically build reference sets from them.

Specifically, there are two factors that make it difficult to automatically construct reference sets from certain posts. The first is the proportion of junk tokens (i.e. those that should be ignored) as compared to attribute tokens (those that should be extracted). If the proportion of junk to attribute tokens is close to even, then it is hard to distinguish what is junk and what is not when generating attributes for the constructed reference set. Along these lines, if many of the attributes for a given reference set are multi-token terms, especially within a single tuple, then it is hard for the ILA algorithm to determine

where one attribute ends and the next begins, resulting in an ill defined reference set tuple. A good example of this is a hotel which may have a multi-token name ("Courtyard by Marriott") and a multi-token area ("Rancho Cordova"), back to back within a post, e.g. "2.5* Courtyard Marriott Rancho Cordova $43 4/14." Both of these problems are "boundary issues" where ILA gets confused as to which tokens belong to which attribute.

These boundary issues can be studied by creating a distribution of token characteristics that describes the textual properties of the posts and the relationships between the junk tokens and the attribute tokens. I define five distinct types of bigrams and define my distribution as the likelihood of each bigram type occurring in the set of posts. Table 3.15 defines these bigram types along with an example of the type (in bold) within the context of of a post that contains two attributes: a car make (Land Rover) and a car model (Discovery). Note that Table 3.15 also lists a name for each bigram type which I use to reference that type. Therefore, by comparing distributions of these five bigram types for the posts across various domains I can compare the characteristics of each domain and see if the boundary problems might occur.

Table 3.15: Bigram types that describe domain characteristics

$Attr_i$ | $Attr_j$ ("DIFF ATTR")	Two tokens from different attributes	... brand new Land **Rover Discovery** for ... ...
$Attr_i$ | $Attr_i$ ("SAME ATTR")	Two tokens from the same attribute	... brand new **Land Rover** Discovery for ... ...
$Attr_i$ | Junk ("ATTR JUNK")	A token followed by junk	... brand new Land Rover **Discovery for** ... ...
Junk | $Attr_i$ ("JUNK ATTR")	A junk token followed by an attribute	... brand **new Land** Rover Discovery for ... ...
Junk | Junk ("JUNK JUNK")	Two junk tokens	... **brand new** Land Rover Discovery for ... ...

The key concept of the bigram distributions are the distributions of the junk tokens and the tokens of the various attributes. However, this requires labeled data since the

bigram types distinguish between attribute-value tokens and junk tokens.[5] Yet, the ILA approach is automatic. If I want a mechanism by which the system can determine whether or not to automatically construct the reference set, then I do not want to burden a user with labeling large amounts of data to generate the bigram distributions. Note that deciding to automatically construct or not is a pre-processing step, and as such it is optional. Therefore, I do not think it is a problem to mix the single-label approach with the fully automatic ILA method, since one could use ILA, automatically, without using the approach described here.

So, to generate the labeled data for building the distributions in an efficient and simple manner, a user takes a *single post* from a domain, and labels it. Then, the algorithm finds all of the posts in the set that also contain the labeled tokens, and treats those tokens as labeled within the new post. By doing this bootstrapping I can generate sufficiently labeled data, and this is a process easy enough that I expect little burden on a user. One benefit of this approach is that it is robust. It is only used for estimating distributions, so I am not overly concerned about false labels, whereas I might be in a machine-learning context.

As an example, given the post "Honda Accord 2002 - $5000" I would label the make as Honda, the model as Accord and the year as 2002. Then, I would retrieve the following posts, with the labeled attributes bootstrapped from my single labeled example:

[5]Also, I make a distinction to define an attribute as only those values contained in a reference set. This means I do not define prices, dates, etc. as attributes in this context. I do this to define the distributions with respect to the reference set attributes only, since those are what I want ILA to discover.

Wanted Honda Accord Coupe {Make= Honda, Model=Accord}

2002 Honda Accord EX Sedan 4D {Make= Honda, Model=Accord, Year=2002}

2002 Accord for sale {Model=Accord, Year=2–2}

Accord 2002 must see!! {Model=Accord, Year=2002}

...

Once the algorithm has bootstrapped the labeled data, it can generate the distribution using the five bigram types. To test the bootstrapping, and to generate the distributions used below, I ran the bootstrapping method 20 times and generated the distributions for each of the five domains described in this chapter (BFT Posts, eBay Comics, Craig's Cars, Skis and Laptops). Table 3.3.1 shows the average number of labeled posts returned by bootstrapping a single post for each domain.[6]

Table 3.16: Number of bootstrapped posts

Domain	**Avg. # Bootstrapped Posts**
BFT Posts	20.95
Craig's Cars	10.90
eBay Comics	31.30
Laptops	65.75
Skis	16.35

Now that I have a mechanism to simply and efficiently generate the distributions, I use these distributions to compare the domains. In particular, given that the ILA technique is unable to discover useful reference sets from the BFT Posts and eBay Comics domains, I next show that these domains have similar distributions of the bigram types. Further, these distributions show that the boundary issues are indeed the impediment

[6]I did not allow the case where bootstrapping returns less than 1 post from the set, because that would mean you did not bootstrap at all, but only have the original labeled post. However, this can be easily avoided by first clustering the tokens and then simply picking a single post from the largest cluster.

to successfully building the reference sets. On the other hand, the distributions for the Craig's Cars, Skis and Laptop domains are different distributions from the BFT Posts and eBay Comics domains (and similar distributions to each other), which is expected since the automatic method works well in these cases. Figure 3.7 shows the distribution values for the different bigram types for each of the five domains averaged over the 20 trials. The x-axis of Figure 3.7 gives the percentage of the total distribution for each bigram type, and the y-axis shows each bigram type referenced by its name as given in Table 3.15.

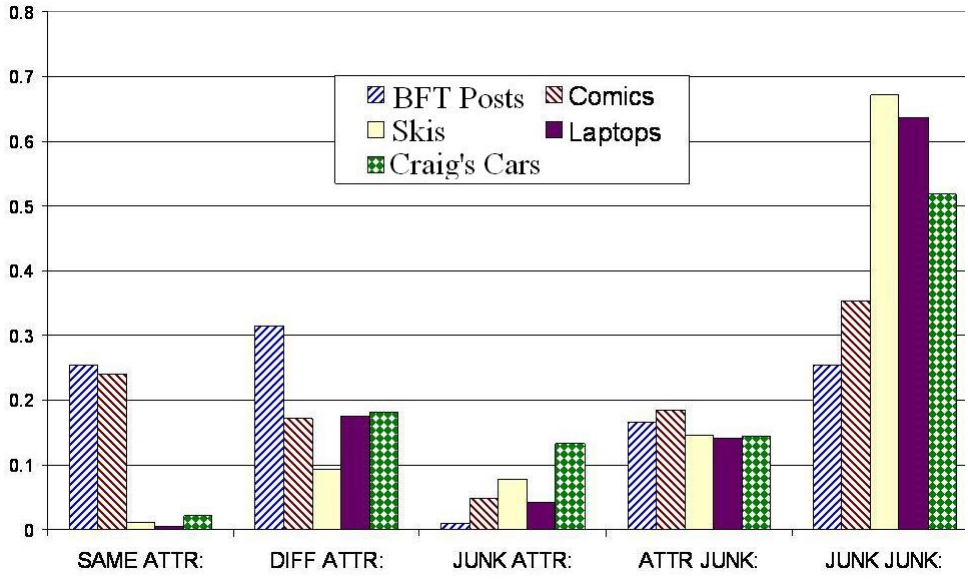

Figure 3.7: Average distribution values for each domain using 10 random posts

Figure 3.7 shows that the eBay Comics and BFT Posts domains have similar distributions to each other which are different from the rest of the domains. Further, both the eBay Comics and BFT Posts domains have similar values for the "SAME ATTR" bigram type and the "JUNK JUNK" type. Meanwhile, the other domains have very low values for the "SAME ATTR" bigram type and very high values for the "JUNK JUNK"

type. The fact that the BFT Posts and eBay Comics domains have comparative values for the "SAME ATTR" bigram type and the "JUNK JUNK" type shows that these two domains have many multi-token attributes from the same tuple, but there are almost as many junk tokens intermingled in the post which leads to boundary issues.

Specifically, according to Figure 3.7, in the BFT Posts and Comics domains about 25% of the bigrams in a post are from the same attribute. Couple this with the fact that almost the same number of bigrams represent two junk tokens ("JUNK JUNK" in the graph) or two tokens from a different attribute ("DIFF ATTR" in the graph), and it is clear to see that ILA gets confused as to what attribute a token should belong to, or whether that token is indeed an attribute instead of junk. Contrast this with the Skis, Laptops, and Craig's Cars domains, where a large majority of the tokens are junk, and those that are not are usually from different attributes (though not always), which is a case well suited for my automatic method. I must make note, however, that the Laptops and Craig's Cars domains have the same distribution for different attribute tokens as the Comics domain, which shows my technique is able to find multi-token attributes. It is more an issue of mixing the tokens of the many multi-token attributes that creates the boundary problem. For instance, in the Comics post "FANTASTIC FOUR ANUAL [*sic*] #2- DR.DOOM ORIGIN" it is hard to distinguish where one attribute ends and the next begins, given the many multi-token attributes.

To make the similarities between the Skis/Laptops/Cars and the BFT/Comics distributions more explicit than looking at the graph of Figure 3.7 I quantify the similarities mathematically. Since the graph compares probability distributions for the different bigram types, a natural way to quantify the comparison of the distributions is to use the

Kullback-Leibler (K-L) divergence, which measures the difference between two probability distributions. If one calls these distributions P and Q, then the Kullback-Leibler divergence is:

$$KL(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

If two distributions are similar, they have a low divergence. So, the BFT Posts and eBay Comics domains will have a low divergence when compared to each other using the average distributions. Further, I expect they will have a high divergence when compared to the other domains. However, K-L divergence is asymmetric (it only compares P to Q and not Q to P), so to make it symmetric I use a symmetric K-L divergence instead, which is defined as:

$$S_{KL}(P||Q) = \frac{KL(P||Q) + KL(Q||P)}{2}$$

Note that from this point forward, when I reference K-L divergence, I mean the symmetric version.

Table 3.14 shows the K-L divergence for all pairs of domains for the average distributions presented in Figure 3.7. The results for Table 3.14 are presented in ascending order of K-L divergence, which means that the pairs at the top of the table are the most similar. Clearly, from the table, the Laptops, Skis, and Craig's Cars domains are all similar to each other, and the BFT Posts and eBay Comics domains are similar to each other. Further, the members of the Laptops/Skis/Cars group are all dissimilar as compared to the BFT/Comics domains. These dissimilar pairs are at least three times greater than the similar values, showing there is a strong differentiation between the domains that are similar to each other and those that are not.

Table 3.17: K-L divergence between domains using average distributions

Domain	Domain	K-L divergence	
Laptops	Skis	0.06	
Craig's Cars	Skis	0.10	**Similar**
Craig's Cars	Laptops	0.11	
BFT Posts	eBay Comics	0.13	**Similar**
Craig's Cars	eBay Comics	0.49	
eBay Comics	Skis	0.71	
eBay Comics	Laptops	0.78	**Dissimlar**
BFT Posts	Craig's Cars	0.83	
BFT Posts	Laptops	1.01	
BFT Posts	Skis	1.14	

Up to this point, I have examined the average distributions. I now compare how many times across the 20 trials the distributions are similar to each other. This demonstrates a confidence level for each individual run in determining the similarity between the distributions. In this context, if the divergence is below a threshold (empirically chosen as 2 in this case, since the average KL-divergence for all pairs in all trials is 2.6), then I consider the distribution pair to match. I define the confidence as the percentage of trials that match out of all of the trials run.

Table 3.15 shows that indeed, the confidences hold well across individual trials. The confidence (percentage of matches) for the BFT/Comics pairing is high, as are the confidence measures for the pairs in the class of Laptops/Cars/Skis. However, the percentage of trials that matches across the BFT/Comics and Laptops/Cars/Skis domains are quite small.

Now, I have both a simple bootstrapping mechanism to generate the distributions, and a way to determine whether or not the ILA approach can automatically build the reference

Table 3.18: Percent of trials where domains match (under K-L thresh)

Domain	Domain	% Matches
Laptops	**Skis**	1
Skis	**Craig's Cars**	0.9
BFT Posts	**eBay Comics**	0.9
Laptops	**Craig's Cars**	0.8
Laptops	BFT Posts	0.45
Laptops	eBay Comics	0.45
Skis	BFT Posts	0.25
Skis	eBay Comics	0.25
BFT Posts	Craig's Cars	0.2
Craig's Cars	eBay Comics	0.2

set, since I have confidence in using a single run to determine the similarity of distributions to those that will work well for automatic construction (Skis/Laptops/Cars) versus those that require a manual reference set (BFT/Comics). Therefore, given an arbitrary set of posts, I should be able to label a single post, generate a distribution and then compare it to my five known distributions. If the average of the K-L divergence between the given distribution and the Comics and BFT distributions is less than a threshold (set to 2, as it is above), then the algorithm should suggest to the user to manually construct the reference set. If the average of the K-L divergence between the given distribution and the Skis, Laptops, and Craig's Cars domains is less than the threshold, the algorithm can automatically build a reference set and therefore should run the ILA algorithm. Lastly, if none of the averages are below the threshold, the algorithm should be conservative and suggest that the distribution is "unknown." Note that I choose to use the average because it gives a better similarity to the whole class of known distributions, rather than a single individual. I call this algorithm "Bootstrap-Compare," and it is given in Table 3.19.

Table 3.19: Method to determine whether to automatically construct a reference set.

```
BOOTSTRAP-COMPARE(POSTS P)
  p_s ∈ P ← labeled
  # Label the single post: p_s
  LabeldPosts P_L = BOOTSTRAPLABELS(p_s, P)
  Distribution D = GETBIGRAMTYPESDISTRIBUTION(P_L)
  If (Average(KL-DIVERGENCE(D, {COMICS, HOTELS})) < THRESH)
    Return "manual"
    # User should manually create reference set.
  Else If(Average KL-DIVERGENCE(D, {CARS, LAPTOPS, SKIS})) < THRESH)
    Run ILA
    # Reference set can be built automatically.
  Else
    Return "unknown"
    # No similar known sets.
```

I test the Bootstrap-Compare approach using two new domains: digital cameras for sale on eBay, and bibliographic citations from the Cora data set labeled for extraction. The digital cameras data set (called "Digicams") consists of 3,001 posts to eBay about cameras for sale each with a brand, a model, and a model number. The Cora data set contains 500 bibliographic citations. The Cora citations are an interesting set since I can imagine the case where a user just wants to extract (create a reference set of) the years, conference/journal names (called "booktitles"), and authors to see which authors publish where and when. Attributes such as the paper titles, conference locations, etc. can effectively be ignored as junk tokens. Descriptions of both of these domains are given in Table 3.20.

