BUILDING QUERYABLE DATASETS FROM UNGRAMMATICAL AND UNSTRUCTURED SOURCES

by

Matthew Jeremy Michelson

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

August 2005

Acknowledgements

I would especially like to thank Craig Knoblock for chairing my Master's thesis committee. Also, I am grateful to Kevin Knight and Cyrus Shahabi for forming the rest of my committee. Thanks to William Cohen, Andrew McCallum and Fabio Ciravegna for allowing public use of their systems and code. I am indebted to Kristina Lerman and Snehal Thakkar for their comments on this work.

This research is based upon work supported in part by the National Science Foundation under Award No. IIS-0324955, in part by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010, in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-1-0504, in part by the Air Force Office of Scientific Research under grant number FA9550-04-1-0105, and in part by the United States Air Force under contract number F49620-02-C-0103.

The U.S. Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding 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.

Contents

List Of Tables

List Of Figures

Abstract

For agents to act on behalf of users, they will have to query the vast amounts of textual data on the internet. However, much of this text cannot be queried because it is neither grammatical nor formally structured enough to support traditional information extraction approaches to annotation. Examples of such text, called "posts," include item descriptions on Ebay or internet classifieds like Craig's list. This work describes an approach to annotating posts by combining record linkage with information extraction. This approach leverages collections of known entities, called "reference sets," by first aligning a post to a member of a reference set, and then exploiting this matched member during information extraction. This thesis compares this extraction approach to more traditional information extraction methods that rely on structural and grammatical characteristics, and it shows that this approach outperforms traditional methods on this type of data.

Chapter 1

Introduction

The envisioned future of the internet involves morphing the current World Wide Web into a vast network of semantically annotated documents called the Semantic Web. Part of this vision includes agents working on the behalf of users, autonomously performing tasks such as booking hotels and making appointments. These agents perform their tasks by deriving knowledge from the semantic annotation included as mark-up in the documents. For instance, an agent could query a number of data sources to find the lowest price for a given hotel and then book a reservation.

However, the ability to query data sources derives from the ability to annotate the sources, and annotating certain data sources is a difficult process. Specifically, semantic annotation usually relies on information extraction techniques to identify the attributes in the text for labeling. Information extraction, in turn, relies on grammatical and structural characteristics of the text to identify the attributes to extract. Yet, there are many data sources on the internet that are useful to query, but the textual data contained in them makes annotation difficult. In these cases the text is not structured enough for wrapper extraction technologies such as Stalker [25] or RoadRunner [12]. Nor is it grammatical

enough to exploit Natural Language Processing (NLP) based extraction techniques such as those used in Whisk [28] or Rapier [5]. With such textual data traditional information extraction techniques for annotation are not applicable because of the lack of structure.

The ungrammatical and unstructured textual data of these sources are called "posts." Examples of "posts" include the text of EBay posts, internet classifieds like Craig's list, bulletin boards such as Bidding For Travel,[1] or even the summary text below the hyperlinks returned after querying Google. Figure 1.1 shows an example post from Bidding For Travel, along with the desired semantic annotation added to this post to make it queryable.

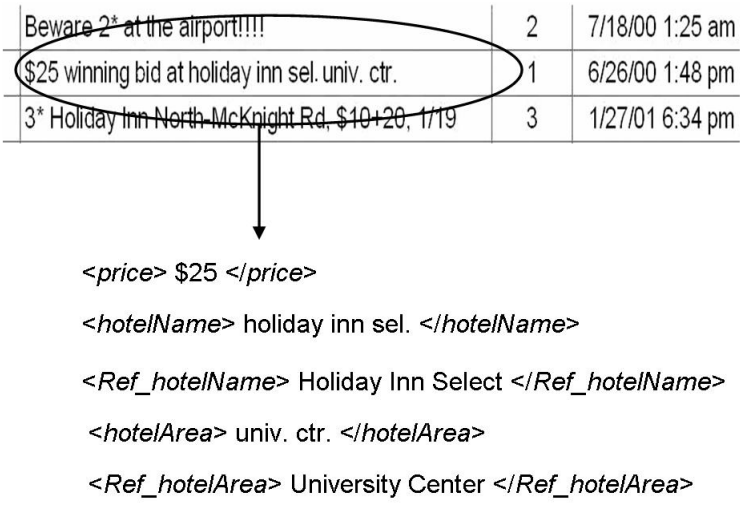

Figure 1.1: A post from Bidding For Travel

The lack of grammar and structure in posts can be overcome by incorporating knowledge into information extraction. This extra knowledge, called "reference sets," consists

[1] www.biddingfortravel.com

of collections of known entities with the associated, common attributes. A reference set can be an online (or offline) set of reference documents, such as the CIA World Fact Book.[2] It can also be an online (or offline) database, such as the Comics Price Guide.[3] With the Semantic Web one can envision building reference sets from the numerous ontologies that already exist. Using standardized ontologies to build reference sets allows a consensus agreement upon reference set values, which implies higher reliability for these reference sets over others that might exist as one expert's opinion. Continuing with the hotel example from Figure 1.1, assume there is an ontology of U.S. hotels used to construct a reference set with the following attributes: city, state, star rating, hotel name, area name, etc.

To use reference sets for semantic annotation this work exploits the attributes in the reference set to determine the attributes from the post that can be extracted. To do this, the first step finds the best matching member of the reference set for the post. This is called the "record linkage" or "alignment" step. This record linkage step provides the reference set values to look for in the post. During the information extraction step that follows, the parts of the post are extracted that best match the attribute values from the reference set member chosen during the record linkage step. Once extracted, the parts of the post are labeled with the attribute type (schema label) of the reference set attribute they match.

Reference sets are further exploited by including annotation for the values that come from the matching reference set member during the record linkage step. Since attribute

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

values differ across posts, these reference attributes provide a set of standard values for querying the dataset. Also, by including attributes from the matching reference set member, the alignment can provide annotation for attributes that were not included by the user in the post.

In addition to annotation from the reference set, this technique also annotates attributes in the post that are identifiable, but not provided by the reference sets. Examples of such attributes include prices or dates. In many cases, such attributes would compose huge reference sets. Thus it is easier to employ more traditional extraction methods since these types of attributes have consistent characteristics that can be exploited.

As a running example, consider Figure 1.2 which illustrates the approach of this thesis on the circled post of Figure 1.1. For purposes of exposition, the reference set shown only has 2 attributes: hotel name and hotel area. The example post matches the reference set member with the hotel name of "Holiday Inn Select" and the hotel area of "University Center." Using this match the algorithm labels the tokens "univ. ctr." of the post as the "Hotel Area," since they match the hotel area attribute of the matching reference set record. Similarly, "holiday inn sel." is labeled as the Hotel Name. More generally, the technique annotates all of the attributes in the post that match those of the reference set. Figure 1.2 also shows the annotation of a price and the annotation that comes from the attributes of the reference set, shown as "Ref_hotel..." This thesis extends my previous work on annotating ungrammatical and unstructured sources [24] by providing additional details about the algorithms and presenting more experimental results. The overall approach has two steps, the alignment step and the extraction step. Chapters 2 and 3 present each of these steps in detail. Chapter 4 focuses on the experimental

