Journal of Artificial Intelligence Research 31 (2008) 543-590 Submitted 08/07; published 03/08

Creating Relational Data from Unstructured and Ungrammatical Data Sources

Matthew Michelson MICHELSO@ISI.EDU
Craig A. Knoblock KNOBLOCK@ISI.EDU
University of Southern California
Information Sciences Instistute
4676 Admiralty Way
Marina del Rey, CA 90292 USA

Abstract

In order for agents to act on behalf of users, they will have to retrieve and integrate vast amounts of textual data on the World Wide Web. However, much of the useful data on the Web is neither grammatical nor formally structured, making querying difficult. Examples of these types of data sources are online classifieds like Craigslist[1] and auction item listings like eBay.[2] We call this unstructured, ungrammatical data "posts." The unstructured nature of posts makes query and integration difficult because the attributes are embedded within the text. Also, these attributes do not conform to standardized values, which prevents queries based on a common attribute value. The schema is unknown and the values may vary dramatically making accurate search difficult. Creating relational data for easy querying requires that we define a schema for the embedded attributes and extract values from the posts while standardizing these values. Traditional information extraction (IE) is inadequate to perform this task because it relies on clues from the data, such as structure or natural language, neither of which are found in posts. Furthermore, traditional information extraction does not incorporate data cleaning, which is necessary to accurately query and integrate the source. The two-step approach described in this paper creates relational data sets from unstructured and ungrammatical text by addressing both issues. To do this, we require a set of known entities called a "reference set." The first step aligns each post to each member of each reference set. This allows our algorithm to define a schema over the post and include standard values for the attributes defined by this schema. The second step performs information extraction for the attributes, including attributes not easily represented by reference sets, such as a price. In this manner we create a relational structure over previously unstructured data, supporting deep and accurate queries over the data as well as standard values for integration. Our experimental results show that our technique matches the posts to the reference set accurately and efficiently and outperforms state-of-the-art extraction systems on the extraction task from posts.

1. Introduction

The future vision of the Web includes computer agents searching for information, making decisions and taking actions on behalf of human users. For instance, an agent could query a number of data sources to find the lowest price for a given car and then email the user the car listing, along with directions to the seller and available appointments to see the car.

1. www.craigslist.org
2. www.ebay.com

This requires the agent to contain two data gathering mechanisms: the ability to query sources and the ability to integrate relevant sources of information.

However, these data gathering mechanisms assume that the sources themselves are designed to support relational queries, such as having well defined schema and standard values for the attributes. Yet this is not always the case. There are many data sources on the World Wide Web that would be useful to query, but the textual data within them is unstructured and is not designed to support querying. We call the text of such data sources "posts." Examples of "posts" include the text of eBay auction listings, Internet classifieds like Craigslist, bulletin boards such as Bidding For Travel[3], or even the summary text below the hyperlinks returned after querying Google. As a running example, consider the three posts for used car classifieds shown in Table 1.

Table 1: Three posts for Honda Civics from Craigslist

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

The current method to query posts, whether by an agent or a person, is keyword search. However, keyword search is inaccurate and cannot support relational queries. For example, a difference in spelling between the keyword and that same attribute within a post would limit that post from being returned in the search. This would be the case if a user searched the example listings for "Civic" since the second post would not be returned. Another factor which limits keyword accuracy is the exclusion of redundant attributes. For example, some classified posts about cars only include the car model, and not the make, since the make is implied by the model. This is shown in the first and third post of Table 1. In these cases, if a user does a keyword search using the make "Honda," these posts will not be returned.

Moreover, keyword search is not a rich query framework. For instance, consider the query, *What is the average price for all Hondas from 1999 or later?* To do this with keyword search requires a user to search on "Honda" and retrieve all that are from 1999 or later. Then the user must traverse the returned set, keeping track of the prices and removing incorrectly returned posts.

However, if a schema with standardized attribute values is defined over the entities in the posts, then a user could run the example query using a simple SQL statement and do so accurately, addressing both problems created by keyword search. The standardized attribute values ensure invariance to issues such as spelling differences. Also, each post is associated with a full schema with values, so even though a post might not contain a car make, for instance, its schema does and has the correct value for it, so it will be returned in a query on car makes. Furthermore, these standardized values allow for integration of the source with outside sources. Integrating sources usually entails joining the two sources directly on attributes or translations of the attributes. Without standardized values and

3. www.biddingfortravel.com

a schema, it would not be possible to link these ungrammatical and unstructured data sources with outside sources. This paper addresses the problem of adding a schema with standardized attributes over the set of posts, creating a relational data set that can support deep and accurate queries.