Table 3.20: Two domains for testing the Bootstrap-Compare method

Name	**Source**	**Website**	**Attributes**	**Records**
Digicams	eBay	www.ebay.com	brand, model, model num.	3,001
Cora	Cora	http://www.cs.umass.edu/ mccallum/ data/cora-ie.tar.gz	author, booktitle, year	500

First, I ran the ILA method on each of these domains, and then generated field-level extraction results using 100 labeled posts from the Digicams domain, and 500 posts

from the Cora domain. The results Table 3.21 show that the ILA method works well on the Digicams data, but poorly on the Cora domain. Therefore, the Bootstrap-Compare method should suggest it can use ILA for Digicams, but not for Cora.

Table 3.21: Extraction results using ILA on Digicams and Cora

Digicams			
Brand	Recall	Prec.	F_1-Meas.
	67.39	71.26	69.27
Model	Recall	Prec.	F_1-Meas.
	54.65	61.04	57.67
Model Num.	Recall	Prec.	F_1-Meas.
	20.37	26.19	22.92

Cora			
Author	Recall	Prec.	F_1-Meas.
	0.00	0.00	0.00
Booktitle	Recall	Prec.	F_1-Meas.
	0.00	0.00	0.00
Year	Recall	Prec.	F_1-Meas.
	0.20	2.22	0.37

To test the effectiveness of the Bootstrap-Compare method, I ran it 20 times for each domain and record whether it suggests it can automatically construct the reference set, or not. For the Digicams domain, the Bootstrap-Compare algorithm correctly identified that a reference set can be automatically constructed for 18/20 (90%) of the trials. This gives us a 90% confidence that Bootstrap-Compare can identify this situation. For the Cora domain, the algorithm suggested the user manually create a reference set in 20/20 (100%) of the trials. From these results I see the expected behavior of Bootstrap-Compare, namely, it suggests to automatically construct the reference set for the Digicams domain, but not for the Cora domain. Although this is an optional step for constructing reference sets, especially since it requires a single labeled post, it can nonetheless help a user in deciding whether to use the ILA method for building their reference set or not.

Chapter 4

A Machine Learning Approach to Reference-Set Based Extraction

Although Chapters 2 and 3 present an automatic approach to extraction that covers the cases for both manually constructed and automatically constructed reference sets, there is still a special case to consider where an automatic approach to extraction would not be as suitable. In particular, there are situations where a user requires highly accurate extraction of attributes. This includes the high-accuracy extraction of attributes that are not easily represented in a reference set, such as prices or dates. I call these "common attributes." Common attributes are usually an infinitely large set, but they contain certain characteristics that can be exploited for extraction. For instance, regular expressions can identify possible prices.

However, mixing the extraction of reference set attributes and common attributes can lead to ambiguous extractions. For instance, while a regular expression for prices might pull out the token "626" from a classified ad for cars, it might also be the case that the "626" in the post refers to the car model, since the Mazda company makes a car called the 626.

Another issue for high-accurate extraction stems from the fact that the automatic methods can not explicitly model true positives from false negatives. Since the automatic method only considers attributes in agreement to deal with possible false positives, it may leave out certain extractions for attributes because it could not differentiate which attribute from the reference set is better. Therefore the automatic method may lower its recall because of the above effect.

To handle both the issue of ambiguities and attribute-in-agreement issue, this chapter presents an extraction algorithm that exploits reference sets using techniques from machine learning. While the method of this chapter yields high accuracy extractions, including "common attribute" extractions, it does so at the cost of manually labeling data for both matching and extraction. However, by using a reference set to aid disambiguation, the system handles common attribute extraction from posts with high accuracy. Further, by explicitly modeling matches versus non-matches and correct extractions versus incorrect extractions, the system can learn how to distinguish true positives from false negatives, without resorting to using only the attributes-in-agreement from the matches. This leads to a higher level of recall for the non-common attributes. Note, however, that the same reference-set based extraction framework holds for the algorithm of this chapter. The goal is still to match to a reference set and then exploit those matches for extraction. The key difference in this chapter is that the matching and extraction steps are done using techniques from machine learning that yield very high accuracy extractions, particularly for the common attributes.

4.1 Aligning Posts to a Reference Set

As in the general reference-set based extraction framework, the algorithm needs to first decide which members of the reference set best matches the posts. In the context of our machine learning approach, this matching is known as record linkage [26]. Record linkage can be broken into two steps: generating candidate matches, called "blocking"; and then separating the true matches from these candidates in the "matching" step. Blocking is a necessary step since using a machine learning component for matching involves a more costly matching decision than in our automatic method which uses vector-based similarity matching, so the number of candidate matches needs to be minimized to mitigate this cost.

In our approach, the blocking generates candidate matches based on similarity methods over certain attributes from the reference set as they compare to the posts. For our cars example, the algorithm may determine that it can generate candidates by finding common tokens between the posts and the make attribute of the reference set. That is, it may find that all posts with "Honda" in them are good candidates for detailed examination during the matching step with the "Honda" tuples of the reference set. This step is detailed in Section 4.1.1 and is crucial in limiting the number of candidates matches we later examine during the matching step. After generating candidates, the algorithm generates a large set of features between each post and its candidate matches from the reference set. Using these features, the algorithm employs machine learning methods to separate the true matches from the false positives generated during blocking. This matching is detailed in Section 4.1.2.

4.1.1 Generating Candidates by Learning Blocking Schemes for Record Linkage

It is infeasible to compare each post to all of the members of a reference set since matching using machine learning is more costly than our automatic matching technique. Therefore a preprocessing step generates candidate matches by comparing all the records between the sets using fast, approximate methods. This is called *blocking* because it can be thought of as partitioning the full cross product of record comparisons into mutually exclusive blocks [50]. That is, to block on an attribute, first an algorithm sorts or clusters the data sets by the attribute. Then it applies the comparison method to only a single member of a block. After blocking, the candidate matches (from the matching block) are examined in detail to discover true matches.

There are two main goals of blocking. First, blocking should limit the number of candidate matches, which limits the number of expensive, detailed comparisons needed during record linkage. Second, blocking should not exclude any true matches from the set of candidate matches. This means there is a trade-off between finding all matching records and limiting the size of the candidate matches. So, the overall goal of blocking is to make the matching step more scalable, by limiting the number of comparisons it must make, while not hindering its accuracy by passing as many true matches to it as possible.

Most blocking is done using the *multi-pass approach* [29], which combines the candidates generated during independent runs. For example, with cars data, a blocking method might make one pass over the data blocking on tokens in the car model, while another run might block using tokens of the make along with common tokens in the

trim values. One can view the multi-pass approach as a rule in disjunctive normal form, where each conjunction in the rule defines each run, and the union of these rules combines the candidates generated during each run. In the context of blocking, I call this disjunctive rule a "blocking scheme." Using my care example, the rule might become ({*token-match, model*} $\wedge$ ({*token-match, year*}) $\cup$ ({*token-match, make*})). The effectiveness of the multi-pass approach hinges upon which methods and attributes are chosen in the conjunctions.

Note that each conjunction is a set of {*method, attribute*} pairs, and I do not make restrictions on which methods can be used. The set of methods could include full string metrics such as cosine similarity, simple common token matching as outlined above, or even state-of-the-art n-gram methods as shown in my experiments. The key for methods is not necessarily choosing the fastest (though I show how to account for the method speed below), but rather choosing the methods that will generate the smallest set of candidate matches that still cover the true positives, since it is the matching step that will consume the most time.

Therefore, a blocking scheme should include enough conjunctions to cover as many true matches as it can. For example, the first conjunct might not cover all of the true matches if the datasets being compared do not overlap in all of the years, so the second conjunct can cover the rest of the true matches. This is the same as adding more independent runs to the multi-pass approach.

However, since a blocking scheme includes as many conjunctions as it needs, these conjunctions should limit the number of candidates they generate. For example, the second conjunct is going to generate a lot of unnecessary candidates since it will return

all records that share the same make. By adding more {*method, attribute*} pairs to a conjunction, the blocking method limits the number of candidates it generates. For example, if I change ({*token-match, make*}) to ({*token-match, make*} $\wedge$ {*token-match, trim*}) the rule still covers new true matches, but it generates fewer additional candidates.

Therefore effective blocking schemes should learn conjunctions that minimize the false positives, but learn enough of these conjunctions to cover as many true matches as possible. These two goals of blocking can be clearly defined by the Reduction Ratio and Pairs Completeness [24].

The *Reduction Ratio* (RR) quantifies how well the current blocking scheme minimizes the number of candidates. Let C be the number of candidate matches and N be the size of the cross product between both data sets.

$$RR = 1 - C/N$$

It should be clear that adding more {*method,attribute*} pairs to a conjunction increases its RR, as when I changed ({*token-match, zip*}) to ({*token-match, zip*} $\wedge$ {*token-match, first name*}). Pairs Completeness (PC) measures the coverage of true positives, i.e., how many of the true matches are in the candidate set versus those in the entire set. If S_m is the number of true matches in the candidate set, and N_m is the number of matches in the entire dataset, then:

$$PC = S_m/N_m$$

Adding more disjuncts can increase the PC. For example, I added the second conjunction to the example blocking scheme because the first did not cover all of the matches.

The blocking approach in this paper, "Blocking Scheme Learner" (BSL), learns effective blocking schemes in disjunctive normal form by maximizing the reduction ratio and pairs completeness. In this way, BSL tries to maximize the two goals of blocking. Previous work showed BSL aided the scalability of record linkage [45], and in this chapter I extend that idea by showing that it also can work in the case of matching posts to the reference set records.

The BSL algorithm uses a modified version of the Sequential Covering Algorithm (SCA), used to discover disjunctive sets of rules from labeled training data [47]. In my case, SCA will learn disjunctive sets of conjunctions consisting of {method, attribute} pairs. Basically, each call to *LEARN-ONE-RULE* generates a conjunction, and BSL keeps iterating over this call, covering the true matches left over after each iteration. This way SCA learns a full blocking scheme. The BSL algorithm is shown in Table 4.1.

Table 4.1: Modified Sequential Covering Algorithm

SEQUENTIAL-COVERING(*class*, *attributes*, *examples*)
LearnedRules ← {}
Rule ← *LEARN-ONE-RULE*(*class*, *attributes*, *examples*)
While examples left to cover, do
LearnedRules ← *LearnedRules* ∪ *Rule*
Examples ← *Examples* - {*Examples covered by Rule*}
Rule ← *LEARN-ONE-RULE*(*class*, *attributes*, *examples*)
If Rule contains any previously learned rules, remove these contained rules.
Return LearnedRules

There are two modifications to the classic SCA algorithm, which are shown in bold. First, BSL runs until there are no more examples left to cover, rather than stopping at some threshold. This ensures that the algorithm maximizes the number of true matches generated as candidates by the final blocking rule (Pairs Completeness). Note that this might, in turn, yield a large number of candidates, hurting the Reduction Ratio. However,

omitting true matches directly affects the accuracy of record linkage, and blocking is a preprocessing step for record linkage, so it is more important to cover as many true matches as possible. This way BSL fulfills one of the blocking goals: not eliminating true matches if possible. Second, if BSL learns a new conjunction (in the *LEARN-ONE-RULE* step) and its current blocking scheme has a rule that already contains the newly learned rule, then BSL removes the rule containing the newly learned rule. This is an optimization that allows BSL to check rule containment as its progresses, rather than at the end.

Checking rule containment is possible because BSL can guarantee that it learns less restrictive rules as it goes. I prove this guarantee as follows, using proof by contradiction. Assume two attributes A and B, and a method X. Also, assume that BSL previously learned rules contain the following conjunction, ($\{X, A\}$) and BSL currently learned the rule ($\{X, A\} \wedge \{X, B\}$). That is, assume the learned rules contains a rule that is less specific than the currently learned rule. If this were the case, then there must be at least one training example covered by ($\{X, A\} \wedge \{X, B\}$) that is not covered by ($\{X, A\}$), since SCA dictates that BSL remove all examples covered by ($\{X, A\}$) when it is learned. Clearly, this cannot happen, since any examples covered by the more specific ($\{X, A\} \wedge \{X, B\}$) would have been covered by ($\{X, A\}$) already and removed, which means BSL could not have learned the rule ($\{X, A\} \wedge \{X, B\}$). Thus, there exists a contradiction.

As I stated before, the two main goals of blocking are to minimize the size of the candidate set, while not removing any true matches from this set. I have already mentioned how BSL maximizes the number of true positives in the candidate set and now I describe how BSL minimizes the overall size of the candidate set, which yields more

scalable record linkage. To minimize the candidate set's size, BSL learns as restrictive a conjunction as possible during each call to *LEARN-ONE-RULE* during the SCA. I define restrictive as minimizing the number of candidates generated, as long as a certain number of true matches are still covered. (Without this restriction, BSL could learn conjunctions that perfectly minimize the number of candidates: they simply return none.)

To do this, the *LEARN-ONE-RULE* step performs a general-to-specific beam search. It starts with an empty conjunction and at each step adds the {method, attribute} pair that yields the smallest set of candidates that still cover at least a set number of true matches. That is, BSL learns the conjunction that maximizes the Reduction Ratio, while at the same time covering a minimum value of Pairs Completeness. BSL uses a beam search to allow for some backtracking, since the search is greedy. However, since the beam search goes from general-to-specific, it ensures that the final rule is as restrictive as possible. The full *LEARN-ONE-RULE* is given in Table 4.2.

The constraint that a conjunction covers a minimum PC ensures that the learned conjunction does not over-fit to the data. Without this restriction, it would be possible

Table 4.2: Learning a conjunction of {method, attribute} pairs

LEARN-ONE-RULE($attributes, examples, min\_thresh, k$)

Best-Conjunction $\leftarrow$ {}
Candidate-conjunctions $\leftarrow$ *all {method, attribute} pairs*
While Candidate-conjunctions not empty, do
  For each ch $\in$ *Candidate-conjunctions*
    If not first iteration
      ch $\leftarrow$ *ch* $\cup$ {method,attribute}
    Remove any ch that are duplicates, inconsistent or not max. specific
    if REDUCTION-RATIO(ch) > REDUCTION-RATIO(Best-Conjunction)
    and PAIRS-COMPLETENESS(ch) $\geq$ *min_thresh*
      Best-Conjunction $\leftarrow$ *ch*
  Candidate-conjunctions $\leftarrow$ *best k members of Candidate-conjunctions*
return Best-conjunction

for *LEARN-ONE-RULE* to learn a conjunction that returns no candidates, uselessly producing an optimal RR.