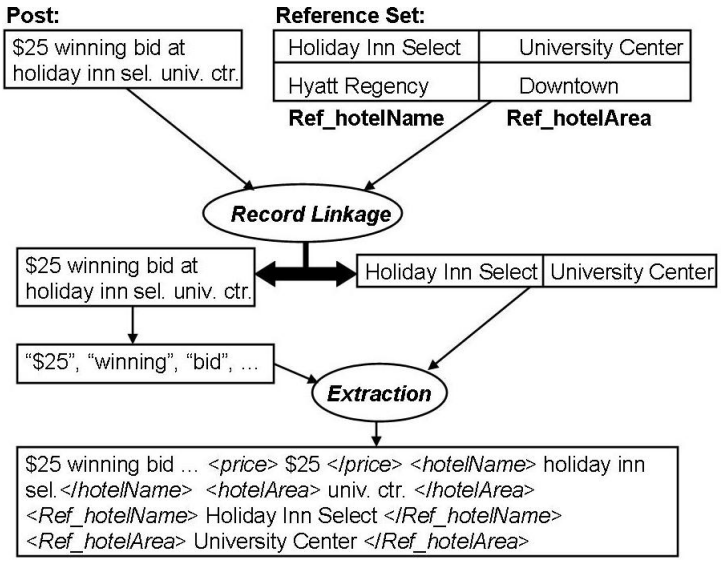

Figure 1.2: Annotation Algorithm

evaluation of the alignment and extraction steps detailed in Chapters 2 and 3. Beyond the experimental performance there are other items worth noting. Chapter 5 presents these issues as the Discussion. Chapter 6 describes the related work in the field and Chapter 7 finishes the thesis with my conclusions and future directions.

Chapter 2

Aligning Posts to a Reference Set

To exploit the reference set attributes for annotation, the system needs to first decide which member of the reference set best matches the post. This alignment not only provides the set of reference set attributes that are leveraged for the information extraction step, but it also annotates the post with the reference set attributes, which allows for querying the data with standard values.

Since it is infeasible to compare the post against all members of the reference set, the alignment step first constructs a set of candidate matches from the reference set. This candidate generation is called "blocking." The goal of blocking is to efficiently whittle down the set of possible matches, while not eliminating any true matches. Blocking is an active field of research [2], and the technique used in this thesis defines a candidate as any member of the reference set that has a common subsequence of letters, called an *n-gram*, with the post.

From this set of candidates one can find the member of the reference set that best matches the current post. That is, one data source's record (the post) must align to a record from the other data source (the reference set candidates). This alignment is called

record linkage [14]. However, the record linkage problem presented in this thesis differs

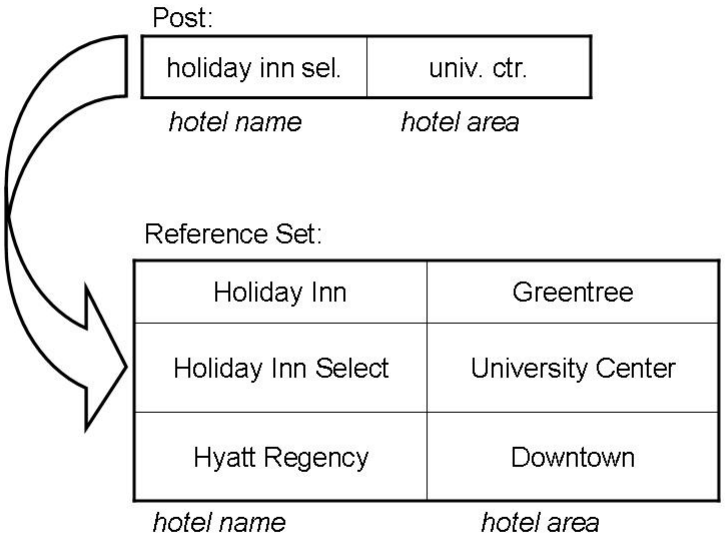

Figure 2.1: The traditional record linkage problem

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, using the example from Figure 1.2, and assuming decomposed attributes, the *hotel name* from the post is compared to the *hotel names* of the reference set. This is also done for the *hotel areas*. The record from the reference set that best matches the post based on the similarities between the attributes would be considered the match. This is represented in Figure 2.1. 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

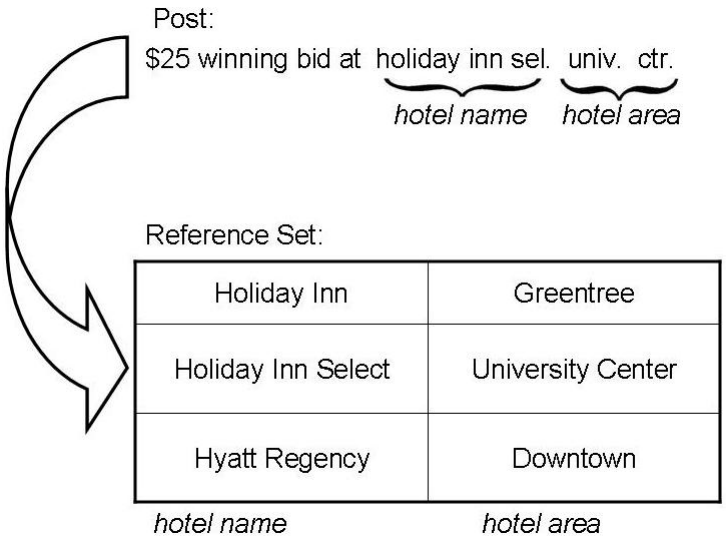

Figure 2.2: The problem of matching a post to the reference set

tokens present in the post. Figure 2.2 depicts this problem. With this type of matching traditional record linkage approaches do not apply.

Instead, the alignment 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. However, by using only the record level similarity it is possible for two candidates 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 2.3. In the figure, the two candidates share the same hotel name with the post. However, the first candidate has an area in

Figure 2.3: Two records with equal record level but different field level similarities

common with the post while the second candidate has a star rating in common. Since both candidates share the same name, and both have another attribute in common, it is possible that they generate the same record level comparison. Yet, a hotel area should be more discriminative than a star rating, since many more hotels share the same star rating than the same hotel area. This difference in individual attributes needs to be reflected.

To discriminate between attributes, the alignment 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 alignment 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 alignment step generates the *RL_scores* vector between the post and all of the attributes concatenated together. To generate field level comparisons, the alignment 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 1.2, the schema has 2 attributes <*Hotel Name, Hotel Area*>. Assuming the current candidate is <"Hyatt", "PIT Airport">, then the V_{RL} looks like:

$$\begin{aligned} V_{RL} = <& RL\_scores(post,\ \text{“Hyatt”}),\\ & RL\_scores(post,\ \text{“PIT Airport”}),\\ & RL\_scores(post,\ \text{“Hyatt PIT Airport”})> \end{aligned}$$

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 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, where as edit distance requires that the order of the tokens match 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 of these different similarity types the *RL_scores* vector is built up of 3 vectors that reflect the each of the different similarity types discussed above. Hence, *RL_scores* is:

$$\begin{aligned} \textit{RL\_scores(post, attribute)} = < & \textit{token\_scores(post, attribute)}, \\ & \textit{edit\_scores(post, attribute)}, \\ & \textit{other\_scores(post, attribute)} > \end{aligned}$$