One way to create a relational data set from the posts is to define a schema and then fill in values for the schema elements using techniques such as information extraction. This is sometimes called semantic annotation. For example, taking the second post of Table 1 and semantically annotating it might yield "93- 4dr Honda Civc LX Stick Shift $1800 <make>Honda< \make> <model>Civc< \model> <trim>4dr LX< \trim> <year>1993< \year> <price>1800< \price>." However, traditional information extraction, relies on grammatical and structural characteristics of the text to identify the attributes to extract. Yet posts by definition are not structured or grammatical. Therefore, wrapper extraction technologies such as Stalker (Muslea, Minton, & Knoblock, 2001) or RoadRunner (Crescenzi, Mecca, & Merialdo, 2001) cannot exploit the structure of the posts. Nor are posts grammatical enough to exploit Natural Language Processing (NLP) based extraction techniques such as those used in Whisk (Soderland, 1999) or Rapier (Califf & Mooney, 1999).

Beyond the difficulties in extracting the attributes within a post using traditional extraction methods, we also require that the values for the attributes are standardized, which is a process known as data cleaning. Otherwise, querying our newly relational data would be inaccurate and boil down to keyword search. For instance, using the annotation above, we would still need to query where the model is "Civc" to return this record. Traditional extraction does not address this.

However, most data cleaning algorithms assume that there are tuple-to-tuple transformations (Lee, Ling, Lu, & Ko, 1999; Chaudhuri, Ganjam, Ganti, & Motwani, 2003). 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 the attributes are embedded within the post, which maps to a set of attributes from the reference set. Therefore we need to take a different approach to the problems of figuring out the attributes within a post and cleaning them.

Our approach to creating relational data sets from unstructured and ungrammatical posts exploits "reference sets." A reference set consists 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.[4] It can also be an online (or offline) database, such as the Comics Price Guide.[5] 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. Using our car example, a reference set might be the Edmunds car buying guide[6], which defines a schema for cars as well as standard values for attributes such as the model and the trim. In order to construct reference sets from Web sources, such as the

4. http://www.cia.gov/cia/publications/factbook/
5. www.comicspriceguide.com
6. www.edmunds.com

Edmunds car buying guide, we use wrapper technologies (Agent Builder[7] in this case) to scrape data from the Web source, using the schema that the source defines for the car.

To use a reference set to build a relational data set we exploit the attributes in the reference set to determine the attributes from the post that can be extracted. The first step of our algorithm finds the best matching member of the reference set for the post. This is called the "record linkage" step. By matching a post to a member of the reference set we can define schema elements for the post using the schema of the reference set, and we can provide standard attributes for these attributes by using the attributes from the reference set when a user queries the posts.

Next, we perform information extraction to extract the actual values in the post that match the schema elements defined by the reference set. This step 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 record linkage step. In this step we also extract attributes that are not easily represented by reference sets, such as prices or dates. Although we already have the schema and standardized attributes required to create a relational data set over the posts, we still extract the actual attributes embedded within the post so that we can more accurately learn to extract the attributes not represented by a reference set, such as prices and dates. While these attributes can be extracted using regular expressions, if we extract the actual attributes within the post we might be able to do so more accurately. For example, consider the "Ford 500" car. Without actually extracting the attributes within a post, we might extract "500" as a price, when it is actually a car name. Our overall approach is outlined in Figure 1.

Although we previously describe a similar approach to semantically annotating posts (Michelson & Knoblock, 2005), this paper extends that research by combining the annotation with our work on more scalable record matching (Michelson & Knoblock, 2006). Not only does this make the matching step for our annotation more scalable, it also demonstrates that our work on efficient record matching extends to our unique problem of matching posts, with embedded attributes, to structured, relational data. This paper also presents a more detailed description than our past work, including a more thorough evaluation of the procedure than previously, using larger experimental data sets including a reference set that includes tens of thousands of records.

This article is organized as follows. We first describe our algorithm for aligning the posts to the best matching members of the reference set in Section 2. In particular, we show how this matching takes place, and how we efficiently generate candidate matches to make the matching procedure more scalable. In Section 3, we demonstrate how to exploit the matches to extract the attributes embedded within the post. We present some experiments in Section 4, validating our approaches to blocking, matching and information extraction for unstructured and ungrammatical text. We follow with a discussion of these results in Section 5 and then present related work in Section 6. We finish with some final thoughts and conclusions in Section 7.