The algorithm's behavior is well defined for the minimum PC threshold. Consider, the case where the algorithm learns the most restrictive rule it can with the minimum coverage. In this case, the parameter ends up partitioning the space of the cross product of example records by the threshold amount. That is, if we set the threshold amount to 50% of the examples covered, the most restrictive first rule covers 50% of the examples. The next rule covers 50% of what is remaining, which is 25% of the examples. The next will cover 12.5% of the examples, etc. In this sense, the parameter is well defined. If a user sets the threshold high, BSL learns fewer, less restrictive conjunctions, possibly limiting the RR, although this may increase PC slightly. If the threshold is set lower, BSL covers more examples, but it needs to learn more conjuncts. These newer conjuncts, in turn, may be subsumed by later conjuncts, so they will be a waste of time to learn. So, as long as this parameter is small enough, it should not affect the coverage of the final blocking scheme, and smaller than that just slows down the learning. I set this parameter to 50% for my experiments[1].

Now I analyze the running time of BSL, and show how BSL can take into account the running time of different blocking methods, if need be. Assume that I have x (*method*, *attribute*) pairs such as (*token*, $first - name$). Now, assume that the beam size is b, since I use general-to-specific beam-search in the Learn-One-Rule procedure. Also, for the time being, assume each (method, attribute) pair can generate its blocking candidates in $O(1)$ time. (I relax this assumption later.) Each time BSL hits the Learn-One-Rule

[1]Setting this parameter lower than 50% had an insignificant effect on my results, and setting it much higher, to 90%, only increased the PC by a small amount (if at all), while decreasing the RR.

phase, it will try all rules in the beam with all of the (attribute, method) pairs not in the current beam rules. So, in the worst case, this takes $O(bx)$ each time, since for each (method, attribute) pair in the beam, BSL tries it against all other (method, attribute) pairs. Now, in the worst case, each learned disjunct would only cover 1 training example, so the final rule is a disjunction of all pairs x. Therefore, BSL runs the Learn-One-Rule x times, resulting in a learning time of $O(bx^2)$. If BSL has e training examples, the full training time is $O(ebx^2)$, for BSL to learn the blocking scheme.

Now, while I assumed above that each (method, attribute) runs in $O(1)$ time, this is clearly not the case, since there is a substantial amount of literature on blocking methods and further the blocking times can vary significantly [5]. I define a function $t_x(e)$ that represents how long it takes for a single (method, attribute) pair in x to generate the e candidates in the training example. Using this notation, Learn-One-Rule's time becomes $O(b(xt_x(e)))$ (it runs $t_x(e)$ time for each pair in x) and so the full training time becomes $O(eb(xt_x(e))^2)$. Clearly the most expensive blocking methodology dominates such a running time. Once a rule is learned, it is bounded by the time it takes to run the rule and (method, attribute) pairs involved, so it takes $O(xt_x(n))$, where n is the number of records for classification.

From a practical standpoint, I can easily modify BSL to account for the time it takes certain blocking methods to generate their candidates. In the Learn-One-Rule step, I change the performance metric to reflect both Reduction Ratio and blocking time as a weighted average. That is, given W_{rr} as the weight for Reduction Ratio and W_b as the weight for the blocking time, I modify Learn-One-Rule to maximize the performance of any disjunct based on this weighted average. Table 4.3 shows the modified version of

Learn-One-Rule, and the changes are shown in **bold**.

Table 4.3: Learning a conjunction of {method, attribute} pairs using weights

LEARN-ONE-RULE$(attributes, examples, min\_thresh, k)$

Best-Conj $\leftarrow$ {}
Candidate-conjunctions $\leftarrow$ *all* {*method, attribute*} *pairs*
While Candidate-conjunctions not empty, do
  For each ch $\in$ *Candidate-conjunctions*
    If not first iteration
      $ch \leftarrow ch \cup$ {method,attribute}
    Remove any ch that are duplicates, inconsistent or not max. specific
    SCORE(ch) $= W_{rr}*$***REDUCTION-RATIO(ch)***$+W_b*$***BLOCK-TIME(ch)***
    SCORE(Best-Conj) $= W_{rr}*$***REDUCTION-RATIO(Best-conj)***$+W_b*$***BLOCK-TIME(Best-conj)***
    if SCORE(ch) $>$ ***SCORE(Best-conj)***
    and PAIRS-COMPLETENESS(ch) $\geq$ *min_thresh*
      Best-conj $\leftarrow$ *ch*
  Candidate-conjunctions $\leftarrow$ *best k members of Candidate-conjunctions*
return Best-conj

Note that when W_b is set to 0, the above version of Learn-One-Rule resolves into using the same version of Learn-One-Rule as used throughout this chapter which only considers the Reduction Ratio. Since the methods (token and n-gram match) are simple to compute, requiring more time to build the initial index than to do the candidate generation, one can safely set W_b to 0. Also, making this trade-off of time versus reduction might not always be an appropriate decision. Although a method may be fast, if it does not sufficiently reduce the reduction ratio, then the time it takes the record linkage step might increase more than the time it would have taken to run the blocking using a method that provides a larger increase in reduction ratio. Since classification often takes much longer than candidate generation, the goal should be to minimize candidates (maximize reduction ratio), which in turn minimizes classification time. Further, the key insight of BSL is not only that it chooses the blocking method, but more importantly that it chooses the appropriate attributes to block on. In this sense, BSL is more like a feature selection

algorithm than a blocking method. As I show in our experiments, for blocking it is more important to pick the right attribute combinations, as BSL does, even using simple methods, than to do blocking using the most sophisticated methods.

Although BSL was originally conceived for blocking in traditional record linkage, I can easily extend the BSL algorithm to handle the case of matching posts to members of the reference set. This is a special case because the posts have all the attributes embedded within them while the reference set data is relational and structured into schema elements. To handle this special case, rather than matching attribute and method pairs across the data sources during LEARN-ONE-RULE, BSL instead compares attribute and method pairs from the relational data to the entire post. This is a small change, showing that the same algorithm works well even in this special case.

Once BSL learns a good blocking scheme, it can now efficiently generate candidates from the post set to align to the reference set. This blocking step is essential for mapping large amounts of unstructured and ungrammatical data sources to larger and larger reference sets.

4.1.2 The Matching Step

From the set of candidates generated during blocking one can find the member of the reference set that best matches the current post. As mentioned throughout this thesis, reference-set based extraction breaks into two larger steps: first, finding the matches between the posts and reference set, and second, using those matches to aid extraction. By using machine learning for this "matching step," this alignment procedure can be referred to as record linkage [26]. However, the record linkage problem presented in

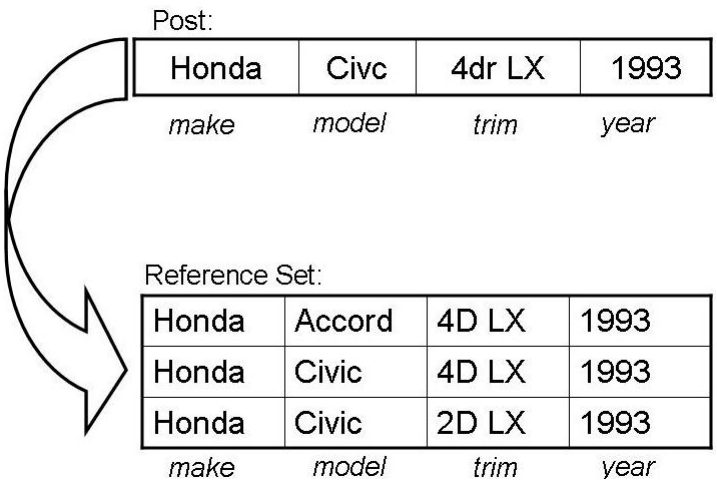

Figure 4.1: The traditional record linkage problem

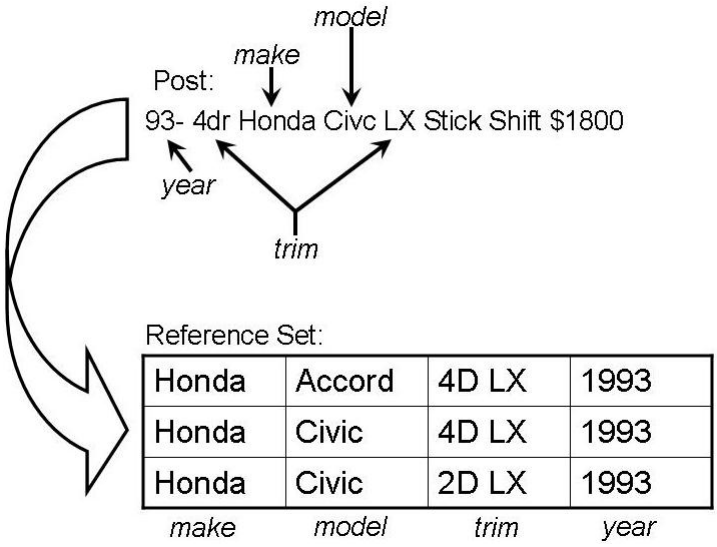

Figure 4.2: The problem of matching a post to the reference set

this chapter differs from the "traditional" record linkage problem and is not well studied. Traditional record linkage matches a record from one data source to a record from another data source by relating their respective, decomposed attributes. For instance, Figure 4.1 shows how traditional record linkage might match two sets of cars, each with make, model, trim and year attributes. Note that each attribute is decomposed from the other

attributes and the matches are determined by comparing each attribute from one set to the other set, i.e. comparing the makes, models, etc. Yet, the attributes of the posts are embedded within a single piece of text and *not yet identified*. This text is compared to the reference set, which is already decomposed into attributes and which does not have the extraneous tokens present in the post. Figure 4.2 depicts this problem. With this type of matching traditional record linkage approaches do not apply.

Instead, the matching step compares the post to all of the attributes of the reference set concatenated together. Since the post is compared to a whole record from the reference set (in the sense that it has all of the attributes), this comparison is at the "record level" and it approximately reflects how similar all of the embedded attributes of the post are to all of the attributes of the candidate match. This mimics the idea of traditional record linkage, that comparing all of the fields determines the similarity at the record level.

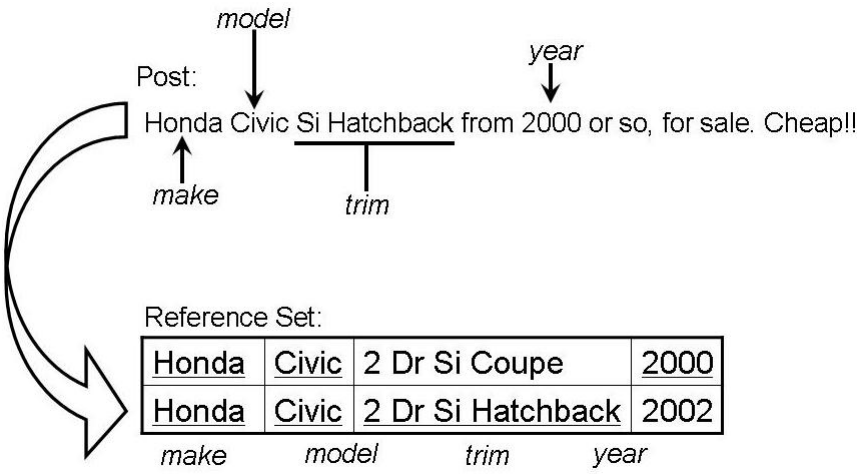

Figure 4.3: Two records with equal record level but different field level similarities

However, by using only the record level similarity it is possible for two candidate matches to generate the same record level similarity while differing on individual attributes. If one of these attributes is more discriminative than the other, there needs to be some way to reflect that. For example, consider Figure 4.3. In the figure, the two candidates share the same make and model. However, the first candidate shares the year while the second candidate shares the trim. Since both candidates share the same make and model, and both have another attribute in common, it is possible that they generate the same record level comparison. Yet, a trim on car, especially with a rare thing like a "Hatchback" should be more discriminative than sharing a year, since there are lots of cars with the same make, model and year, that differ only by the trim. This difference in individual attributes needs to be reflected.

To discriminate between attributes, the matching step borrows the idea from traditional record linkage that incorporating the individual comparisons between each attribute from each data source is the best way to determine a match. That is, just the record level information is not enough to discriminate matches, field level comparisons must be exploited as well. To do "field level" comparisons the matching step compares the post to each individual attribute of the reference set.

These record and field level comparisons are represented by a vector of different similarity functions called *RL_scores*. By incorporating different similarity functions, *RL_scores* reflects the different types of similarity that exist between text. Hence, for the record level comparison, the matching step generates the *RL_scores* vector between the post and all of the attributes concatenated. To generate field level comparisons, the matching step calculates the *RL_scores* between the post and each of the individual

attributes of the reference set. All of these *RL_scores* vectors are then stored in a vector called V_{RL}. Once populated, V_{RL} represents the record and field level similarities between a post and a member of the reference set.

In the example reference set from Figure 4.2, the schema has 4 attributes <*make, model, trim, year*>. Assuming the current candidate is <"Honda", "Civic", "4D LX", "1993">, then the V_{RL} looks like:

$$
\begin{aligned}
V_{RL} = <& RL\_scores(post,\ \text{“Honda”}),\\
& RL\_scores(post,\ \text{“Civic”}),\\
& RL\_scores(post,\ \text{“4D LX”}),\\
& RL\_scores(post,\ \text{“1993”}),\\
& RL\_scores(post,\ \text{“Honda Civic 4D LX 1993”})>
\end{aligned}
$$

Or more generally:

$$
\begin{aligned}
V_{RL} = <& RL\_scores(post,\ attribute_1),\\
& RL\_scores(post,\ attribute_2),\\
& \ldots,\\
& RL\_scores(post,\ attribute_n),\\
& RL\_scores(post,\ attribute_1\ attribute_2\ \ldots attribute_n)>
\end{aligned}
$$

The *RL_scores* vector is meant to include notions of the many ways that exist to define the similarity between the textual values of the data sources. It might be the case that one attribute differs from another in a few misplaced, missing or changed letters. This sort of similarity identifies two attributes that are similar, but misspelled, and is called "edit distance." Another type of textual similarity looks at the tokens of the attributes and defines similarity based upon the number of tokens shared between the attributes.

This "token level" similarity is not robust to spelling mistakes, but it puts no emphasis on the order of the tokens, whereas edit distance requires that the order of the tokens match in order for the attributes to be similar. Lastly, there are cases where one attribute may sound like another, even if they are both spelled differently, or one attribute may share a common root word with another attribute, which implies a "stemmed" similarity. These last two examples are neither token nor edit distance based similarities.