The vector *token_scores* comprises token level similarity scores. One similarity score included in this vector is the Jensen-Shannon distance between sets of tokens [11]. Jensen-Shannon similarity defines the score between the tokens of two strings, S and T, by defining a distribution of tokens over the strings, called P_S and P_T respectively. These distributions are then used to calculate the similarity between S and T. Before calculating Jensen-Shannon distance, one must define the Kullback-Leibler (KL) divergence which reflects the measure of dissimilarity between two probability mass functions. Given two

probability functions *p(x)* and *q(x)* over some quantity x, the KL divergence between p and q is:

$$KL(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

Another necessary term to define, Q(w), describes the average probability of a token w in the distributions of S and T. That is:

$$Q(w) = \frac{(P_S(w) + P_T(w))}{2}$$

With these definitions, the Jensen-Shannon distance between S and T is defined as:

$$Jensen - Shannon(S,T) = \frac{(KL(P_S||Q) + KL(P_T||Q))}{2}$$

In words, Jensen-Shannon distance is a way to combine the probabilities of common tokens occurring in S and T to generate a sense of similarity between the strings. However, S and T are too sparse of a language model to learn good probability distributions over token occurrence, so smoothing must be incorporated to account for tokens unseen in S and T. This smoothing is akin to including prior knowledge about token occurrence into the probabilities P_S and P_T which can be accomplished using a Dirichlet prior, for example. For this purpose, the vector *token_scores* includes a Jensen-Shannon score with distributions P_S and P_T estimated using a Dirichlet prior with α set to 1. Also included in *token_scores* is another Jensen-Shannon score with the probabilities smoothed using a Jelenik-Mercer mixture model [34]. The Jelenik-Mercer mixture model is a linear interpolation between the maximum-likelihood probability of a token in the string and the

probability of the token occurring in the corpus of strings, called the collection model. The interpolation is controlled by the mixture coefficient λ, which in this case was set to 0.5, so that each piece of the interpolation was weighted equally.

The *token_scores* also includes the Jaccard similarity between strings S and T [11]. If one considers S and T as a bag of tokens, then Jaccard similarity is the ratio between the number of tokens in common between S and T over the cardinality of the union of tokens in S and T:

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

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 Levenstein distance, which returns the minimum number of operations to turn string S into string T. To generate this minimum number, Levenstein distance associates a cost with each operation, and returns the set of operations that minimizes the total cost. The operation costs are as follows:

Copy character of S into T = 0
Insert single character into T = 1
Delete single character from S = 1
Substitute single character = 1

One extension to the Levenstein distance is the Smith-Waterman distance [27], which also appears in *edit_scores*. Like Levenstein distance, Smith-Waterman is an edit-distance because it generates a similarity score based upon the number of operations needed to

transform string S into string T. However, in the Smith-Waterman algorithm the associated operation costs are not fixed, but are chosen values, and the algorithm maximizes the total cost of the operations. One of the chosen values is a distance function, d, between two characters that may be chosen arbitrarily to reflect a relationship between the characters. For instance, in comparing DNA strands, d might reflect amino acid substitutability. This distance can have different values depending on whether the operation is a copy or a substitution. Perhaps copy receives a positive cost while substitution receives a negative one. The other chosen value, denoted G, is the gap cost that penalizes inserting or deleting characters. Since G is a penalty, its value is negative which leads to the possibility of negative total operational costs up to a certain point in the set of operations. To account for this a "start over" operation is included which sets all negative total costs to 0. Hence, the Smith-Waterman finds the maximized set of operations with the following associated costs:

Copy/Substitute character of S into T = d
Insert single character into T = -G
Delete single character from S = -G
Start over = 0

The last score in the vector *edit_scores* is the Jaro-Winkler similarity [33], which is an extension of the Jaro metric [17]. 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. There are two definitions necessary to computer the Jaro metric. A character from S is said to be "in common" with a character from T if the character from S has the same value as the character from T, and falls within a small neighborhood of indexes around the letter in T. With this definition, S' becomes all of the characters

from S "in common" with T, and vice versa for T'. Now, the "transposition" between s'∈S' and t'∈T' is the index i such that s'≠t'. That is to say, i represents an index in the "in common" strings S' and T' where the letters do not match. Based on these transpositions, $\text{Trans}_{S',T'}$ is defined as half of the number of transpositions between S' and T'. Using these definitions, Jaro metric is:

$$Jaro(S,T) = \frac{1}{3} \cdot \left(\frac{|S'|}{|S|} + \frac{|T'|}{|T|} + \frac{|S'| - Trans_{S',T'}}{|S'|} \right)$$

The extension of Jaro to Jaro-Winkler distance adjusts the weights such that when strings share a match from the first letter for a specified prefix length (i.e. they share a common prefix), the similarity is higher than it would be with just a Jaro metric. The underlying assumption is that certain attributes are more likely similar if they share a certain prefix. This works well for proper nouns, as many attribute values are, but could be problematic for arbitrary words that might share a prefix, such as "*comm*otion" and "*comm*union." For the calculation, the prefix length l is defined as the length of the longest common prefix. The calculation defines that if l is less than four, it is set to four. The Jaro-Winkler equation is defined as:

$$Jaro - Winkler(S,T) = Jaro(S,T) + (l \cdot 0.1 \cdot (1 - Jaro(S,T))$$

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

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

All of the similarities in the *edit_scores* and *token_scores* vector are defined in the SecondString package [11] which was used for the experimental implementation as described in Chapter 4.

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. This vector includes two types of string similarities. 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 [26], 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:

$$\begin{aligned} &\textit{other\_scores(post, attribute)=<Porter-Stemmer(post, attribute),} \\ &\qquad\qquad\qquad\qquad\qquad\qquad \textit{Soundex(post, attribute)>} \end{aligned}$$

Figure 2.4 shows the full composition of V_{RL}, with all of the constituent similarity scores shown.

Once a V_{RL} is constructed for each of the candidates, the alignment 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. Since 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

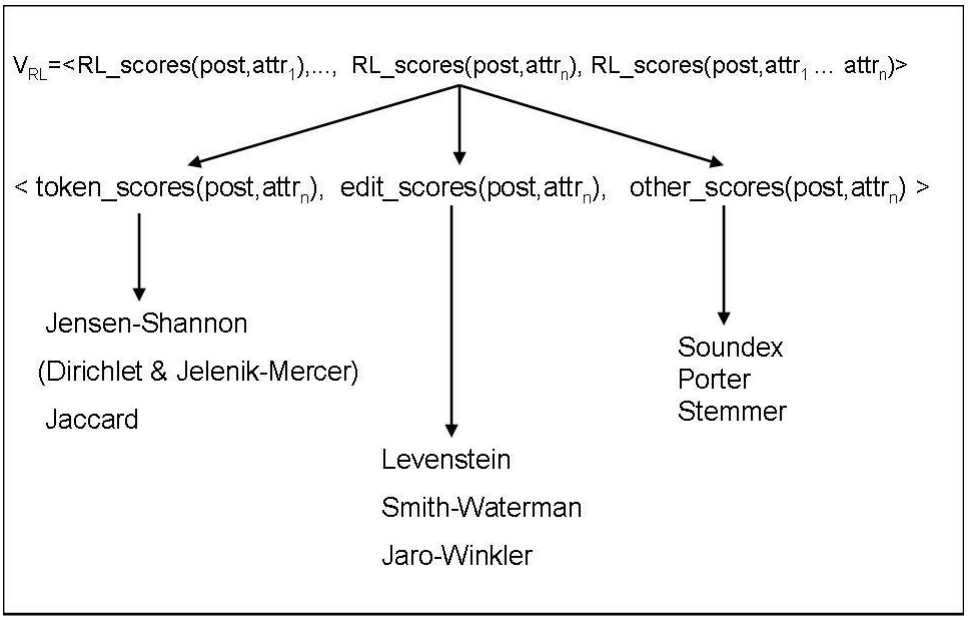

Figure 2.4: The full vector of similarity scores used for record linkage

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}:

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

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

After rescoring they become:

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

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

After rescoring, the alignment step passes each V_{RL} to a Support Vector Machine (SVM) [18] 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.