7. A product of Fetch Technologies http://www.fetch.com/products.asp

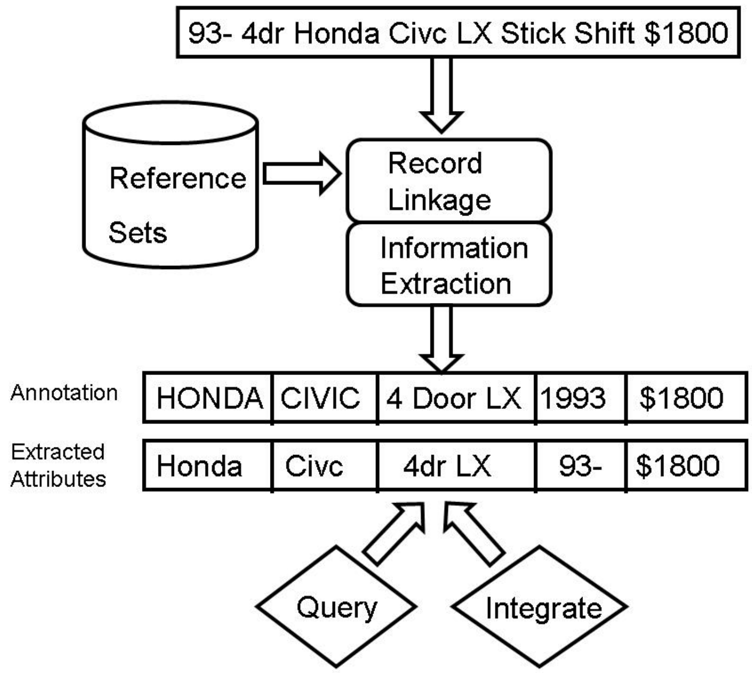

Figure 1: Creating relational data from unstructured sources

2. Aligning Posts to a Reference Set

To exploit the reference set attributes to create relational data from the posts, the algorithm needs to first decide which member of the reference set best matches the post. This matching, known as record linkage (Fellegi & Sunter, 1969), provides the schema and attribute values necessary to query and integrate the unstructured and ungrammatical data source. 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.

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. This step is detailed in Section 2.1 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 2.2.

2.1 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. 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 (Newcombe, 1967). That is, to block on an attribute, first we sort or cluster the data sets by the attribute. Then we apply the comparison method to only a single member of a block. After blocking, the candidate matches 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* (Hernandez & Stolfo, 1998), which combines the candidates generated during independent runs. For example, with our cars data, we 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. Using our example, our 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 we 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 our experiments. The key for methods is not necessarily choosing the fastest (though we 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, we can limit the number of candidates it generates. For example, if we change ({*token-match,* *make*}) to ({*token-match, make*} ∧ {*token-match, trim*}) we still cover new true matches, but we generate 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 (Elfeky, Verykios, & Elmagarmid, 2002).

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 we changed ({*token-match, zip*}) to ({*token-match, zip*} ∧ {*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 our PC. For example, we added the second conjunction to our 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. Previously we showed BSL aided the scalability of record linkage (Michelson & Knoblock, 2006), and this paper extends 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 (Mitchell, 1997). In our 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 2.

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 we maximize 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 we learn a new conjunction (in the *LEARN-ONE-RULE* step) and our current blocking scheme has a rule that already contains the newly learned rule, then we can remove the rule containing the newly learned rule. This is an optimization that allows us to check rule containment as we go, rather than at the end.

Table 2: Modified Sequential Covering Algorithm

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

The rule containment is possible because we can guarantee that we learn less restrictive rules as we go. We can prove this guarantee as follows. Our proof is done by contradiction. Assume we have two attributes A and B, and a method X. Also, assume that our previously learned rules contain the following conjunction, $(\{X,\ A\})$ and we currently learned the rule $(\{X,\ A\} \wedge \{X,\ B\})$. That is, we assume our 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 we remove all examples covered by $(\{X,\ A\})$ when we learn it. 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 we could not have learned the rule $(\{X,\ A\} \wedge \{X,\ B\})$. Thus, we have a contradiction.