To capture all these different similarity types, the *RL_scores* vector is built of three vectors that reflect the each of the different similarity types discussed above. Hence, *RL_scores* is:

RL_scores(post, attribute)=<token_scores(post, attribute),
edit_scores(post, attribute),
other_scores(post, attribute)>

The vector *token_scores* comprises three token level similarity scores. Two similarity scores included in this vector are based on the Jensen-Shannon distance, which defines similarities over probability distributions of the tokens. One uses a Dirichlet prior [17] and the other smooths its token probabilities using a Jelenik-Mercer mixture model [64]. The last metric in the *token_scores* vector is the Jaccard similarity.

With all of the scores included, the *token_scores* vector takes the form:

token_scores(post, attribute)=<Jensen-Shannon-Dirichlet(post, attribute),
Jensen-Shannon-JM-Mixture(post, attribute),
Jaccard(post, attribute)>

The vector *edit_scores* consists of the edit distance scores which are comparisons between strings at the character level defined by operations that turn one string into another. For instance, the *edit_scores* vector includes the Levenshtein distance [36], which returns the minimum number of operations to turn string S into string T, and the Smith-Waterman distance [56] which is an extension to the Levenshtein distance. The last score in the vector *edit_scores* is the Jaro-Winkler similarity [63], which is an extension of the Jaro metric [30] used to find similar proper nouns. While not a strict edit-distance, because it does not regard operations of transformations, the Jaro-Winkler metric is a useful determinant of string similarity.

With all of the character level metrics, the *edit_scores* vector is defined as:

edit_scores(post, attribute)=<Levenshtein(post, attribute),
Smith-Waterman(post, attribute),
Jaro-Winkler(post, attribute)>

All the similarities in the *edit_scores* and *token_scores* vector are defined in the SecondString package [17] which was used for the experimental implementation as described in Section 4.3.

Lastly, the vector *other_scores* captures the two types of similarity that did not fit into either the token level or edit distance similarity vector. The first is the Soundex score between the post and the attribute. Soundex uses the phonetics of a token as a basis for determining the similarity. That is, misspelled words that sound the same will receive a high Soundex score for similarity. The other similarity is based upon the Porter stemming algorithm [52], which removes the suffixes from strings so that the root words can be compared for similarity. This helps alleviate possible errors introduced by

the prefix assumption introduced by the Jaro-Winkler metric, since the stems are scored rather than the prefixes. Including both of these scores, the *other_scores* vector becomes:

other_scores(post, attribute)=<Porter-Stemmer(post, attribute),

Soundex(post, attribute)>

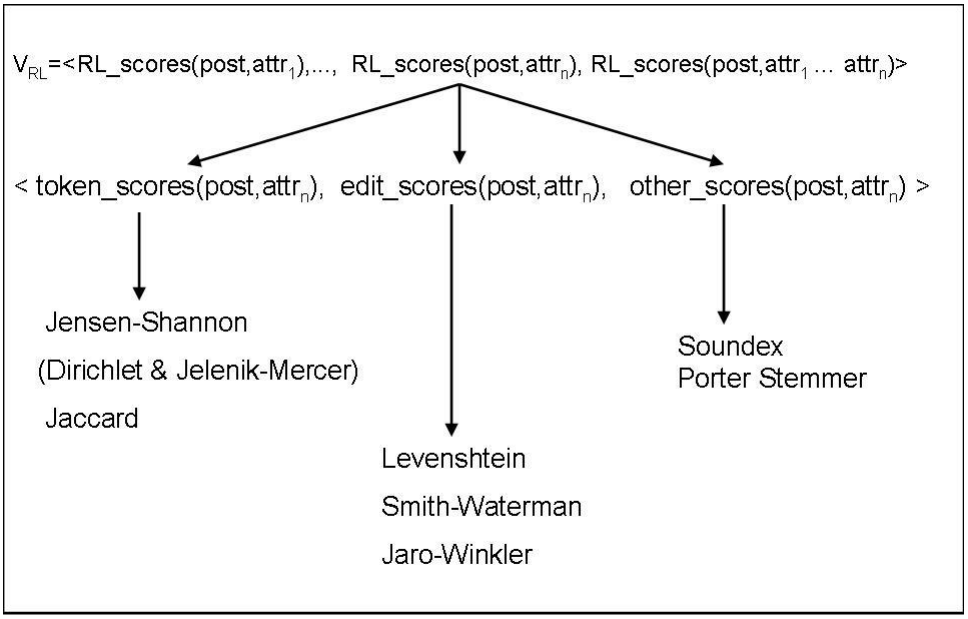

Figure 4.4: The full vector of similarity scores used for record linkage

Figure 4.4 shows the full composition of V_{RL}, with all the constituent similarity scores.

Once a V_{RL} is constructed for each of the candidate matches, the matching step then performs a binary rescoring on each V_{RL} to further help determine the best match amongst the candidates. This rescoring helps determine the best possible match for the post by separating out the best candidate as much as possible. Because there might be a few candidates with similarly close values, and only one of them is a best match, the rescoring emphasizes the best match by downgrading the close matches so that they have

the same element values as the more obvious non-matches, while boosting the difference in score with the best candidate's elements.

To rescore the vectors of candidate set C, the rescoring method iterates through the elements x_i of all $V_{RL} \in C$, and the V_{RL}(s) that contain the maximum value for x_i map this x_i to 1, while all of the other V_{RL}(s) map x_i to 0. Mathematically, the rescoring method is:

$$\forall V_{RL_j} \in C, j = 0...|C|$$

$$\forall x_i \in V_{RL_j}, i = 0...\left|V_{RL_j}\right|$$

$$f(x_i, V_{RL_j}) = \begin{cases} 1, x_i = \max(\forall x_t \in V_{RL_s}, V_{RL_s} \in C, t = i, s = 0...|C|) \\ 0, otherwise \end{cases}$$

For example, suppose C contains 2 candidates, V_{RL_1} and V_{RL_2}:

$$V_{RL_1} = <\{.999,...,1.2\},...,\{0.45,...,0.22\}>$$

$$V_{RL_2} = <\{.888,...,0.0\},...,\{0.65,...,0.22\}>$$

After rescoring they become:

$$V_{RL_1} = <\{1,...,1\},...,\{0,...,1\}>$$

$$V_{RL_2} = <\{0,...,0\},...,\{1,...,1\}>$$

After rescoring, the matching step passes each V_{RL} to a Support Vector Machine (SVM) [31] trained to label them as matches or non-matches. The best match is the candidate that the SVM classifies as a match, with the maximally positive score for the decision function. If more than one candidate share the same maximum score from the decision function, then they are thrown out as matches. This enforces a strict 1-1 mapping between posts and members of the reference set. However, a 1-n relationship

can be captured by relaxing this restriction. To do this the algorithm keeps either the first candidate with the maximal decision score, or chooses one randomly from the set of candidates with the maximum decision score.

Although I use SVMs in this paper to differentiate matches from non-matches, the algorithm is not strictly tied to this method. The main characteristics for my learning problem are that the feature vectors are sparse (because of the binary rescoring) and the concepts are dense (since many useful features may be needed and thus none should be pruned by feature selection). I also tried to use a Naïve Bayes classifier for the matching task, but it was monumentally overwhelmed by the number of features and the number of training examples. Yet this is not to say that other methods that can deal with sparse feature vectors and dense concepts, such as online logistic regression or boosting, could not be used in place of SVM.

After the match for a post is found, the attributes of the matching reference set member are added as annotation to the post by including the values of the reference set attributes with tags that reflect the schema of the reference set. The overall matching algorithm is shown in Figure 4.5.

4.2 Extracting Data from Posts

As stated, the motivation of the algorithm of this chapter is a high-accuracy extraction algorithm that can extract both attributes from a reference set, and those that are not easily represented by a reference set, although this is done at the cost of manually labeling training data. To perform extraction, following the generic reference-set based

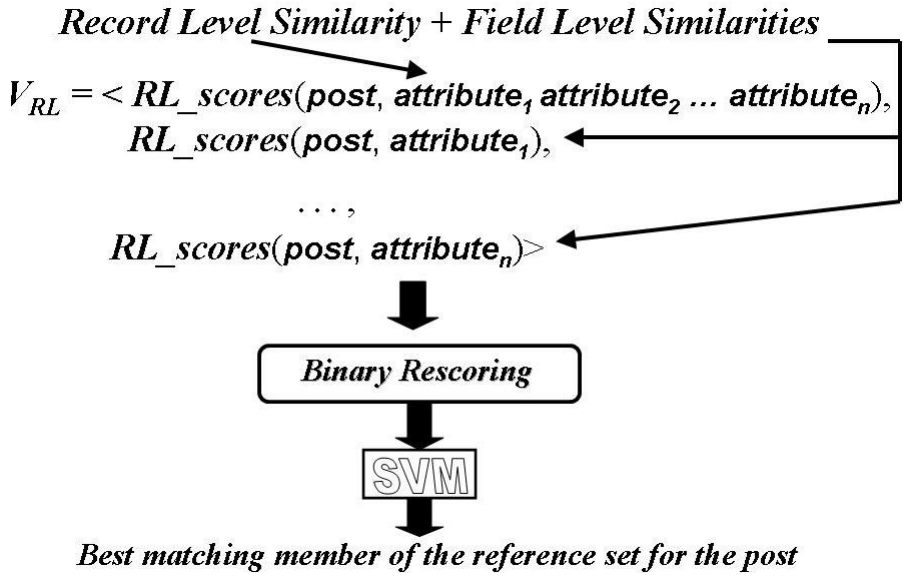

Figure 4.5: Our approach to matching posts to records from a reference set

framework, the matches from the previous matching step are used for extraction. However, unlike the extraction methods of the previous chapters, this extraction algorithm uses machine learning so it can specifically model extraction types to help disambiguate attribute extractions and improve the recall of extraction.

In a broad sense, the extraction algorithm of this chapter has two parts. First the algorithm labels each token with a possible attribute label or as "junk" to be ignored. After all the tokens in a post are labeled, it then cleans each of the extracted labels. Figure 4.6 shows the detailed procedure. Each of the steps shown in this figure are described in detail below.

To begin the extraction process, the post is broken into tokens. Using the post from Figure 4.6 as an example, the set of tokens becomes, {"93-", "4dr", "Honda",...}. Each of these tokens is then scored against each attribute of the record from the reference set that was deemed the match.

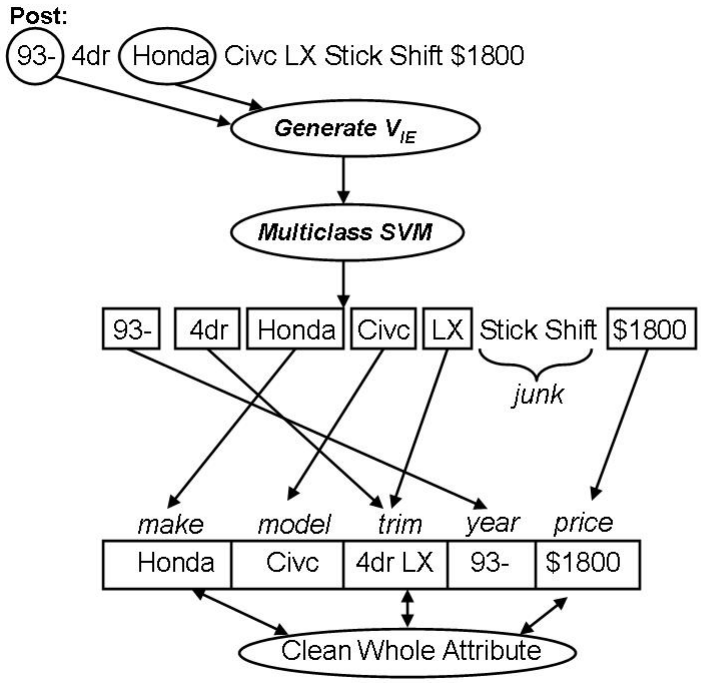

Figure 4.6: Extraction process for attributes

To score the tokens, the extraction process builds a vector of scores, V_{IE}. Like the V_{RL} vector of the matching step, V_{IE} is composed of vectors which represent the similarities between the token and the attributes of the reference set. However, the composition of V_{IE} is slightly different from V_{RL}. It contains no comparison to the concatenation of all the attributes, and the vectors that compose V_{IE} are different from those that compose V_{RL}. Specifically, the vectors that form V_{IE} are called *IE_scores*, and are similar to the *RL_scores* that compose V_{RL}, except they do not contain the *token_scores* component, since each *IE_scores* only uses one token from the post at a time.

The *RL_scores* vector:

RL_scores(post, attribute)=<token_scores(post, attribute),

edit_scores(post, attribute),

other_scores(post, attribute)>

becomes:

IE_scores(token, attribute)=<edit_scores(token, attribute),

other_scores(token, attribute)>

The other main difference between V_{IE} and V_{RL} is that V_{IE} contains a unique vector that contains user defined functions, such as regular expressions, to capture the "common attributes" that are not easily represented by reference sets, such as prices or dates. These attribute types generally exhibit consistent characteristics that allow them to be extracted, and they are usually infeasible to represent in reference sets. This vector is called *common_scores* because the types of characteristics of "common attributes" for extraction are "common" enough between to be easily constructed and used (such as regular expressions).

Using the post of Figure 4.6, assume the reference set match has the make "Honda," the model "Civic" and the year "1993." This means the matching tuple would be {"Honda", "Civic", "1993"}. This match generates the following V_{IE} for the token "civc" of the post:

V_{IE}=<common_scores("civic"),

IE_scores("civc","Honda"),

IE_scores("civc","Civic"),

IE_scores("civc","1993")>

More generally, for a given token, V_{IE} looks like:

$$V_{IE} = <common\_scores(token),$$
$$IE\_scores(token,\ attribute_1),$$
$$IE\_scores(token,\ attribute_2)$$
$$\ldots,$$
$$IE\_scores(token,\ attribute_n)>$$

Each V_{IE} is then passed to a structured SVM [60; 59] trained to give it an attribute type label, such as *make*, *model*, or *price*. Intuitively, similar attribute types should have similar V_{IE} vectors. The makes should generally have high scores against the make attribute of the reference set, and small scores against the other attributes. Further, structured SVMs are able to infer the extraction labels collectively, which helps in deciding between possible token labels. This makes the use of structured SVMs an ideal machine learning method for the extraction task, since collective labeling aids in the disambiguation. Note that since each V_{IE} is not a member of a cluster where the winner takes all, there is no binary rescoring.