When 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 along with tags that reflect the schema of the reference set. This is shown in Figure 1.2, by the attributes Ref_hotelName and Ref_hotelArea. Beyond providing a standardized set of values to query the posts, there are other implications of including this annotation that are examined in the Discussion, in Chapter 5.

Chapter 3

Extracting Data from Posts

The algorithm infuses information extraction with extra knowledge, rather than relying on possibly inconsistent characteristics. To garner this extra knowledge, the approach exploits the idea of reference sets by using the attributes from the matching reference set member as a basis for identifying similar attributes in the post. Then, the algorithm can label these extracted values from the post with the schema from the reference set, thus adding annotation based on the extracted values.

To begin the extraction process, the post is broken into tokens. Using the circled post from Figure 1.1 as an example, set of tokens becomes, {"$25", "winning", "bid",...}. Each of these tokens is then scored against each attribute of the record from the reference set that was deemed the match.

To score the tokens, the extraction process builds a vector of scores, V_{IE}. Like the V_{RL} vector of the alignment 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 than V_{RL}. It contains no comparison to the concatenation of all of 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 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 makes traditional extraction methods a good choice for these attributes. This vector is called *common_scores* because the types of characteristics used to extract these attributes are common enough between them to be used for extraction.

Using the example from Figure 1.2, the reference set match is the tuple {"Holiday Inn Select","University Center"}. This match generates the following V_{IE} for the token "univ." of the post:

V_{IE}=*<common_scores("univ."),*
IE_scores("univ.","Holiday Inn Select"),
IE_scores("univ.","University Center")>

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

$$
\begin{aligned}
V_{IE} = <& \textit{common\_scores(token)}, \\
& \textit{IE\_scores(token, attribute}_1\textit{)}, \\
& \textit{IE\_scores(token, attribute}_2\textit{)} \\
& \ldots, \\
& \textit{IE\_scores(token, attribute}_n\textit{)}>
\end{aligned}
$$

Each V_{IE} is then passed to a multiclass SVM [30] trained to give it an attribute type label, such as *hotel name*, *hotel area*, or *price*. Intuitively, similar attribute types should have similar V_{IE} vectors. The hotel names should generally have high scores against the hotel name attribute of the reference set, and small scores against the other attributes. 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 that should not be annotated, 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 a difficult task 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.

However, labeling each token in isolation introduces the possibility 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 *hotel area*. This leads to noisy tokens within the whole *hotel area* annotation. Therefore, labeling tokens individually gives an approximation of the data to be extracted.

The extraction approach can overcome the problems of labeling the tokens in isolation by comparing the whole extracted field to its analogue reference set attribute. Once all of the 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. Using the circled post from Figure 1.1, assume the phrase "holiday inn sel. univ. ctr." was "holiday inn sel. ***in*** univ. ctr.". In this case, the system might extract "holiday inn sel. in" as the *hotel name*. In isolation, the token "in" could be the "inn" of a hotel name. Comparing this extracted hotel name to the reference attribute, "Holiday Inn Select," generates a Jaccard similarity of 0.4 and a Jaro-Winkler score of 0.87.

Next, the cleaning method goes through the extracted attribute, removing one token at a time and calculating new Jaccard and Jaro-Winkler similarities. If both new scores are higher than the baselines, that token becomes a removal candidate. Once all of the tokens are processed in this way, the removal candidate with the highest scores is removed, and the whole process is repeated. The scores derived using the removed token then become the new baseline to compare against. The process ends when there are no more tokens that yield improved scores over the baselines.

In the example, the cleaning method finds that "in" is a removal candidate since removing this token from the extracted hotel name yields a Jaccard score of 0.5 and a Jaro-Winkler score of 0.92, which are both higher than the baseline scores. Since it has the highest scores after trying each token in the iteration, it is removed and the baseline scores update. Then, since none of the remaining tokens provide improved scores, the process terminates, yielding a more accurate attribute value. This process is shown graphically in Figure 3.1, and the pseudocode for the algorithm is shown in Figure 3.2.

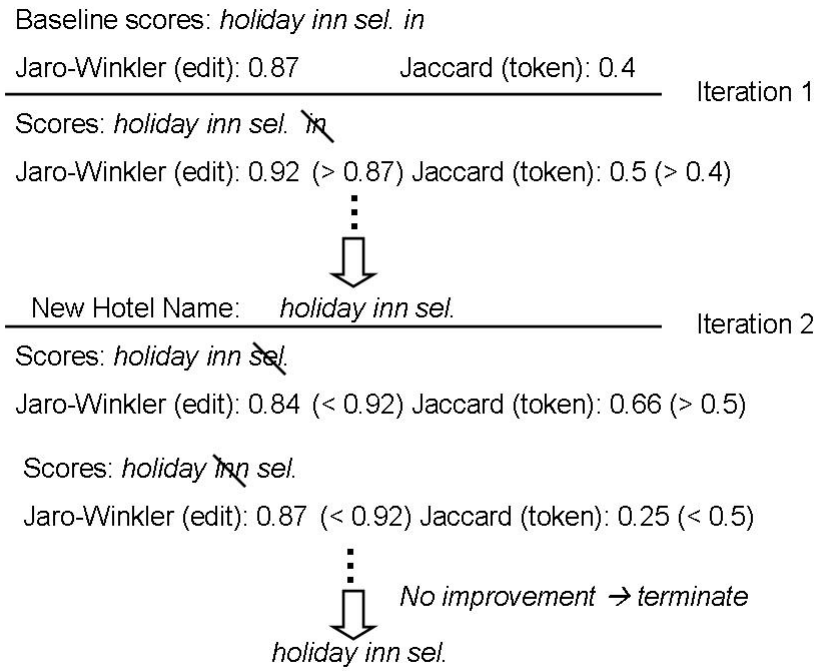

Figure 3.1: Improving extraction accuracy with reference set attributes

Aside from increasing the accuracy of extraction, this approach can also disambiguate possible attribute labels. For instance, a token might be both a *hotel name* and a *hotel area*, but not both at the same time. The token "airport," for example, could be part of

Algorithm 3.0.1: 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 $C = null$
- **exit**

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

Figure 3.2: Algorithm to clean a whole extracted attribute

a *hotel name* or a *hotel area*. In this case, the extraction process could label it as both attribute types, and the above approach would remove the token from the attribute that it is not. However, the current implementation assigns only one label per token, so this disambiguation technique was not tested.

The whole extraction process takes a token of the text, creates the V_{IE} and passes this to the SVM which generates a label for the token. Then each field is cleaned up and added as semantic annotation for the post. This produces the output shown at the end of Figure 1.2. The whole procedure is produced graphically in Figure 3.3.