As we 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. We have already mentioned how BSL maximizes the number of true positives in the candidate set and now we describe how BSL minimizes the overall size of the candidate set, which yields more scalable record linkage. To minimize the candidate set's size, we learn as restrictive a conjunction as we can during each call to *LEARN-ONE-RULE* during the SCA. We define restrictive as minimizing the number of candidates generated, as long as a certain number of true matches are still covered. (Without this restriction, we 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, we learn the conjunction that maximizes the Reduction Ratio, while at the same time covering a minimum value of Pairs Completeness. We use a beam search to allow for some backtracking, since the search is greedy. However, since the beam search goes from general-to-specific, we can ensure that the final rule is as restrictive as possible. The full *LEARN-ONE-RULE* is given in Table 3.

The constraint that a conjunction has a minimum PC ensures that the learned conjunction does not over-fit to the data. Without this restriction, it would be possible 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 is learning as restrictive a rule as 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 we set the threshold high, we will learn fewer, less restrictive conjunctions, possibly limiting our RR, although this may increase PC slightly. If we set it lower, we cover more examples, but we need 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. We set this parameter to 50% for our experiments[8].

Now we analyze the running time of BSL and we show how BSL can take into account the running time of different blocking methods, if need be. Assume that we have x ($method$, $attribute$) pairs such as ($token$, $first-name$). Now, assume that our beam size is b, since we use general-to-specific beam-search in our Learn-One-Rule procedure. Also, for the time being, assume each (method, attribute) pair can generate its blocking candidates in $O(1)$ time. (We relax this assumption later.) Each time we hit Learn-One-Rule within BSL, we 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, we try it against all other (method, attribute) pairs. Now, in the worst case, each learned disjunct would only cover 1 training example, so our rule is a disjunction of all pairs x. Therefore, we run the Learn-One-Rule x times, resulting in a learning time of $O(bx^2)$. If we have e training examples, the full training time is $O(ebx^2)$, for BSL to learn the blocking scheme.

Now, while we 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

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

Table 3: Learning a conjunction of {method, attribute} pairs

```
LEARN-ONE-RULE(attributes, examples, min_thresh, k)
Best-Conjunction ← {}
Candidate-conjunctions ← all {method, attribute} pairs
While Candidate-conjunctions not empty, do
  For each ch ∈ Candidate-conjunctions
    If not first iteration
      ch ← ch ∪ {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) ≥ min_thresh
      Best-Conjunction ← ch
  Candidate-conjunctions ← best k members of Candidate-conjunctions
return Best-conjunction
```

further the blocking times can vary significantly (Bilenko, Kamath, & Mooney, 2006). Let us 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 our training example. Using this notation, our Learn-One-Rule time becomes $O(b(xt_x(e)))$ (we run $t_x(e)$ time for each pair in x) and so our full training time becomes $O(eb(xt_x(e))^2)$. Clearly such a running time will be dominated by the most expensive blocking methodology. 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 we are classifying.

From a practical standpoint, we can easily modify BSL to account for the time it takes certain blocking methods to generate their candidates. In the Learn-One-Rule step, we 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, we modify Learn-One-Rule to maximize the performance of any disjunct based on this weighted average. Table 4 shows the modified version of Learn-One-Rule, and the changes are shown in **bold**.

Table 4: Learning a conjunction of {method, attribute} pairs using weights

```
LEARN-ONE-RULE(attributes, examples, min_thresh, k)
Best-Conj ← {}
Candidate-conjunctions ← all {method, attribute} pairs
While Candidate-conjunctions not empty, do
  For each ch ∈ Candidate-conjunctions
    If not first iteration
      ch ← ch ∪ {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) ≥ min_thresh
      Best-conj ← ch
  Candidate-conjunctions ← best k members of Candidate-conjunctions
return Best-conj
```

Note that when we set W_b to 0, we are using the same version of Learn-One-Rule as used throughout this paper, where we only consider the Reduction Ratio. Since our methods (token and n-gram match) are simple to compute, requiring more time to build the initial index than to do the candidate generation, we 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 we choose the blocking method, but more importantly that we choose the appropriate attributes to block on. In this sense, BSL is more like a feature selection algorithm than a blocking method. As we 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.

We can easily extend our 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 our LEARN-ONE-RULE, we instead compare 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 we learn a good blocking scheme, we 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.

2.2 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. That is, one data source's record (the post) must align to a record from the other data source (the reference set candidates). While the whole alignment procedure is referred to as record linkage (Fellegi & Sunter, 1969), we refer to finding the particular matches after blocking as the "matching step."