Since there are many irrelevant tokens in the post, the SVM learns that any V_{IE} that does associate with a learned attribute type should be labeled as "junk", which can then be ignored. Without the benefits of a reference set, recognizing junk is difficult because the characteristics of the text in the posts are unreliable. For example, if extraction relies solely on capitalization and token location, the junk phrase "Great Deal" might be annotated as an attribute. Many traditional extraction systems that work in the domain of ungrammatical and unstructured text, such as addresses and bibliographies, assume that each token of the text must be classified as something, an assumption that cannot be made with posts.

Nonetheless, it is possible that a junk token will receive an incorrect class label. For example, if a junk token has enough matching letters, it might be labeled as a *trim* (since trims may only be a single letter or two). This leads to noisy tokens within the whole extracted *trim* attribute. Therefore, labeling tokens individually gives an approximation of the data to be extracted.

The extraction approach can overcome the problems of generating noisy, labeled tokens by comparing the whole extracted field to its analogue reference set attribute, similarly to the process in Chapter 2. After all tokens from a post are processed, whole attributes are built and compared to the corresponding attributes from the reference set. This allows removal of the tokens that introduce noise in the extracted attribute.

The removal of noisy tokens from an extracted attribute starts with generating two baseline scores between the extracted attribute and the reference set attribute. One is a Jaccard similarity, to reflect the token level similarity between the two attributes. However, since there are many misspellings and such, an edit-distance based similarity metric, the Jaro-Winkler metric, is also used. These baselines demonstrate how accurately the system extracted/classified the tokens in isolation. Then, the algorithm removes a single token at a time, and rescores the similarity between the reference set attribute and the updated extracted attribute. If the score of the updated extraction is higher than the baseline, the removed token becomes a candidate for removal. After trying all attributes, the candidate for removal that yielded the highest score is removed, and the baseline score is updated to this new similarity score. If no tokens yield a higher score when removed, then the cleaning is finished. The algorithm is shown formally in Figure 4.7.

Algorithm 4.2.1: CLEANATTRIBUTE(E, R)

comment: Clean extracted attribute E using reference set attribute R

RemovalCandidates $C \leftarrow null$
$JaroWinkler_{Baseline} \leftarrow$ JAROWINKLER(E, R)
$Jaccard_{Baseline} \leftarrow$ JACCARD(E, R)
for each token $t \in E$

do
- $X_t \leftarrow$ REMOVETOKEN(t, E)
- $JaroWinkler_{X_t} \leftarrow$ JAROWINKLER(X_t, R)
- $Jaccard_{X_t} \leftarrow$ JACCARD(X_t, R)
- **if** $JaroWinkler_{X_t} > JaroWinkler_{Baseline}$ *and* $Jaccard_{X_t} > Jaccard_{Baseline}$
  - **then** $C \leftarrow C \cup t$

if $C = null$
- **return** (E)

else
- $E \leftarrow$ RemoveMaxCandidate(C,E)
- CLEANATTRIBUTE(E, R)

Figure 4.7: Algorithm to clean an extracted attribute

Note, however, that I do not limit the machine learning component of our extraction algorithm to SVMs. Instead, I claim that in some cases, reference sets can aid machine learning extraction, and to test this, in my architecture I can replace the SVM component with other methods. For example, in my extraction experiments I replace the SVM extractor with a Conditional Random Field (CRF) [32] extractor that uses the V_{IE} as features.

Therefore, the whole extraction process takes a token of the text, creates the V_{IE} and passes this to the machine-learning extractor which generates a label for the token. Then each field is cleaned and the extracted attribute is saved.

4.3 Results

I built the "Phoebus" system to experimentally validate the machine learning approach of this chapter. Specifically, Phoebus tests the technique's accuracy in both the record linkage and the extraction, and incorporates the BSL algorithm for learning and using blocking schemes. The experimental posts data comes from three domains of posts: *BFT Posts*, *eBay Comics*, and *Craig's Cars*, which are described in Chapters 2 and 3. In particular, these three domains have attributes for extraction that are "common attributes" (such as prices and dates) that are hard to distinguish from other attributes from the reference set (such as BFT Posts' numeric star rating, eBay Comics' numeric issue number, and Craig's Cars' years). The three reference sets used for each domain are called Hotels, Comics and Cars respectively, and are described in Table 2.4 of Chapter 2.

Unlike the BFT and eBay comics domains, a strict 1-1 relationship between the post and reference set was not enforced in the Craig's Cars domain. As described previously, Phoebus relaxed the 1-1 relationship to form a 1-n relationship between the posts and the reference set. Sometimes the records do not contain enough attributes to discriminate a single best reference member. For instance, posts that contain just a model and a year might match a couple of reference set records that would differ on the trim attribute, but have the same make, model, and year. Yet, Phoebus can still use this make, model and year accurately for extraction. So, in this case, as mentioned previously, the algorithm picks one of the matches. This way, it can exploit the attributes that it can from the reference set, since it has confidence in those. In some sense, this is similar to the "attributes in agreement" idea.

Since this chapter's technique uses supervised machine learning, for the experiments the posts in each domain are split into two folds, one for training and one for testing. This is usually called two-fold cross validation. However, in many cases two-fold cross validation results in using 50% of the data for training and 50% for testing. I believe that this is too much data to have to label manually, especially as data sets become large. Instead, my experiments focus on using more realistic amounts of training data. One set of experiments uses 30% of the posts for training and tests on the remaining 70%, and the second set of experiments uses just 10% of the posts to train, testing on the remaining 90%. I argue that training on small amounts of data, such as 10%, is an important empirical procedure since real world data sets are large and labeling 50% of such large data sets is time consuming and unrealistic. In fact, the size of the Cars domain prevented me from using 30% of the data for training, since the machine learning algorithms could not scale to the number of training tuples this would generate. So for the Cars domain I only run experiments training on 10% of the data. All experiments are performed 10 times, and I report the average results for these 10 trials.

4.3.1 Record Linkage Results

In this subsection I report our record linkage results, broken down into separate discussions of the blocking results and the matching results.

4.3.1.1 Blocking Results

In order for the BSL algorithm to learn a blocking scheme, it must be provided with methods it can use to compare the attributes. For all domains and experiments I use two

common, simple methods. The first, which I call "token," compares any matching token between the attributes. The second method, "ngram3," considers any matching 3-grams between the attributes.

It is important to note that a comparison between BSL and other blocking *methods*, such as the Canopies method [43] and Bigram indexing [3], is slightly misaligned because the algorithms solve different problems. Methods such as Bigram indexing are techniques that make the *process* of each blocking pass on an attribute more efficient. The goal of BSL, however, is to select which *attribute combinations* should be used for blocking as a whole, trying different attribute and method pairs. Nonetheless, I contend that it is more important to select the right attribute combinations, even using simple methods, than it is to use more sophisticated methods, but without insight as to which attributes might be useful. To test this hypothesis, I compare BSL using the token and 3-gram methods to Bigram indexing over all of the attributes. This is equivalent to forming a disjunction over all attributes using Bigram indexing as the method. I chose Bigram indexing in particular because it is designed to perform "fuzzy blocking" which seems necessary in the case of noisy post data. As stated previously [3], I use a threshold of 0.3 for Bigram indexing, since that works the best. I also compare BSL to running a disjunction over all attributes using the simple token method only. In my results, I call this blocking rule "Disjunction." This disjunction mirrors the idea of picking the simplest possible blocking method: generating candidates by finding a common token in at least one attribute.

As stated previously, the two goals of blocking can be quantified by the Reduction Ratio (RR) and the Pairs Completeness (PC). Table 4.4 shows not only these values but also how many candidates were generated on average over the entire test set, comparing

Table 4.4: Blocking results using the BSL algorithm (amount of data used for training shown in parentheses).

	RR	PC	# Cands	Time Learn (s)	Time Run (s)	Time match (s)
Hotels (30%)						
BSL	81.56	99.79	19,153	69.25	24.05	60.93
Disjunction	67.02	99.82	34,262	0	12.49	109.00
Bigrams	61.35	72.77	40,151	0	1.2	127.74
Hotels (10%)						
BSL	84.47	99.07	20,742	37.67	31.87	65.99
Disjunction	66.91	99.82	44,202	0	15.676	140.62
Bigrams	60.71	90.39	52,492	0	1.57	167.00
eBay Comics (30%)						
BSL	42.97	99.75	284,283	85.59	36.66	834.94
Disjunction	37.39	100.00	312,078	0	45.77	916.57
Bigrams	36.72	69.20	315,453	0	102.23	926.48
eBay Comics (10%)						
BSL	42.97	99.74	365,454	34.26	35.65	1,073.34
Disjunction	37.33	100.00	401,541	0	52.183	1,179.32
Bigrams	36.75	88.41	405,283	0	131.34	1,190.31
Craig's Cars (10%)						
BSL	88.48	92.23	5,343,424	465.85	805.36	25,114.09
Disjunction	87.92	89.90	5,603,146	0	343.22	26.334.79
Bigrams	97.11	4.31	1,805,275	0	996.45	8,484.79

the three different approaches. Table 4.4 also shows how long it took each method to learn the rule and run the rule. Lastly, the column "Time match" shows how long the classifier needs to run given the number of candidates generated by the blocking scheme.

Table 4.5 shows a few example blocking schemes that the BSL algorithm generated. For a comparison of the attributes BSL selected to the attributes picked manually for different domains where the data is structured the reader is pointed to my previous work on the topic [45].

The results of Table 4.4 validate the idea that it is more important to pick the correct attributes to block on (using simple methods) than to use sophisticated methods without attention to the attributes. Comparing the BSL rule to the Bigram results, the

Table 4.5: Some example blocking schemes learned for each of the domains.

BFT Domain (30%)
({hotel area,token} ∧ {hotel name,token} ∧ {star rating, token}) ∪ ({hotel name, ngram3})
BFT Domain (10%)
({hotel area,token} ∧ {hotel name,token}) ∪ ({hotel name,ngram3})
eBay Comics Domain (30%)
({title, token})
eBay Comics Domain (10%)
({title, token}) ∪ ({issue number,token} ∧ {publisher,token} ∧ {title,ngram3})
Craig's Cars Domain (10%)
({make,token}) ∪ ({model,ngram3}) ∪ ({year,token} ∧ {make,ngram3})

combination of PC and RR is always better using BSL. Note that although in the Cars domain Bigram took significantly less time with the classifier due to its large RR, it did so because it only had a PC of 4%. In this case, Bigrams was not even covering 5% of the true matches.

Further, the BSL results are better than using the simplest method possible (the Disjuction), especially in the cases where there are many records to test upon. As the number of records scales up, it becomes increasingly important to gain a good RR, while maintaining a good PC value as well. This savings is dramatically demonstrated by the Cars domain, where BSL outperformed the Disjunction in both PC and RR.

One surprising aspect of these results is how prevalent the token method is within all the domains. I expect that the ngram method would be used almost exclusively since there are many spelling mistakes within the posts. However, this is not the case. I hypothesize that the learning algorithm uses the token methods because they occur with more regularity across the posts than the common ngrams would since the spelling mistakes might vary quite differently across the posts. This suggests that there might be

more regularity, in terms of what one can learn from the data, across the posts than I initially surmised.

Another interesting result is the poor reduction ratio of the eBay Comics domain. This happens because most of the rules contain the disjunct that finds a common token within the comic title. This rule produces such a poor reduction ratio because the value for this attribute is the same across almost all reference set records. That is to say, when there are just a few unique values for the BSL algorithm to use for blocking, the reduction ratio will be small. In this domain, there are only two values for the comic title attribute, "Fantastic Four" and "Incredible Hulk" in the Comic reference set. So it makes sense that if blocking is done using the title attribute only, the reduction is about half, since blocking on the value "Fantastic Four" just gets rid of the "Incredible Hulk" comics. This points to an interesting limitation of the BSL algorithm. If there are not many distinct values for the different attribute and method pairs that BSL can use to learn from, then this lack of values cripples the performance of the reduction ratio. Intuitively though, this makes sense, since it is hard to distinguish good candidate matches from bad candidate matches if they share the same attribute values.

Another result worth mentioning is that in the BFT domain BSL gets a lower RR but the same PC when it uses less training data. This happens because the BSL algorithm runs until it has no more examples to cover, so if those last few examples introduce a new disjunct that produces a lot of candidates, while only covering a few more true positives, then this would cause the RR to decrease, while keeping the PC at the same high rate. This is in fact what happens in this case. One way to curb this behavior would be to set some sort of stopping threshold for BSL, but as I mention previously, maximizing the PC

is the most important thing, so I choose not to do this. The goal is for BSL to cover as many true positives as it can, even if that means losing a bit in the reduction.

In fact, I next test this notion explicitly. I set a threshold in the SCA such that after 95% of the training examples are covered, the algorithm stops and returns the learned blocking scheme. This helps to avoid the situation where BSL learns a very general conjunction, solely to cover the last few remaining training examples. When that happens, BSL might end up lowering the RR, at the expense of covering just those last training examples, because the rule learned to cover those last examples is overly general and returns too many candidate matches.

Table 4.6: A comparison of BSL covering all training examples, and covering 95% of the training examples

Domain	Record Linkage F-Measure	RR	PC
BFT Domain			
No Thresh (30%)	90.63	81.56	**99.79**
95% Thresh (30%)	90.63	**87.63**	97.66
eBay Comics Domain			
No Thresh (30%)	91.30	42.97	99.75
95% Thresh (30%)	91.47	42.97	99.69
Craig's Cars Domain			
No Thresh (10%)	**77.04**	88.48	**92.23**
95% Thresh (10%)	67.14	**92.67**	83.95

Table 4.6 shows that when BSL uses a threshold in the BFT and Craig's Cars domain there is a statistically significant drop in Pairs Completeness with a statistically significant increase in Reduction Ratio.[2] This is expected behavior since the threshold causes BSL to kick out of SCA before it can cover the last few training examples, which in turn allows

[2]Bold means statistically significant using a two-tailed t-test with α set to 0.05

BSL to retain a rule with high RR, but lower PC. However, looking at the record linkage results (shown as Record Linkage F-Measure in Table 4.6), this threshold does in fact have a large, detrimental effect for the Craig's Cars domain.[3] *When BSL uses a threshold, the candidates not discovered by the rule generated when using the threshold have an effect of 10% on the final F-measure match results.*[4] Therefore, since the F-measure results differ by so much, I conclude that it is worthwhile to maximize PC when learning rules with BSL, even if the RR may decrease. That is to say, even in the presence of noise, which in turn may lead to overly generic blocking schemes, BSL should try to maximize the true matches it covers, because avoiding even the most difficult cases to cover may affect the matching results. As Table 4.6 shows, this is especially true in the Craig's Cars domain where matching is much more difficult than in the BFT domain.