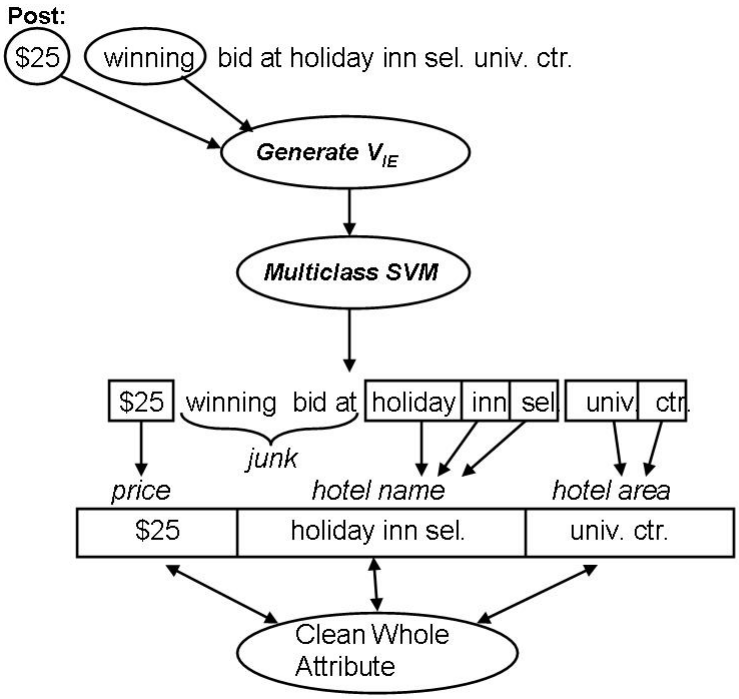

Figure 3.3: Extraction process for attributes

Chapter 4

Results

A system named Phoebus was built to experimentally validate the overall algorithm for semantic annotation. Specifically, Phoebus tests the technique's accuracy in both the alignment step and the extraction step. The experimental data, comes from three domains of posts: *hotels*, *comic books*, and *cars*.

The data from the *hotel* domain contains the attributes hotel name, hotel area, star rating, price and dates, which are extracted for annotation. This data comes from the Bidding For Travel website[1] which is a forum where users share successful bids for Priceline on items such as airline tickets and hotel rates. The experimental data is limited to postings about hotel rates in Sacramento, San Diego and Pittsburgh, which compose a data set with 1125 posts. The reference set comes from the Bidding For Travel hotel guides, which are special posts listing all of the hotels ever posted about in a given area. These special posts provide hotel names, hotel areas and star ratings, which are the reference set attributes. This reference set contains 132 records.

[1] www.biddingfortravel.com

The experimental data for the *comic* domain comes from posts on EBay. To generate this data set, Ebay was searched by the keywords "Incredible Hulk" and "Fantastic Four" within the comic books for sale portion of the website (Although, this returned some items that are not comics, such as t–shirts and some sets of comics not limited to those searched for, thus making the problem more difficult.) The returned records contain the attributes comic title, issue number, price, condition, publisher, publication year and the description, which are extracted for annotation. (Note: the description is a few word description commonly associated with a comic book, such as *1st appearance the Rhino.*) The total number of posts in this data set is 776. The *comic* domain reference set uses data from the Comics Price Guide,[2] which lists all of the Incredible Hulk and Fantastic Four comics, as well as all of the possible comic book conditions. This yields two reference sets, one for the conditions and one for the comics. The conditions reference set has 20 members. The Comic reference set has the attributes title, issue number, description, and publisher and contains 918 records.

The *cars* data consists of posts made to Craig's list[3] regarding cars for sale. This dataset came from the first 10 pages of posts on the cars-for-sale pages for the cities of Los Angeles, San Francisco and New York City. Unfortunately, the reference set did not cover all of the cars mentioned in the posts, so those posts that referred to missing members of the reference set were removed. This is one limitation of the algorithm; there needs to be a reference member for the approach to work. However, this problem is easily overcome by incorporating multiple reference sets for full coverage. If a post did not

[2]http://www.comicspriceguide.com/
[3]www.craigslist.com

refer to any reference set member, for example, a seller of automotive services or CarFax reports, it was not removed from the reference set. This resulted in 855 posts, which were annotated with the car make, the car model, the car trim, the year and the price. The reference set for this domain came from the Edmunds website.[4] The reference set contains Acura, Honda, Hyundai, Isuzu, Lexus, Mazda, Mitsubishi, Nissan, Subaru, Suzuki and Toyota cars and Sport-Utility-Vehicles from 1990 to 2003. This data set contains the attributes car make, car model, trim and year and has 3171 records.

There are two small experimental differences with the *cars* domain from the other two. The blocking method for the other two domains generated candidates based on common n-grams. In the *cars* domain this produced huge numbers of candidates, invalidating this as a blocking method for this data. Instead, Phoebus generated candidates using common tokens between the post and just the model and year attributes of the reference set, rather than all of the attributes. This reduced the number of candidates for each post by an order of magnitude, which proved a much better blocking technique.

In fact, blocking with unstructured and ungrammatical text is an interesting problem. Much like with record linkage, the lack of decomposed attributes and the extraneous junk tokens make blocking using more sophisticated methods difficult. This is why simple blocking was done using the set or subset of tokens with common tokens or n-grams. Also, if such an annotation scheme is to annotate posts in real time as users enter them, the blocking method must also be fast. Real-time blocking has not been considered much in lieu of blocking methods that spend lots of time preprocessing the data so as to block

[4] www.edmunds.com

as efficiently as possible. Real-time blocking on data without decomposed attributes and with junk tokens is an area of future research.

Also, a strict 1-1 relationship between the post and reference set was enforced for the other two domains, but not the cars domain. As described in Chapter 2, Phoebus relaxed 1-1 relationship to form a 1-n relationship between the posts and the reference set. Since many of the records do not contain enough attributes to discriminate a single best reference member, such as posts that contain just a model and a year, any reference set member with the matching year and model could be accurately used for extraction. Also, the more restrictive blocking scheme eliminates most of the posts with no matches because they do not generate any candidates. So this relaxation on the 1-to-1 mapping does not introduce problems in that regard, as it might have in the other domains, where the more relaxed blocking methods might errantly link a post that relates to no member of the reference set to a few candidates.

For the experiments, posts in each domain are split into two folds, one for training and one for testing, where the training fold is 30% of the total posts, and the testing fold is the remaining 70%. The experiments are performed 10 times and the average results for these 10 trials are reported.

4.0.1 Alignment Results

Since this alignment approach hinges on leveraging reference sets, it becomes necessary to show the alignment step performs well. To measure this accuracy, the experiments employ the usual record linkage statistics:

$$Precision = \frac{\#CorrectMatches}{\#TotalMatchesMade}$$

$$Recall = \frac{\#CorrectMatches}{\#PossibleMatches}$$

$$F - Measure = \frac{2 * Precision * Recall}{Precison + Recall}$$

The record linkage approach in this thesis is compared to WHIRL [10]. 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 as 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 (70% of the posts) and the reference set with the attributes concatenated together 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 that has

matches from the join, the reference set member with the highest similarity score is called its match.

The record linkage results for both Phoebus and WHIRL are shown in Table 4.1 and all results are statistically significant using a two-tailed paired t-test with α=0.05. 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.