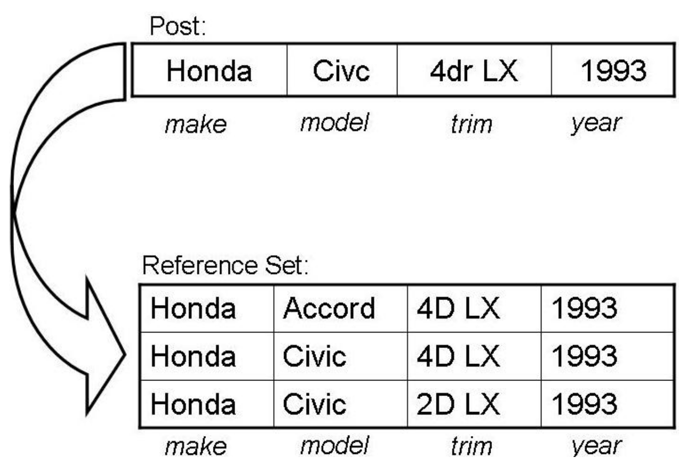

Figure 2: The traditional record linkage problem

However, the record linkage problem presented in this article 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, using the second post from Table 1, and assuming decomposed attributes, the *make* from the post is compared to the *make* of the reference

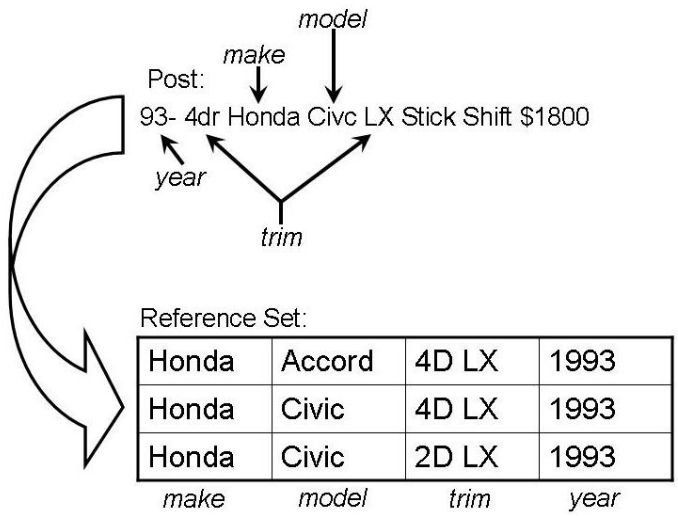

Figure 3: The problem of matching a post to the reference set

set. This is also done for the *models*, the *trims*, etc. 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. 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 3 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.

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 4. 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

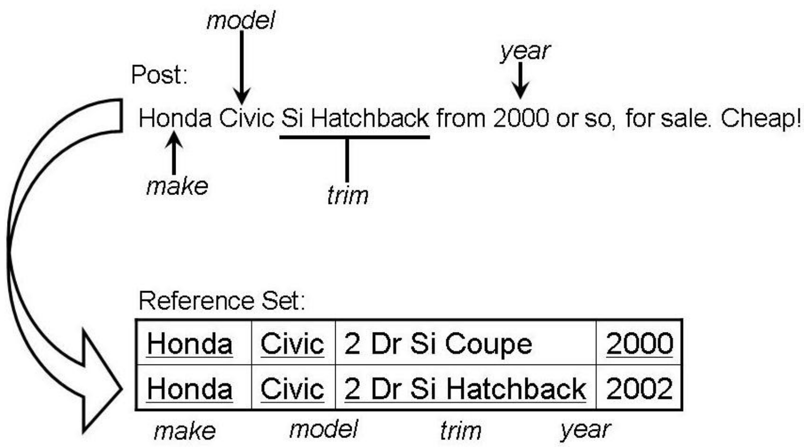

Figure 4: Two records with equal record level but different field level similarities

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 3, 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:

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

Or more generally:

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

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:

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

The vector *token_scores* comprises 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 (Cohen, Ravikumar, & Feinberg, 2003) and the other smooths its token probabilities using a Jelenik-Mercer mixture model (Zhai & Lafferty, 2001). 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:

$$
\begin{aligned}
\textit{token\_scores(post, attribute)} = <&\textit{Jensen-Shannon-Dirichlet(post, attribute)},\\
&\textit{Jensen-Shannon-JM-Mixture(post, attribute)},\\
&\textit{Jaccard(post, attribute)}>
\end{aligned}
$$