Interestingly, in the eBay Comics domain there is not see a statistically significant difference in the RR and PC. This is because across trials BSL almost always learns the same rule whether it uses a threshold or not, and this rule covers enough training examples that the threshold is not hit. Further, there is no statistically significant change in the F-measure record linkage results for this domain. This is expected since BSL would generate the same candidate matches, whether it uses the threshold or not, since in both cases it almost always learns the same blocking rules.

The BSL results are encouraging because they show that the algorithm also works for blocking when matching unstructured and ungrammatical text to a relational data source.

[3]Please see subsection 4.3.1.2 for a description of the record linkage experiments and results.

[4]Much of this difference is attributed to the non-threshold version of the algorithm learning a final predicate that includes the make attribute by itself, which the version with a threshold does not learn. Since each make attribute value covers many records, it generates many candidates which results in increasing PC while reducing RR.

This means the algorithm works in this special case too, not just the case of traditional record linkage where one matches one structured source to another. This means the overall algorithm of this chapter is more scalable because it uses fewer candidate matches.

4.3.1.2 Matching Results

The matching step approach in this chapter is compared to WHIRL [16]. WHIRL performs record linkage by performing soft-joins using vector-based cosine similarities between the attributes. Other record linkage systems require decomposed attributes for matching, which is not the case with the posts. WHIRL serves as the benchmark because it does not have this requirement. To mirror the alignment task of Phoebus, the experiment supplies WHIRL with two tables: the test set of posts (either 70% or 90% of the posts) and the reference set with the attributes concatenated to approximate a record level match. The concatenation is also used because when matching on each individual attribute, it is not obvious how to combine the matching attributes to construct a whole matching reference set member.

To perform the record linkage, WHIRL does soft-joins across the tables, which produces a list of matches, ordered by descending similarity score. For each post with matches from the join, the reference set member(s) with the highest similarity score(s) is called its match. In the Craig's Cars domain the matches are 1-N, so this means that only 1 match from the reference set will be exploited later in the information extraction step. To mirror this idea, the number of possible matches in a 1-N domain for recall is counted as the number of posts that have a match in the reference set, rather than the reference set members themselves that match. Also, this means that we only add a single match

to our total number of correct matches for a given post, rather than all of the correct matches, since only one matters. This is done for both WHIRL and Phoebus, and more accurately reflects how well each algorithm would perform as the processing step before our information extraction step.

The record linkage results for both Phoebus and WHIRL are shown in Table 4.7. Note that the amount of training data for each domain is shown in parentheses. All results are statistically significant using a two-tailed paired t-test with α=0.05, except for the precision between WHIRL and Phoebus in the Craig's Cars domain, and the precision between Phoebus trained on 10% and 30% of the training data in the eBay Comics domain.

Phoebus outperforms WHIRL because it uses many similarity types to distinguish matches. Also, since Phoebus uses both a record level *and* attribute level similarities, it is able to distinguish between records that differ in more discriminative attributes. This is especially apparent in the Craig's Cars domain. First, these results indicate the difficulty of matching car posts to the large reference set. This is the largest experimental domain yet used for this problem, and it is encouraging how well my approach outperforms the baseline. It is also interesting that the results suggest that both techniques are equally accurate in terms of precision (in fact, there is no statistically significant difference between them in this sense) but Phoebus is able to retrieve many more relevant matches. This means Phoebus can capture more rich features that predict matches than WHIRL's cosine similarity alone. I expect this behavior because Phoebus has a notion of both field and token level similarity, using many different similarity measures. This justifies my use

of the many similarity types and field and record level information, since the goal is to find as many matches as possible.

Table 4.7: Record linkage results

	Precision	Recall	F-measure
BFT			
Phoebus (30%)	87.70	93.78	**90.63**
Phoebus (10%)	87.85	92.46	90.09
WHIRL	83.53	83.61	83.13
eBay Comics			
Phoebus (30%)	87.49	95.46	**91.30**
Phoebus (10%)	85.35	93.18	89.09
WHIRL	73.89	81.63	77.57
Craig's Cars			
Phoebus (10%)	69.98	85.68	**77.04**
WHIRL	70.43	63.36	66.71

It is also encouraging that using only 10% of the data for labeling, Phoebus is able to perform almost as well as using 30% of the data for training. Since the amount of data on the Web is vast, only having to label 10% of the data to get comparative results is preferable when the cost of labeling data is great. Especially since the clean annotation, and hence relational data, comes from correctly matching the posts to the reference set, not having to label much of the data is important if I want this technique to be widely applicable. In fact, as stated above, I faced this practical issue in the Craig's Cars domain where I was unable to use 30% for training since the machine learning method would not scale to the number of candidates generated by this much training data.

While the matching method performs well and outperforms WHIRL from the results above, it is not clear whether it is the use of the many string metrics, the inclusion of the

attributes and their concatenation, or the SVM itself that provides this advantage. To test the advantages of each piece, I ran several experiments isolating each of these ideas.

First, I ran Phoebus matching on only the concatenation of the attributes from the reference set, rather than the concatenation and all the attributes individually. Earlier, I stated that I use the concatenation to mirror the idea of record level similarity, and I also use each attribute to mirror field level similarity. It is my hypothesis that in some cases, a post will match different reference set records with the same record level score (using the concatenation), but it will do so matching on different attributes. By removing the individual attributes and leaving only the concatenation of them for matching, I can test how the concatenation influences the matching in isolation. Table 4.8 shows these results for the different domains.

Table 4.8: Matching using only the concatenation

	Precision	Recall	F-Measure
BFT			
Phoebus (30%)	87.70	93.78	90.63
Concatenation Only	88.49	93.19	90.78
WHIRL	83.61	83.53	83.13
eBay Comics			
Phoebus (30%)	87.49	95.46	**91.30**
Concatenation Only	61.81	46.55	51.31
WHIRL	73.89	81.63	77.57
Craig's Cars			
Phoebus (10%)	69.98	85.68	**77.04**
Concatenation Only	47.94	58.73	52.79
WHIRL	70.43	63.36	66.71

For the Craig's Cars and eBay Comics there is an improvement in F-measure, indicating that using the attributes and the concatenation is much better for matching

than using the concatenation alone. This supports the notion that the matcher also needs a method to capture the significance of matching individual attributes since some attributes are better indicators of matching than others. It is also interesting to note that for both these domains, WHIRL does a better job than the machine learning using only the concatenation, even though WHIRL also uses a concatenation of the attributes. This is because WHIRL uses information-retrieval-style matching to find the best match, and the machine learning technique tries to learn the characteristics of the best match. Clearly, it is very difficult to learn what such characteristics are.

In the BFT domain, there is not a statistically significant difference in F-measure using the concatenation alone. This means that the concatenation is sufficient to determine the matches, so there is no need for individual fields to play a role. More specifically, the hotel name and area seem to be the most important attributes for matching and by including them as part of the concatenation, the concatenation is still distinguishable enough between all records to determine matches. Since in two of the three domains there is a huge improvement, and across all domains the algorithm never loses in F-measure, using both the concatenation and the individual attributes is valid for the matching. Also, since in two domains the concatenation alone was worse than WHIRL, I conclude that part of the reason Phoebus can outperform WHIRL is the use of the individual attributes for matching.

The next experiment tests how important it is to include all of the string metrics in the feature vector for matching. To test this idea, I compare using all the metrics to using just one, the Jensen-Shannon distance. I choose the Jensen-Shannon distance because it outperformed both TF/IDF and even a "soft" TF/IDF (one that accounts for fuzzy

Table 4.9: Using all string metrics versus using only the Jensen-Shannon distance

	Precision	Recall	F-Measure
BFT			
Phoebus (30%)	87.70	93.78	90.63
Jensen-Shannon Only	89.65	92.28	90.94
WHIRL	83.61	83.53	83.13
eBay Comics			
Phoebus (30%)	87.49	95.46	**91.30**
Jensen-Shannon Only	65.36	69.96	67.58
WHIRL	73.89	81.63	77.57
Craig's Cars			
Phoebus (10%)	69.98	85.68	**77.04**
Jensen-Shannon Only	72.87	59.43	67.94
WHIRL	70.43	63.36	66.71

token matches) in the task of selecting the right reference sets for a given set of posts in Chapter 2. These results are shown in Table 4.9.

As Table 4.9 shows, using all the metrics yielded a statistically significant, large improvement in F-measure for the eBay Comics and Craig's Cars domains. This means that some of the other string metrics, such as the edit distances, capture similarities that the Jensen-Shannon distance alone does not. Interestingly, in both domains, using Phoebus with only the Jensen-Shannon distance does not dominate WHIRL's performance. Therefore, the results of Table 4.9 and Table 4.8 demonstrate that Phoebus benefits from the combination of many, varied similarity metrics along with the use of individual attributes for field level similarities, and both of these aspects contribute to Phoebus outperforming WHIRL.

In the case of the BFT data, there is not a statistically significant difference in the matching results, so in this case the other metrics do not provide relevant information

for matching. Therefore, all the matches missed by the Jensen-Shannon only method are also missed when I include all of the metrics. Hence, either these missed matches are very difficult to discover, or there is not a string metric in my method yet that can capture the similarity. For example, when the post has a token "DT" and the reference set record it should match has a hotel area of "Downtown," then an abbreviation metric could capture this relationship. However, Phoebus does not include an abbreviation similarity measure.

Since none of the techniques in isolation consistently outperforms WHIRL, I conclude that Phoebus outperforms WHIRL because it combines multiple string metrics, it uses both individual attributes and the concatenation, and, as stated in Section 4.1.2, the SVM classifier is well suited for the record linkage task. These results also justify the inclusion of many metrics and the individual attributes, along with our use of SVM as our classifier.

The last matching experiment justifies our binary rescoring mechanism. Table 4.10 shows the results of performing the binary rescoring for record linkage versus not performing this binary recoring. I hypothesize earlier in this chapter that the binary rescoring will allow the classifier to more accurately make match decisions because the rescoring separates out the best candidate as much as possible. Table 4.10 shows this to be the case, as across all domains when I perform the binary rescoring the algorithm gains a statistically significant amount in the F-measure. This shows that the record linkage is more easily able to identify the true matches from the possible candidates when the only difference in the record linkage algorithm is the use of binary rescoring.

Table 4.10: Record linkage results with and without binary rescoring

	Precision	Recall	F-Measure
BFT			
Phoebus (30%)	87.70	93.78	**90.63**
No Binary Rescoring	75.44	81.82	78.50
Phoebus (10%)	87.85	92.46	**90.09**
No Binary Rescoring	73.49	78.40	75.86
eBay Comics			
Phoebus (30%)	87.49	95.46	**91.30**
No Binary Rescoring	84.87	89.91	87.31
Phoebus (10%)	85.35	93.18	**89.09**
No Binary Rescoring	81.52	88.26	84.75
Craig's Cars			
Phoebus (10%)	69.98	85.68	**77.04**
No Binary Rescoring	39.78	48.77	43.82

4.3.2 Extraction Results

This section presents results that experimentally validate the supervised machine learning approach to extracting the actual attributes embedded within the post.

As stated in Section 4.2 on Extraction, I also created a version of Phoebus that uses CRFs, which I call PhoebusCRF. PhoebusCRF uses the same extraction features (V_{IE}) as Phoebus using the SVM, such as the common score regular expressions and the string similarity metrics. I include PhoebusCRF to show that supervised extraction in general can benefit from our reference set matching when used in a reference-set based extraction framework.

One component of the extraction vector V_{IE} is the vector *common_scores*, which includes user defined functions, such as regular expressions. Since these are the only domain specific functions used in the algorithm, the *common_scores* for each domain must be specified. For the Hotels domain, the *common_scores* includes the functions

matchPriceRegex and *matchDateRegex*. Each of these functions gives a positive score if a token matches a price or date regular expression, and 0 otherwise. For the Comic domain, *common_scores* contains the functions *matchPriceRegex* and *matchYearRegex*, which also give positive scores when a token matches the regular expression. In the Cars domain, *common_scores* uses the function *matchPriceRegex* (since year is an attribute of the reference set, we do not use a common score to capture its form).

Table 4.11, Table 4.12, and Table 4.13 compare the field-level extraction results using Phoebus and PhoebusCRF to the ARX results of Chapter 2 and Chapter 3. The results for CRF-Win, CRF-Orth, and Amilcare are repeated for comparison. Unlike the matching results, I only use 10% of the data for training for Phoebus and PhoebusCRF to make the comparison to CRF-Win, CRF-Orth and Amilcare explicit. The attributes shown in *italics* are those represented in the reference set, and those in plain text are "common" attributes. Note that ARX does not have results for the common attributes since it could only extract such attributes using regular expressions, since common attributes are not represented by a reference set.

Phoebus and PhoebusCRF outperform the other systems on almost all attributes (12 of 16), as shown in Table 4.14. In fact, that CRF-Win performed the best on the price attribute of the eBay comic domain is misleading since none of the results for that attribute are statistically significant with respect to each other (there are few extractions of this attribute, so the results varied wildly across the trials). Further, since both Amilcare and ARX use reference set data, for every single attribute, the method with the maximum F-measure benefited from a reference set.