	Prec.	Recall	F-measure
Hotel			
Phoebus	93.60	91.79	**92.68**
WHIRL	83.52	83.61	83.13
Comic			
Phoebus	93.24	84.48	**88.64**
WHIRL	73.89	81.63	77.57
Cars			
Phoebus	93.15	99.57	**96.53**
WHIRL	75.18	40.46	51.86

Table 4.1: Record linkage results

4.0.2 Extraction Results

The extraction algorithm is also experimentally validated and compared against two other information extraction methods.

First, the experiments compare Phoebus against Conditional Random Fields (CRF) [19]. A Conditional Random Field is a probabilistic model that can label and segment data. In labeling tasks, such as Part-of-Speech tagging, CRFs outperform Hidden Markov

Models and Maximum-Entropy Markov Models and present a strong comparison representing these types of probabilistic graphical models [19]. CRFs have also been used effectively for information extraction. For instance, CRFs have been used to combine information extraction and coreference resolution with good results [32]. These experiments use the Simple Tagger implementation of CRFs from the MALLET[22] suite of text processing tools.

Second, the experiments also compare Phoebus to Natural Language Processing (NLP) based extraction techniques. Since the posts are ungrammatical and have unreliable lexical characteristics, these NLP based systems are not expected to do as well on this type of data. The Amilcare [8] system, which uses shallow NLP for extraction, has been shown to outperform other symbolic systems in extraction tasks, and is used as the other system to compare against. For the experiments, Amilcare receives the reference data as gazetteers. Both Simple Tagger and Amilcare are used with default settings.

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 hotel 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*.

The extraction results are presented using Precision, Recall and F-Measure as defined above. Tables 4.2, 4.3 and 4.4 show the results of correctly labeling the tokens within the posts with the correct attribute label for the Hotel, Comic and Cars domains, respectively. Attributes in *italics* are attributes that exist in the reference set. The column Freq shows the average number of tokens that have the associated label.

Table 4.5 shows the results reported for all possible tokens, which is a weighted average, since some attribute types are more frequent than others. Also included in Table 4.5 are "field level" summary results. Field level results regard a piece of extracted information as correctly labeled only if all of the tokens that should be present are, and there are no extraneous tokens. In this sense, it is a harsh, all or nothing metric, but it is a good measure of how useful a technique would really be.

The F-Measures were tested for statistical significance using a two-tailed paired t-test with α=0.05. In Table 4.2 the only F-Measure difference that was not significant was the *Star* attribute between Phoebus and Simple Tagger. In Table 4.3 the F-Measures for Price were not significant between Phoebus and Simple Tagger and between Phoebus and Amilcare. In Table 4.4 the Recall for Make was not significant between Phoebus and Amilcare, the Recall for Year was not significant between Phoebus and Simple Tagger, and the Precision for Price was not significant between Phoebus and Simple Tagger. In Table 4.5 all differences in F-Measure proved statistically significant.

Phoebus outperforms the other systems on almost all attributes, and for all summary results. The results are most striking in the cars domain, where some of the attributes, such as trim and model were very difficult for the systems that lacked the extra knowledge provided by the reference set.

Hotel					
		Prec.	Recall	F-Measure	Freq
Area	Phoebus	89.25	87.5	**88.28**	809.7
	Simple Tagger	92.28	81.24	86.39	
	Amilcare	74.20	78.16	76.04	
Date	Phoebus	87.45	90.62	**88.99**	751.9
	Simple Tagger	70.23	81.58	75.47	
	Amilcare	93.27	81.74	86.94	
Name	Phoebus	94.23	91.85	93.02	1873.9
	Simple Tagger	93.28	93.82	**93.54**	
	Amilcare	83.61	90.49	86.90	
Price	Phoebus	98.68	92.58	**95.53**	850.1
	Simple Tagger	75.93	85.93	80.61	
	Amilcare	89.66	82.68	85.86	
Star	Phoebus	97.94	96.61	**97.84**	766.4
	Simple Tagger	97.16	97.52	97.34	
	Amilcare	96.50	92.26	94.27	

Table 4.2: Extraction results: Hotel domain

Comic					
		Prec.	Recall	F-Measure	Freq
Condition	Phoebus	91.80	84.56	**88.01**	410.3
	Simple Tagger	78.11	77.76	77.80	
	Amilcare	79.18	67.74	72.80	
Descript.	Phoebus	69.21	51.50	59.00	504.0
	Simple Tagger	62.25	79.85	**69.86**	
	Amilcare	55.14	58.46	56.39	
Issue	Phoebus	93.73	86.18	**89.79**	669.9
	Simple Tagger	86.97	85.99	86.43	
	Amilcare	88.58	77.68	82.67	
Price	Phoebus	80.00	60.27	**68.46**	10.7
	Simple Tagger	84.44	44.24	55.77	
	Amilcare	60.00	34.75	43.54	
Publisher	Phoebus	83.81	95.08	**89.07**	61.1
	Simple Tagger	88.54	78.31	82.83	
	Amilcare	90.82	70.48	79.73	
Title	Phoebus	97.06	89.90	93.34	1191.1
	Simple Tagger	97.54	96.63	**97.07**	
	Amilcare	96.32	93.77	94.98	
Year	Phoebus	98.81	77.60	**84.92**	120.9
	Simple Tagger	87.07	51.05	64.24	
	Amilcare	86.82	72.47	78.79	

Table 4.3: Extraction results: Comic domain

Cars					
		Prec.	Recall	F-Measure	Freq
Make	Phoebus	99.96	97.53	**98.73**	459.4
	Simple Tagger	95.66	86.01	90.56	
	Amilcare	92.34	96.82	94.51	
Model	Phoebus	98.35	94.70	**96.49**	514.2
	Simple Tagger	94.25	79.57	86.28	
	Amilcare	83.71	76.18	79.73	
Trim	Phoebus	91.85	73.36	**81.54**	482.6
	Simple Tagger	84.31	66.68	74.25	
	Amilcare	66.98	58.47	62.33	
Year	Phoebus	97.68	92.10	**94.79**	474.1
	Simple Tagger	79.91	91.47	85.27	
	Amilcare	92.73	85.96	89.18	
Price	Phoebus	97.24	97.12	**97.18**	489.4
	Simple Tagger	98.19	83.91	90.49	
	Amilcare	90.90	91.11	90.93	

Table 4.4: Extraction results: Cars domain

	Hotel					
	Token level			Field level		
	Prec.	Recall	F-Mes.	Prec.	Recall	F-Mes.
Phoebus	93.60	91.79	**92.68**	87.44	85.59	**86.51**
Simple Tagger	86.49	89.13	87.79	79.19	77.23	78.20
Amilcare	86.12	86.14	86.11	85.04	78.94	81.88
	Comic					
	Token level			Field level		
	Prec.	Recall	F-Mes.	Prec.	Recall	F-Mes.
Phoebus	93.24	84.48	**88.64**	81.73	80.84	**81.28**
Simple Tagger	84.41	86.04	85.43	78.05	74.02	75.98
Amilcare	87.66	81.22	84.29	90.40	72.56	80.50
	Cars					
	Token level			Field level		
	Prec.	Recall	F-Mes.	Prec.	Recall	F-Mes.
Phoebus	97.20	92.22	**94.65**	92.67	90.63	**91.64**
Simple Tagger	89.80	81.49	85.44	86.49	80.79	83.54
Amilcare	85.73	81.53	83.58	87.02	79.28	82.92

Table 4.5: Summary extraction results

There were three attributes where Phoebus was outperformed, and two of these warrant remarks. (The hotel name was so similar it is ignored.) On Comic titles, Phoebus had a much lower recall because it was unable to extract parts of titles that were not in the reference set. Consider the post, "The incredible hulk and Wolverine #1 Wendigo". In this post, Phoebus extracts "The incredible hulk," but the actual title is "The incredible hulk and Wolverine." In this case, the limited reference set hindered Phoebus, but this could be corrected by including more reference set data, such as other comic book price guides.