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 (Levenshtein, 1966), which returns the minimum number of operations to turn string S into string T, and the Smith-Waterman distance (Smith & Waterman, 1981) which is an extension to the Levenshtein distance. The last score in the vector *edit_scores* is the Jaro-Winkler similarity (Winkler & Thibaudeau, 1991), which is an extension of the Jaro metric (Jaro, 1989) 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 (Cohen et al., 2003) which was used for the experimental implementation as described in Section 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 (Porter, 1980), 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)>

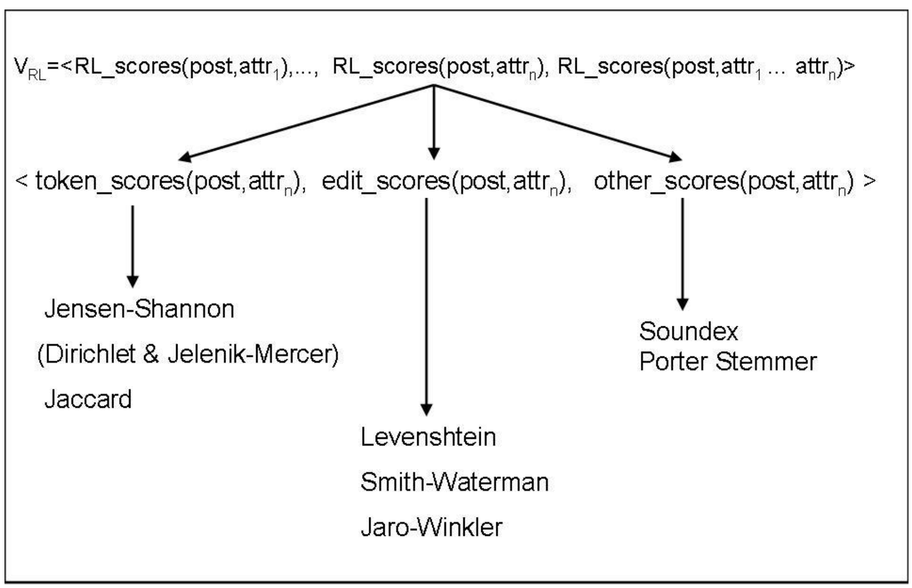

Figure 5: The full vector of similarity scores used for record linkage

Figure 5 shows the full composition of V_{RL}, with all the constituent similarity scores.

Once a V_{RL} is constructed for each of the candidates, 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:

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

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

$$
\begin{aligned}
V_{RL_1} &= <\{.999,...,1.2\},...,\{0.45,...,0.22\}> \\
V_{RL_2} &= <\{.888,...,0.0\},...,\{0.65,...,0.22\}>
\end{aligned}
$$

After rescoring they become:

$$
\begin{aligned}
V_{RL_1} &= <\{1,...,1\},...,\{0,...,1\}> \\
V_{RL_2} &= <\{0,...,0\},...,\{1,...,1\}>
\end{aligned}
$$

After rescoring, the matching step passes each V_{RL} to a Support Vector Machine (SVM) (Joachims, 1999) 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 we use SVMs in this paper to differentiate matches from non-matches, the algorithm is not strictly tied to this method. The main characteristics for our 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). We also tried to use a Naïve Bayes classifier for our 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 6.