Table 4.11: Field level extraction results: BFT domain

BFT Posts				
		Recall	Precision	F-Measure
Local Area	Phoebus	77.80	83.58	80.52
	PhoebusCRF	80.71	83.38	**82.01**
	ARX	62.23	85.36	71.98
	CRF-Orth	64.51	77.14	70.19
	CRF-Win	28.51	39.79	33.07
	Amilcare	64.78	71.59	68.01
Date	Phoebus	82.13	83.06	82.59
	PhoebusCRF	84.39	84.48	84.43
	ARX	.	.	.
	CRF-Orth	81.85	81.00	81.42
	CRF-Win	80.39	79.54	79.96
	Amilcare	86.18	94.10	**89.97**
Hotel Name	Phoebus	75.59	74.25	74.92
	PhoebusCRF	81.46	81.69	**81.57**
	ARX	70.09	77.16	73.46
	CRF-Orth	66.90	68.07	67.47
	CRF-Win	39.84	42.95	41.33
	Amilcare	58.96	67.44	62.91
Price	Phoebus	93.12	98.46	**95.72**
	PhoebusCRF	90.34	92.60	91.46
	ARX	.	.	.
	CRF-Orth	92.61	92.63	92.62
	CRF-Win	90.47	90.40	90.43
	Amilcare	88.04	91.10	89.54
Star Rating	Phoebus	96.94	96.90	**96.92**
	PhoebusCRF	96.17	96.74	96.45
	ARX	83.94	99.44	91.03
	CRF-Orth	94.37	95.17	94.77
	CRF-Win	93.77	94.67	94.21
	Amilcare	95.58	97.35	96.46

Phoebus performs especially well in the Craig's Cars domain, where it is the best system on all the attributes. One interesting thing to note about this result is that while the record linkage results are not spectacular for this domain, they are good enough to yield very highly accurate extraction results. This is because most times when the system is not picking the best match from the reference set, it is still picking one that is close enough such that most of the reference set attributes are useful for extraction. This is

Table 4.12: Field level extraction results: eBay Comics domain.

eBay Comics Posts				
		Recall	Precision	F-Measure
Descr.	Phoebus	30.16	27.15	**28.52**
	PhoebusCRF	15.45	26.83	18.54
	ARX	24.80	15.90	19.38
	CRF-Orth	7.07	24.95	10.81
	CRF-Win	6.08	14.56	8.78
	Amilcare	8.00	52.55	13.78
Issue Num.	Phoebus	80.90	82.17	81.52
	PhoebusCRF	83.01	84.68	**83.84**
	ARX	46.21	87.21	60.41
	CRF-Orth	81.88	80.34	81.10
	CRF-Win	80.67	80.52	80.59
	Amilcare	77.66	89.11	82.98
Price	Phoebus	39.84	60.00	46.91
	PhoebusCRF	29.09	55.40	35.71
	ARX	.	.	.
	CRF-Orth	22.90	34.59	27.43
	CRF-Win	40.53	100.00	**65.61**
	Amilcare	41.21	66.67	50.93
Publisher	Phoebus	99.85	83.89	91.18
	PhoebusCRF	53.22	87.29	64.26
	ARX	100.00	84.16	**91.40**
	CRF-Orth	41.71	88.39	55.42
	CRF-Win	38.42	86.84	52.48
	Amilcare	63.75	90.48	74.75
Title	Phoebus	89.37	89.37	89.37
	PhoebusCRF	90.64	92.13	91.37
	ARX	90.08	90.78	90.43
	CRF-Orth	83.58	84.13	83.85
	CRF-Win	77.69	77.85	77.77
	Amilcare	89.88	95.65	**92.65**
Year	Phoebus	77.50	97.35	**86.28**
	PhoebusCRF	54.63	85.07	66.14
	ARX	.	.	.
	CRF-Orth	47.31	78.14	58.19
	CRF-Win	37.56	66.43	46.00
	Amilcare	77.05	85.67	81.04

why the trim extraction results are the lowest, because that is often the attribute that determines a match from a non-match. The record linkage step likely selects a car that is close, but differs in the trim, so the match is incorrect and the trim will most likely not

Table 4.13: Field level extraction results: Craig's Cars domain.

Craig's Cars Posts				
		Recall	Precision	F-Measure
Make	Phoebus	98.21	99.93	**99.06**
	PhoebusCRF	90.73	96.71	93.36
	ARX	95.99	100.00	97.95
	CRF-Orth	82.03	85.40	83.66
	CRF-Win	80.09	77.38	78.67
	Amilcare	97.58	91.76	94.57
Model	Phoebus	92.61	96.67	**94.59**
	PhoebusCRF	84.58	94.10	88.79
	ARX	83.02	95.01	88.61
	CRF-Orth	71.04	77.81	74.25
	CRF-Win	67.87	69.67	68.72
	Amilcare	78.44	84.31	81.24
Price	Phoebus	97.17	95.91	**96.53**
	PhoebusCRF	93.59	92.59	93.09
	ARX	.	.	.
	CRF-Orth	93.76	91.64	92.69
	CRF-Win	91.60	89.59	90.58
	Amilcare	90.06	91.27	90.28
Trim	Phoebus	63.11	70.15	**66.43**
	PhoebusCRF	55.61	64.95	59.28
	ARX	39.52	66.94	49.70
	CRF-Orth	47.94	47.90	47.88
	CRF-Win	38.77	39.35	38.75
	Amilcare	27.21	53.99	35.94
Year	Phoebus	88.48	98.23	**93.08**
	PhoebusCRF	85.54	96.44	90.59
	ARX	76.28	99.80	86.47
	CRF-Orth	84.99	91.33	88.04
	CRF-Win	83.03	86.12	84.52
	Amilcare	86.32	91.92	88.97

Table 4.14: Summary results for extraction showing the number of times each system had highest F-Measure for an attribute.

Domain	Num. of Max. F-Measures						Total Attributes
	Phoebus	PhoebusCRF	ARX	Amilcare	CRF-Win	CRF-Orth	
BFT	**2**	**2**	0	1	0	0	5
eBay Comics	**2**	1	1	1	1	0	6
Craig's Cars	**5**	0	0	0	0	0	5
All	**9**	**3**	1	2	1	0	16

be extracted correctly, but the rest of the attributes can be extracted using the reference set member.

Another interesting result is that Phoebus or PhoebusCRF handily outperformed the CRF based methods on every attribute (except for comic price which was insignificant for all systems). This lends credibility to my claim that by training the system to extract all of the attributes, even those in the reference set, it can more accurately extract attributes not in the reference set because it is training the system to identify what something is not.

The overall performance of Phoebus validates my supervised machine learning approach to extraction. By infusing information extraction with the outside knowledge of reference sets, Phoebus is able to perform well across three different domains, each representative of a different type of source of posts: the auction sites, Internet classifieds and forum/bulletin boards.

Chapter 5

Related Work

The work in this thesis touches upon a number of areas, and I present related research for the various areas and how my work differs from other research in the field.

First, the main thrust of this research is on the topic of extraction, both automatic and machine-learning based. Although my automatic extraction method touches upon methods of unsupervised extraction [8; 28; 51], such methods as theirs would not work on posts where one cannot assume that any structure, and therefore extraction pattern, will hold for all of the posts. Note that much work on extraction is supervised, and so my automatic approach differs substantially.

Further, although my supervised approach fits into the context of other methods for extraction that use supervised machine learning, my method is quite different as well. Many other methods, such as Conditional Random Fields (CRF), assume at least some structure in the extracted attributes to do the extraction. As my extraction experiments show, the CRF-Win and CRF-Orth implemented using the Simple Tagger implementation of Conditional Random Fields [42] did not perform well. Other extraction approaches, such as Datamold [7] and CRAM [1], segment whole records (like bibliographies) into

attributes, with little structural assumption. In fact, CRAM even uses reference sets to aid its extraction. However, both systems require that every token of a record receive a label, which is not possible with posts that are filled with irrelevant, "junk" tokens. Along the lines of CRAM and Datamold, the work of Bellare and McCallum [4] uses a reference set to train a CRF to extract data, which is similar to my PhoebusCRF implementation. However, there are two differences between PhoebusCRF and their work [4]. First, the work of Bellare and McCallum [4] mentions that reference set records are matched using simple heuristics, but it is unclear how this is done. In my work, matching is done explicitly and accurately through record linkage. Second, their work only uses the records from the reference set to label tokens for training an extraction module, while PhoebusCRF uses the actual values from the matching reference set record to produce useful features for extraction and annotation.

Another IE approach similar to mine performs named entity recognition using "Semi-CRFs" with a dictionary component [15], which functions like a reference set. However, in their work the dictionaries are defined as lists of single attribute entities, so finding an entity in the dictionary is a look-up task. My reference sets are relational data, so finding the match becomes a record linkage task. Further, their work on Semi-CRFs [15] focuses on the task of labeling segments of tokens with a uniform label, which is especially useful for named entity recognition. In the case of posts, however, Phoebus needs to relax such a restriction because in some cases such segments will be interrupted, as the case of a hotel name with the area in the middle of the hotel name segment. So, unlike their work, Phoebus makes no assumptions about the structure of posts. Recently, Semi-CRFs have been extended to use database records in the task of integrating unstructured data with

relational databases [41]. This work is similar to mine in that it links unstructured data, such as paper citations, with relational databases, such as reference sets of authors and venues. The difference is that I view this as a record linkage task, namely finding the right reference set tuple to match. In their paper, even though they use matches from the database to aid extraction, they view the linkage task as an extraction procedure followed by a matching task. Lastly, I are not the first to consider structured SVMs for information extraction. Previous work used structured SVMs to perform Named Entity Recognition [60] but their extraction task does not use reference sets.

Using reference-sets to aid extraction is similar to the work on ontology-based information extraction [25]. Later versions of their work even talk about using ontology-based information extraction as a means to semantically annotate unstructured data such as car classifieds [21]. However, in contrast to my work, the information extraction is performed by a keyword-lookup into the ontology along with structural and contextual rules to aid the labeling. The ontology itself contains keyword misspellings and abbreviations, so that the look-up can be performed in the presence of noisy data. I argue the ontology-based extraction approach is less scalable than a record linkage type matching task because creating and maintaining the ontology requires extensive data engineering in order to encompass all possible common spelling mistakes and abbreviations. Further, if new data is added to the ontology, additional data engineering must be performed. In my work, I can simply add new tuples to the reference set. Also, my methods can construct its own reference set, and none of these approaches mentions building the ontology to use for extraction. Lastly, in contrast to my work, this ontology based work assumes contextual and structural rules will apply, making an assumption about the data to extract from.

Yet another interesting approach to information extraction using ontologies is the Textpresso system which extracts data from biological text [48]. This system uses a regular expression based keyword look-up to label tokens in some text based on the ontology. Once all tokens are labeled, Textpresso can perform "fact extraction" by extracting sequences of labeled tokens that fit a particular pattern, such as gene-allele reference associations. Although this system again uses a reference set for extraction, it differs in that it does a keyword look-up into the lexicon.

My work on automatically building reference sets is related to the topic of ontology creation. My approach follows almost all of the current research on this topic in that I generate some candidate terms and then plug these terms into an hierarchy. The classic paper of Sanderson and Croft [54] laid much of the groundwork by developing the subsumption heuristic based on terms. Other papers have applied the same technique to developing other hierarchies, such as an ontology of Flickr tags [55]. Other techniques find candidate terms, but build the hierarchies using other methods such as Smoothed Formal Concept Analysis [13], Principal Component Analysis [23] or even Google to determine term dependencies [40]. However, there are a number of differences between the above work and mine (beyond the fact that some of the methods use NLP to determine the terms, which we cannot do with posts). First is that all of the other systems build up large concept hierarchies. In my case, I build up a large number of small independent hierarchies, which I then flatten into a reference set. So, I use hierarchy building in order to construct a set of relational tuples, which are only an hierarchy in the sense that they can be represented by trees. My method is more like fleshing out an taxonomy, rather than building a conceptual ontology. I also present work that suggests how many posts

are needed to construct the reference set, in contrast to the other work that just uses many web pages without this consideration. I also present an algorithm that suggests whether the user can actually do the construction by comparing bigram-type distributions to known distributions. These other construction methods do not have this functionality.

The matching step yields annotation for posts and as such it is somewhat related to the topic of semantic annotation as well. Many researchers have followed this path, attempting to automatically mark up documents for the Semantic Web [12; 22; 27; 61; 35]. However, these systems rely on lexical information, such as part-of-speech tagging or shallow Natural Language Processing to do their extraction/annotation (e.g., Amilcare, Ciravegna, 2001) or specific structure to exploit. According to a recent survey [53], systems that perform semantic annotation separate into three categories: rule-based, pattern-based, and wrapper induction methods. However, the rule-based and pattern-based methods rely on regularity within the text, which is not the case with posts. Also, beyond exploiting regular structure, the wrapper induction methods use supervised machine learning instead of unsupervised methods.

The semantic annotation system closest to mine is SemTag [20], which first identifies tokens of interest in the text, and then labels them using the TAP taxonomy, which is similar to a reference set. This taxonomy is carefully crafted, which gives it good accuracy and meaning. However, annotation using ontologies faces the same issues as extraction using ontologies. Particularly, it is hard to model and add new tuples in an ontology. In contrast, my reference sets are flexible and I can incorporate any that I can automatically collect.

Further, SemTag focuses on disambiguation which my approach (both automatic and supervised) avoids. If one looks up the token "Jaguar" it might refer to a car or an animal, because SemTag disambiguates after labeling. In my case, I perform disambiguation before the labeling procedure, during the selection of the relevant reference sets, whether given by the user, selected by the user, or built by the algorithm.

The supervised machine learning approach to matching is similar to record linkage which is not a new topic [26] and is well studied even now [6]. However, most current research focuses on matching one set of records to another set of records based on their decomposed attributes. There is little work on matching data sets where one record is a single string composed of the other data set's attributes to match on, as in the case with posts and reference sets. The WHIRL system [16] allows for record linkage without decomposed attributes, but as shown in results of Chapter 4 my approach outperforms WHIRL, since WHIRL relies solely on the vector-based cosine similarity between the attributes, while my approach exploits a larger set of features to represent both field and record level similarity. I note with interest the EROCS system [10] where the authors tackle the problem of linking full text documents with relational databases. The technique involves filtering out all non-nouns from the text, and then finding the matches in the database. This is an intriguing approach; interesting future work would involve perfor3g a similar filtering for larger documents and then applying my machine learning algorithm to match the remaining nouns to reference sets.

Using the reference set's attributes as normalized values is similar to the idea of data cleaning. However, most data cleaning algorithms assume tuple-to-tuple transformations [34; 11]. That is, some function maps the attributes of one tuple to the attributes of

another. This approach would not work on ungrammatical and unstructured data, where all attributes are embedded within the post, which maps to a set of attributes from the reference set.

My work on automatically selecting reference sets is similar to resource selection in distributed information retrieval, sometimes used for the "hidden web." In resource selection, different servers are chosen from a set of servers to return documents for a given query. Craswell, Bailey, and Hawking [18] compare three popular approaches to resource selection. However, these retrieval techniques execute probe queries to estimate the resource's data coverage and its effectiveness at returning relevant documents. Then, these coverage and effectiveness statistics are used to select and merge the appropriate resources for a query. This overhead is unnecessary for my task. Because I have full access to all of our reference sets in our repository, the system already knows the full data coverage, and does not need to estimate our repository's effectiveness, because it always returns all of the sets.