The Comic description is the other attribute where Simple Tagger outperformed Phoebus. Simple Tagger learned that for the most part, there is an internal structure to descriptions, such that they are almost never broken up in the middle. For instance, many descriptions go from the token "1st" to a few words after, with nothing interrupting them in the middle. Simple Tagger, then, would label all of these tokens as a description. However, lots of times it labeled too many. This way, it had a very high recall for description, by labeling so much data, but it suffered in other categories by labeling things such as conditions as descriptions too.

Phoebus had the highest level of precision for Comic descriptions, but it had a very low recall because it ignored many of the description tokens. Part of this problem stemmed from classifying tokens individually. Since many of the description tokens were difficult to classify from a single token perspective, they were ignored as junk.

The overall performance of Phoebus validates this approach to semantic annotation. 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 that would contain posts: the auction sites, internet classifieds and forum/bulletin boards.

Chapter 5

Discussion

One drawback when using supervised learning systems is the cost to label training data. However, the entire algorithm generalizes well and can perform well with little training data. To validate this point, consider Phoebus trained on 10% of the posts and tested on the other 90% and compare these results to those when the system is trained with 30% of the data. Tables 5.1 and 5.2 present the token and field level summary extraction results for both training set sizes. In all domains, the results are similar when the system is trained with only 10% of the data as compared to training with 30%. As mentioned previously, the *hotel* domain contains 1125 posts, the *comic* domain contains 776 posts and the *cars* domain has 855 posts.

	Prec.	Recall	F-measure
Hotel (30%)	93.60	91.79	92.68
Hotel (10%)	93.66	90.93	92.27
Comic (30%)	93.24	84.48	88.64
Comic (10%)	91.41	83.63	87.34
Cars (30%)	97.20	92.22	94.65
Cars (10%)	96.51	91.82	94.11

Table 5.1: Token level summary extraction results trained with 10% and 30% of the data

	Prec.	Recall	F-measure
Hotel (30%)	87.44	85.59	86.51
Hotel (10%)	86.52	84.54	85.52
Comic (30%)	81.73	80.84	81.28
Comic (10%)	79.94	76.71	78.29
Cars (30%)	92.67	90.63	91.64
Cars (10%)	92.08	90.31	91.19

Table 5.2: Field level summary extraction results trained with 10% and 30% of the data

Since the record linkage approach returns the reference set attributes as annotation, this annotation requires validation. To do this, there needs to be a way to link the results from performing record linkage to extraction results. Note one can consider the fact that a correct match during record linkage is the same as correctly identifying those attributes from the reference set in the post, at the field level. In the ongoing example from Figure 1.2, when the record linkage step matches the post to the reference member with a hotel name of "Holiday Inn Select", it is like extracting this hotel name from the post. Thus, one can consider the record linkage results to be field level extraction results for the attributes in the reference set.

Using the reference set attributes as annotation has some interesting implications. For instance, the attributes from the reference set provide a standardized set of values for querying the data. Also, returning reference set attributes for types not found in the post provides information that might have been missing previously. As an example, a post might leave out the star rating. Yet, now there exists one from the reference set record upon which to query. Another interesting implication arises as a solution for the cases where extraction is hard. None of the systems extracted the comic description well. However, by including the reference set description, the record linkage results reflect

how effectively the approach extracted (and thus labeled), a description. This yields an improvement of over 20% for precision, and almost 10% for recall.

It may seem that using the reference set attributes for annotation is enough and that extraction is unnecessary, but this is not the case. For one thing, one may want to see the actual values entered for different attributes. For instance, a user might have interest in discovering the most common spelling mistake or abbreviation for a attribute. Also, there are cases when the extraction results outperform the record linkage results as seen with the hotel name and star rating. This happens because even if a post is matched to an incorrect member of the reference set, that incorrect member is most likely very close to the correct match, and so it can be used to correctly extract much of the information. As an example, there might be hotels with the same star rating and hotel name, but with a different area. If this area is not included in the post or included in a convoluted way the record linkage step might not get a correct match. However, the reference set member could still be used to extract the hotel name and star rating.

Lastly, the more discriminative information passed to the SVM, the better it will perform. Assume the only required extracted attributes are the price and date from hotel posts, and the rest of the annotation came from the reference set. The SVM still benefits in training it to extract the attributes of the reference set, because the SVM would then know that by classifying a certain piece of information as a certain attribute, it is not another piece of information. It is just as important to recognize what a token is not. For example, consider a reference set hotel named "The $41 Inn." Then when post matches this reference set member, and the SVM sees the token "$41," it knows that it is most

likely a hotel name. If the SVM were not trained to extract all of the attributes, this would be classified as a price, since the SVM has no notion of a hotel name.

Extraction on all of the attributes also helps the system to classify (and ignore) tokens that are "junk". Labeling something as junk is much more descriptive if it is labeled junk out of many possible class labels that could share lexical characteristics. This helps to improve the extraction results on items that are not in the reference set, as evidenced in the results.

On the topic of reference sets, it is important to note that the algorithm is not tied to a single reference set. The algorithm extends to include multiple reference sets by iterating the process for each reference set used.

Consider the following two cases. First, suppose a user wants to extract conference names and cities and she has individual lists of each. If the approach were tied to using one reference set that would require constructing a reference set that contains the power set of cities crossed with conference names. This approach would not scale for many attributes from distinct sources. However, if these lists are used as two reference sets, one for each attribute, the algorithm can run once with the conference name data, and once with a reference set of cities. This iterative exploitation of the reference sets allows for n reference set attributes to be added without a combinatorial explosion.

The next interesting case happens when a post contains more than one of the same attribute. For example, a user needs to extract two cities from some post. If one reference set is used, then it would include the cross product of all cities. However, using a single reference set of city names can be done by slightly modifying the algorithm. The new algorithm makes a first pass with the city reference set. During this pass, the record

linkage match will either be one of the cities that matches best, or a tie between them. In the case of a tie, just choose the first match. Using this reference city the system can then extract the city from the post, and remove it from the post. Then the system simply runs the process again, which will catch the second city, using the same, single reference set. This could be repeated as many times as needed.

One issue that arises with reference sets is the discrepancy between user's knowledge and the domain experts who generally create the reference sets. In the *cars* domain, for instance, users will interchangeably use the attribute values "hatchback," "liftback," and "wagon." The reference set never includes the term "liftback" which suggests it is a synonym for hatchback used in common speech, but not in Edmund's automobile jargon. The term "wagon" is used by Edmunds, but it is not used for some of the cars that users describe as "hatchbacks." This implies there is a slight difference in meaning between the two, according to the reference set authors.