Lastly, there is work related to the BSL algorithm that is proposed to aid the supervised machine learning approach to matching by providing a better approach to blocking. In recent work on learning efficient blocking schemes Bilenko et al., 5 developed a system for learning disjunctive normal form blocking schemes. However, they learn their schemes using a graphical set covering algorithm, while we use a version of the Sequential Covering Algorithm (SCA). There are also similarities between our BSL algorithm and work on mining association rules from transaction data [2]. Both algorithms discover propositional rules. Further, both algorithms use multiple passes over a data set to discover their rules. However despite these similarities, the techniques really solve different

problems. BSL generates a set of candidate matches with a minimal number of false positives. To do this, BSL learns conjunctions that are maximally specific (eliminating many false positives) and unions them together as a single disjunctive rule (to cover the different true positives). Since the conjunctions are maximally specific, BSL uses SCA underneath, which learns rules in a depth-first, general to specific manner [47]. On the other hand, the work of mining association rules [2] looks for actual patterns in the data that represent some internal relationships. There may be many such relationships in the data that could be discovered, so this approach covers the data in a breadth-first fashion, selecting the set of rules at each iteration and extending them by appending to each a new possible item.

Chapter 6

Conclusions

This thesis presents an approach to information extraction from unstructured, ungrammatical text such as internet classifieds, auction titles, and bulletin board postings. My approach, called "reference-set based extraction" exploits a "reference set" of known entities to aid extraction. By using reference sets for extraction, instead of grammar or structure, my technique relaxes the assumption that posts require structure or grammar, which they may not. To exploit a reference set my approach first finds the matches between the unstructured, ungrammatical text and the reference set. It then uses these matches as clues to aid extraction. This thesis presents two approaches to reference-set based extraction. One is a fully automatic method that can even include an automatic approach to constructing the reference set from the unstructured data itself. The other method is highly accurate and uses machine learning to solve the task.

To reiterate, the contributions of this thesis are as follows:

- An automatic matching and extraction algorithm that exploits a given reference set (Chapter 2)

- A method that selects the appropriate reference sets from a repository and uses them for extraction and annotation without training data (Chapter 2)

- An automatic method for constructing reference sets from the posts themselves. This algorithm also is able to suggest the number of posts required to automatically construct the reference set sufficiently (Chapter 3)

- An automatic method whereby the machine can decide whether to construct its own reference set, or require a person to do it manually (Chapter 3)

- A method that uses supervised machine learning to perform high-accuracy extraction on posts, even in the face of ambiguity. (Chapter 4)

Reference-set based extraction allows for structured query support and data integration over unstructured, ungrammatical text, which was previously not possible using just keyword search. For example, keyword search does not support aggregate queries. Neither is it easy to perform joins with outside sources using keyword search. Therefore, in the larger context of information integration, reference-set based extraction allows for a whole new set of sources, "posts," to be included in the task of data integration which previously dealt with structured and semi-structured sources such as databases and webpages respectively. By combining posts with more traditional structured sources, new and interesting applications become available. For instance, by linking the cars for sale on Craigslist with the database of car safety ratings from the National Highway Transportation Safety Administration, a user can search for used cars that conform to desired safety ratings.

Further, information extraction from unstructured, ungrammatical sources can aid other fields such as information retrieval, knowledge bases, and the Semantic Web. In information retrieval, Web crawlers can better index pages by classifying the higher level class of the page. For instance, by using reference-set based extraction on the Craiglist car ads, the Web crawler could determine that the class of items referred to on the web-page are cars, since extraction used a set of Cars for the reference set. In turn, if a user searches for "car classifieds," then the set of car classifieds could be returned even if it never mentions the keyword "car" because the reference-set used for extraction makes it clear that the set of classifieds is about the topic of cars.

Along these lines, the algorithm for automatically constructing reference sets can be used to flesh out ontologies and taxonomies for knowledge bases. Further, reference-set based extraction might be a tool that can aid in the ontology alignment problem. For example, two ontologies of hotels might not have enough information for alignment. It might be that one set has star ratings and names, all of which are unique within the set, and another ontology has a name and location, which are all unique within the set, but the names individually are not unique within the set. In this case, each reference set could be used to find matches to a set of posts about hotels. Then, using transitivity across the matches between the ontologies and the posts, it might be possible to discover how to join the ontologies together in a unified set.

Lastly, one bottleneck for the Semantic Web is marking up the entities on the Web with the tags defined by the description logics. However, if a reference set is linked to an RDF description of the attributes, a bot can use reference-set based extraction

to annotate the vast amount of posts on the Web, helping to realize the vision of the Semantic Web.

There are a number of future directions that research on this topic can follow. First, as stated in Chapter 3, there are certain cases where the automatic method of constructing a reference set is not appropriate. One of these occurs when textual characteristics of the posts make it difficult to automatically construct the reference set. One future topic of research is a more robust and accurate method for automatically constructing reference sets from data when the data does not fit the criteria for automatic creation. This is a larger new topic that may involve combining the automatic construction technique in this thesis with techniques that leverage the entire Web for extracting attributes for entities. Along these lines, in certain cases it may simply not be possible for an automatic method to discover a reference set. In this problem, a very weakly supervised approach might be more appropriate where a user is involved in the reference set creation. However, this user involvement should be limited, such as only providing seed examples of reference set tuples, sets of Web pages outside of the posts for use in the automatic construction, or helping the system prune errant reference set tuples.

Another topic of future research involves increasing the scope of reference-set based extraction beyond unstructured, ungrammatical text. The use of reference sets should be helpful for other topics such as Named-Entity Recognition and for linking larger documents with reference sets. This would allow for query and integration of these larger documents, which is sometimes called information fusion. Another interesting future direction is to allow the mining of synonyms and related string transformations, and

then figuring out how to use these in the context of reference-set based extraction in an automatic and scalable fashion.

Beyond the scope of improving extraction, there are numerous lines of future research that involve using the extracted data, which is output by my reference-set based extraction approach. One future line of research involves using the extracted data for data mining to help users make informed decisions about aspects such as the prices of items for sale, the quality of items for sale, or the general opinions regarding items for sale. For instance, after discovering a reference set of laptops, it would be useful for the system to gather as much other data about each laptop as possible, such as the average price and average review for the computer. It could do this dynamically, based on the newly extracted attributes. Once this is done a user can make a much more informed decision about whether they are looking at a good deal or not. Another way to use the data post extraction is to create sale portals for e-Commerce. For instance, by aggregating all of the used cars on the Web from all of the sites in one area where aggregate joins can be made, users can glean many insights such as who has the best inventory, which source gives the best prices, etc.

In this thesis I describe reference-set based extraction which allows users and computers to make sense of vast amounts of data that were previously only accessible through limited keyword search. With so much information to process, hopefully computer science research will continue in the direction of making sense of the World Wide Web, the single most remarkable source of knowledge.

Bibliography

[1] Eugene Agichtein and Venkatesh Ganti. Mining reference tables for automatic text segmentation. In *the Proceedings of the 10th ACM Conference on Knowledge Discovery and Data Mining*, pages 20 – 29. ACM Press, 2004.

[2] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 207–216. ACM Press, 1993.

[3] Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for record linkage. In *Proceedings of the 9th ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Identification*, pages 25–27, 2003.

[4] Kedar Bellare and Andrew McCallum. Learning extractors from unlabeled text using relevant databases. In *Proceedings of the AAAI Workshop on Information Integration on the Web*, pages 10–16, 2007.

[5] Mikhail Bilenko, Beena Kamath, and Raymond J. Mooney. Adaptive blocking: Learning to scale up record linkage and clustering. In *Proceedings of the 6th IEEE International Conference on Data Mining*, pages 87–96, 2006.

[6] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining*, pages 39–48. ACM Press, 2003.

[7] Vinayak Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. Automatic segmentation of text into structured records. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 175–186. ACM Press, 2001.

[8] Michael J. Cafarella, Doug Downey, Stephen Soderland, and Oren Etzioni. Knowitnow: Fast, scalable information extraction from the web. In *Proceedings of HLT-EMNLP*, pages 563–570. Association for Computational Linguistics, 2005.

[9] Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pages 328–334, 1999.

[10] Venkatesan T. Chakaravarthy, Himanshu Gupta, Prasan Roy, and Mukesh Mohania. Efficiently linking text documents with relevant structured information. In *Proceedings of the International Conference on Very Large Data Bases*, pages 667–678. VLDB Endowment, 2006.

[11] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 313–324. ACM Press, 2003.

[12] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab. Towards the self-annotating web. In *Proceedings of the 13th International Conference on World Wide Web*, pages 462–471. ACM Press, 2004.

[13] Philipp Cimiano, Andreas Hotho, and Steffen Staab. Learning concept hierarchies from text corpora using formal concept analysis. *Journal of Artificial Intelligence Research*, 24:305–339, 2005.

[14] Fabio Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 1251–1256, 2001.

[15] William Cohen and Sunita Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining*, pages 89–98, Seattle, Washington, August 2004. ACM Press.

[16] William W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18(3):288–321, 2000.

[17] William W. Cohen, Pradeep Ravikumar, and Stephen E. Feinberg. A comparison of string metrics for matching names and records. In *Proceedings of the ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Consoliation*, pages 13–18, 2003.

[18] Nick Craswell, Peter Bailey, and David Hawking. Server selection on the world wide web. In *Proceedings of the Conf. on Digital Libraries*, pages 37–46. ACM Press, 2000.

[19] 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. VLDB Endowment., 2001.

[20] S. Dill, N. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proceedings of WWW*, pages 178–186. ACM Press, 2003.

[21] Yihong Ding, David W. Embley, and Stephen W. Liddle. Automatic creation and simplified querying of semantic web content: An approach based on information-extraction ontologies. In *Proceedings of the Asian Semantic Web Conference*, pages 400–414, 2006.

[22] Alexiei Dingli, Fabio Ciravegna, and Yorick Wilks. Automatic semantic annotation using unsupervised information extraction and integration. In *Proceedings of the K-CAP Workshop on Knowledge Markup and Semantic Annotation*, 2003.

[23] Georges Dupret and Benjamin Piwowarski. Principal components for automatic term hierarchy building. In *SPIRE*, pages 37–48, 2006.

[24] Mohamed G. Elfeky, Vassilios S. Verykios, and Ahmed K. Elmagarmid. TAILOR: A record linkage toolbox. In *Proceedings of 18th International Conference on Data Engineering*, pages 17–28, 2002.

[25] 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.

[26] Ivan P. Fellegi and Alan B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[27] Siegfried Handschuh, Steffen Staab, and Fabio Ciravegna. S-cream - semi-automatic creation of metadata. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management*, pages 165–184. Springer Verlag, 2002.

[28] Hany Hassan, Ahmed Hassan, and Ossama Emam. Unsupervised information extraction approach using graph mutual reinforcement. In *the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 501–508. Association for Computational Linguistics, 2006.

[29] Mauricio A. Hernandez and Salvatore J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[30] Matthew A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 89:414–420, 1989.

[31] Thorsten Joachims. *Advances in Kernel Methods - Support Vector Learning*, chapter 11: Making large-Scale SVM Learning Practical. MIT-Press, 1999.

[32] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann, 2001.

[33] Dawn Lawrie, W. Bruce Croft, and Arnold Rosenberg. Finding topic words for hierarchical summarization. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 349–357, New York, NY, USA, 2001. ACM.

[34] Mong-Li Lee, Tok Wang Ling, Hongjun Lu, and Yee Teng Ko. Cleansing data for mining and warehousing. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications*, pages 751–760. Springer-Verlag, 1999.

[35] Kristina Lerman, Cenk Gazen, Steven Minton, and Craig A. Knoblock. Populating the semantic web. In *Proceedings of the AAAI Workshop on Advances in Text Extraction and Mining*, 2004.

[36] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *English translation in Soviet Physics Doklady*, 10(8):707–710, 1966.

[37] Alon Levy. Logic-based techniques in data integration. In Jack Minker, editor, *Logic Based Artificial Intelligence*, pages 575–595. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[38] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd VLDB Conference*, pages 251–262, Bombay, India, 1996. Morgan Kaufmann Publishers Inc.

[39] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[40] Masoud Makrehchi and Mohamed S. Kamel. Automatic taxonomy extraction using google and term dependency. In *IEEE/WIC/ACM International Conference on Web Intelligence*, 2007.

[41] Imran R. Mansuri and Sunita Sarawagi. Integrating unstructured data into relational databases. In *Proceedings of the International Conference on Data Engineering*, page 29. IEEE Computer Society, 2006.

[42] Andrew McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[43] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the 6th ACM SIGKDD*, pages 169–178, 2000.

[44] Matthew Michelson and Craig A. Knoblock. Semantic annotation of unstructured and ungrammatical text. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1091–1098, 2005.

[45] Matthew Michelson and Craig A. Knoblock. Learning blocking schemes for record linkage. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 2006.

[46] Steven N. Minton, Claude Nanjo, Craig A. Knoblock, Martin Michalowski, and Matthew Michelson. A heterogeneous field matching method for record linkage. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM-05)*, 2005.

[47] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[48] Hans-Michael Müller and Eimear E. Kenny Paul W. Sternberg. Textpresso: An ontology-based information retrieval and extraction system for biological literature. *PLoS Biology*, 2(11), 2004.

[49] Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[50] H. B. Newcombe. Record linkage: The design of efficient systems for linking records into individual and family histories. *American Journal of Human Genetics*, 19(3):335–359, 1967.

[51] Marius Paşca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain. Organizing and searching the world wide web of facts - step one: the one-million fact extraction challenge. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, pages 1400–1405. AAAI Press, 2006.

[52] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[53] Lawrence Reeve and Hyoil Han. Survey of semantic annotation platforms. In *Proceedings of ACM Symposium on Applied Computing*, pages 1634–1638. ACM Press, 2005.

[54] Mark Sanderson and Bruce Croft. Deriving concept hierarchies from text. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 206–213, New York, NY, USA, 1999. ACM.

[55] Patrick Schmitz. Inducing ontology from flickr tags. In *Workshop on Collaborative Web Tagging*, 2006.

[56] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

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

[58] Snehal Thakkar, Jose Luis Ambite, and Craig A. Knoblock. Composing, optimizing, and executing plans for bioinformatics web services. *The VLDB Journal, Special Issue on Data Management, Analysis, and Mining for the Life Sciences*, 14(3):330–353, 2005.

[59] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the 21st International Conference on Machine Learning*, page 104. ACM Press, 2004.

[60] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[61] Maria Vargas-Vera, Enrico Motta, John Domingue, Mattia Lanzoni, Arthur Stutt, and Fabio Ciravegna. MnM: Ontology driven semi-automatic and automatic support for semantic markup. In *Proceedings of the 13th International Conference on Knowledge Engineering and Management*, pages 213–221, 2002.

[62] William E. Winkler. The state of record linkage and current research problems. Technical report, U.S. Census Bureau, 1999.

[63] William E. Winkler and Yves Thibaudeau. An application of the fellegi-sunter model of record linkage to the 1990 U.S. Decennial Census. Technical report, Statistical Research Report Series RR91/09 U.S. Bureau of the Census, 1991.

[64] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 334–342. ACM Press, 2001.