Two issues arise from these discrepancies. The first is the users interchanging the words can cause some problems for the extraction and for the record linkage, but this could be overcome by incorporating some sort of thesaurus into the algorithm. During record linkage, a thesaurus could expand certain attribute values used for matching, for example including "hatchback" and "liftback" when the reference set attribute includes the term "wagon." However, there are some more subtle issues here. It is mostly not the case that a "hatchback" is called a "wagon" but it does happen that a "wagon" is called a "hatchback." The frequency of replacement needs to taken into consideration so that errant matches are not created. How to automate this would be a line of future research. The other issue arises from the assumption that Edmunds is correct to define one car as

a "wagon" which has a different meaning from classifying it as a "hatchback." In fact, Edmunds classifies the Mazda Protege5 as a "wagon," while Kelly Blue Book[1] classifies it as a "hatchback." This seems to invalidate the idea that "wagon" is different in meaning from "hatchback." They appear to be simple synonyms, but this would remain unknown without the outside knowledge of Kelly Blue Book. More generally, one assumes that the reference set is a correct set of standardized values, but this is not an absolute truth. That is why the most meaningful reference sets are those that can be constructed from agreed-upon ontologies from the Semantic Web. For instance, a reference set derived from an ontology for cars created by all of the biggest automotive businesses should alleviate many of the issues in meaning, and a thesaurus scheme could work out the discrepancies introduced by the users, rather than the reference sets.

[1] www.kbb.com

Chapter 6

Related Work

This research is motivated by the goal that the cost of annotating documents for the Semantic Web should be free, that is, automatic and invisible to users [16]. Many researchers have followed this path, attempting to automatically mark up documents for the Semantic Web, as proposed here [7, 13, 15, 31]. 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 [8]). This is not an option when the data is ungrammatical, like the post data. In a similar vein, there are systems such as ADEL [21] which rely on the structure to identify and annotate records in web pages. Again, the failure of the posts to exhibit structure makes this approach inappropriate. So, while there is a fair amount of work in automatic labeling, there is not much emphasis on techniques that could do this on text that is unstructured and ungrammatical.

While the idea of record linkage is not new [14] and is well studied even now [3], most of the focus for this work matches 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 [10] allows for record linkage without decomposed attributes, but as shown in Section 4.1 Phoebus outperforms WHIRL, since WHIRL relies solely on the vector-based cosine similarity between the attributes, while Phoebus exploits a larger set of features to represent both field and record level similarity.

Using the reference set's attributes as normalized values is similar to the idea of data cleaning. However, most of the data cleaning algorithms assume that there are tuple-to-tuple transformations [20, 6]. That is, there is some function that maps the attributes of one tuple to the attributes of another. This approach would not work on ungrammatical and unstructured data, where all of the attributes are embedded within the post, which maps to a set of attributes from the reference set.

Information extraction can semantically annotate data, which is why the experiments compare the technique of this thesis to other IE approaches, such as the Simple Tagger Conditional Random Field [22]. Other IE approaches, such as Datamold [4] and CRAM [1], segment whole records (like bibliographies) into attributes. However, both of these systems require that every token of a record receive a label, which is not possible with posts that are filled with irrelevant tokens. CRAM is also similar in its use of reference sets for extraction. However, they assume that the reference set members already match the data for extraction, while Phoebus does this record linkage automatically. Another IE approach similar to ours performs named entity recognition using a dictionary component [9]. However, this technique requires that entire segments have the same class label, while the extraction approach of this thesis can handle the case where an attribute is broken up in the middle by another attribute, say a hotel name interrupted by a hotel area.

Chapter 7

Conclusion

This thesis presented an algorithm for semantically annotating text that is ungrammatical and unstructured. This technique provides much more utility to data sources that are full of information, but cannot support structured queries. Using this approach, Ebay agents could monitor the auctions looking for the best deals, or a user could find the average price of a four star hotel in San Diego. This approach to semantic annotation is necessary as society transitions into the Semantic Web, where information needs annotation for software systems to use it, but users are unwilling to do the extra work to provide the required annotation.

In the future, this technique could link with a mediator [29] framework for automatically acquiring reference sets. This is similar to automatically incorporating secondary sources for record linkage [23]. How to automatically formulate a query to retrieve the correct domain reference set is a direction of future research. With a mediator framework in place, Phoebus could incorporate as many reference sets as needed for full coverage of possible attribute values and attribute types.

Unsupervised approaches to record linkage and extraction are also topics of future research. By including unsupervised record linkage and extraction with a mediator component, the approach would be entirely self-contained, making semantic annotation of posts as simple as starting the process. Also, the current implementation only gives one class label per token. Ideally Phoebus would give a token all possible labels, and then remove the extraneous tokens when the systems cleans up the attributes, as described in Section 3. This disambiguation should lead to much higher accuracy during extraction.

As other future lines of work, it seems appropriate to investigate the inclusion of thesauri for terms in the attributes, with the frequency of replacement of the terms taken into consideration. Also technologies that automatically construct the reference sets (and eventually thesauri) from the numerous ontologies that exist on the Semantic Web is a desired research path.

The long term goal for this work involves automating the entire process. If the record linkage and extraction methods could become unsupervised, a line of future work as well, then the system could automatically generate and incorporate the reference sets, and then apply them to automatically annotate the data source. This would be an ideal approach for making the Semantic Web more useful with no user involvement.

Reference List

[1] Eugene Agichtein and Venkatesh Ganti. Mining reference tables for automatic text segmentation. In *the Proceedings of the 10th ACM Int'l Conf. on Knowledge Discovery and Data Mining*, Seattle, Washington, August 2004. ACM Press.

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

[3] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM Int'l Conf. on Knowledge Discovery and Data Mining*, pages 39–48, Washington, DC, August 2003. ACM Press.

[4] Vinayak Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. Automatic segmentation of text into structured records. In *Proceedings of ACM SIGMOD*, 2001.

[5] 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 and 11th Conference on Innovative Applications of Artificial Intelligence*, pages 328–334, Orlando, Florida, August 1999.

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

[7] 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.

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

[9] 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 Int'l Conf. Knowledge Discovery and Data Mining*, Seattle, Washington, August 2004. ACM Press.

[10] 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.

[11] 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*, 2003.

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

[13] 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.

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

[15] 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*. Springer Verlag, 2002.

[16] James Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.

[17] 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.

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

[19] 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, San Francisco, CA, 2001.

[20] 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.

[21] 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.

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

[23] Martin Michalowski, Snehal Thakkar, and Craig A. Knoblock. Automatically utilizing secondary sources to align information across sources. In *AI Magazine, Special Issue on Semantic Integration*, volume 26, pages 33–45. 2005.

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

[25] 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.

[26] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3), 1980.

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

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

[29] Snehal Thakkar, Jose Luis Ambite, and Craig A. Knoblock. A data integration approach to automatically composing and optimizing web services. In *Proceedings of the ICAPS Workshop on Planning and Scheduling for Web and Grid Services*, Whistler, BC, Canada, 2004.

[30] 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*, Banff, Canada, August 2004.

[31] 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*, 2002.

[32] Ben Wellner, Andrew McCallum, Fuchun Peng, and Michael Hay. An integrated, conditional model of information extraction and coreference with application to citation matching. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 2004.

[33] 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.

[34] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342, New York, NY, USA, 2001. ACM Press.