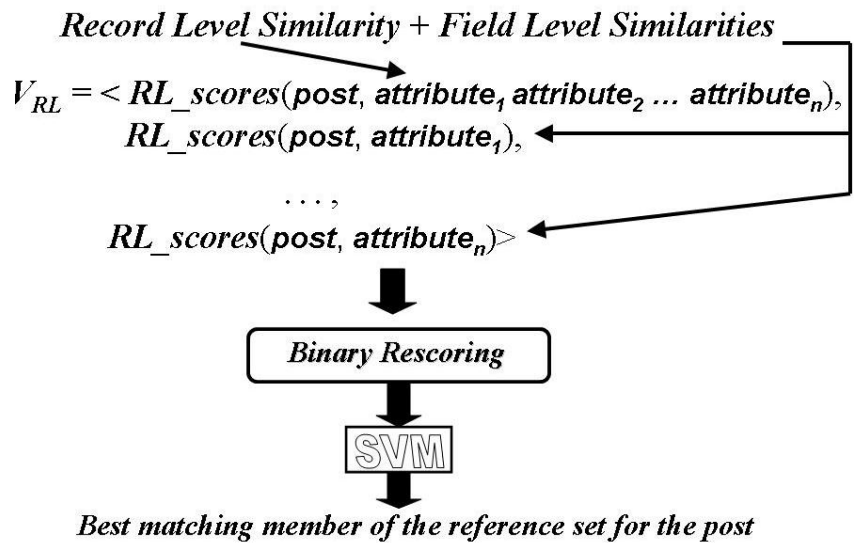

Figure 6: Our approach to matching posts to records from a reference set

In addition to providing a standardized set of values to query the posts, these standardized values allow for integration with outside sources because the values can be standardized to canonical values. For instance, if we want to integrate our car classifieds with a safety ratings website, we can now easily join the sources across the attribute values. In this manner, by approaching annotation as a record linkage problem, we can create relational data from unstructured and ungrammatical data sources. However, to aid in the extraction of attributes not easily represented in reference sets, we perform information extraction on the posts as well.

3. Extracting Data from Posts

Although the record linkage step creates most of the relational data from the posts, there are still attributes we would like to extract from the post which are not easily represented by reference sets, which means the record linkage step can not be used for these attributes. Examples of such attributes are dates and prices. Although many of these such attributes can be extracted using simple techniques, such as regular expressions, we can make their extraction and annotation ever more accurate by using sophisticated information extraction. To motivate this idea, consider the Ford car model called the "500." If we just used regular expressions, we might extract 500 as the price of the car, but this would not be the case. However, if we try to extract all of the attributes, including the model, then we would extract "500" as the model correctly. Furthermore, we might want to extract the actual attributes from a post, as they are, and our extraction algorithm allows this.

To perform extraction, 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.

In a broad sense, the algorithm has two parts. First we label each token with a possible attribute label or as "junk" to be ignored. After all the tokens in a post are labeled, we then clean each of the extracted labels. Figure 7 shows the whole procedure graphically, in detail, using the second post from Table 1. Each of the steps shown in this figure are described in detail below.

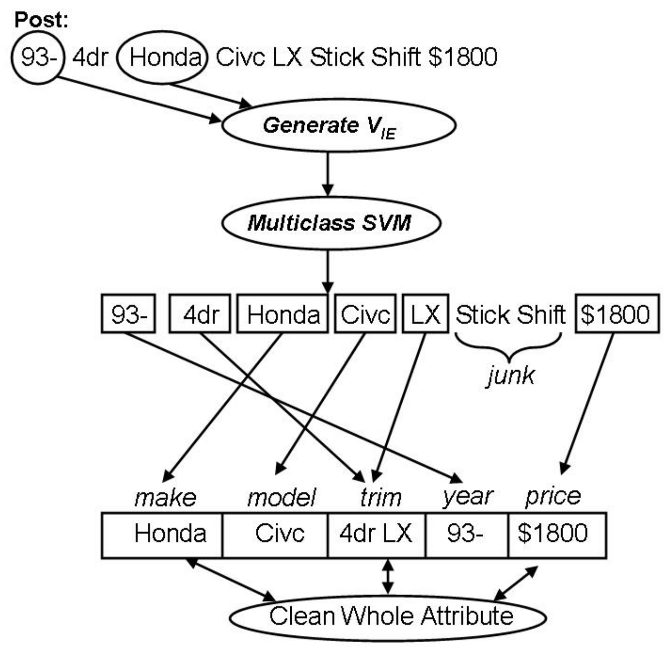

Figure 7: Extraction process for attributes

To begin the extraction process, the post is broken into tokens. Using the first post from Table 1 as an example, set of tokens becomes, {"93", "civic", "5speed",...}. 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 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:

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

becomes:

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

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 to be used for extraction.

Using the first post of Table 1, 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 "civic" of the post:

$$\begin{aligned} V_{IE} = <& \textit{common\_scores(``civic'')},\\ & \textit{IE\_scores(``civic'',``Honda'')},\\ & \textit{IE\_scores(``civic'',``Civic'')},\\ & \textit{IE\_scores(``civic'',``1993'')}> \end{aligned}$$

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 structured SVM (Tsochantaridis, Joachims, Hofmann, & Altun, 2005; Tsochantaridis, Hofmann, Joachims, & Altun, 2004) 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 our task. 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 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. 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.

Using the first post of Table 1 as our ongoing example, assume the phrase "civic (ri)" was extracted as the model. This might occur if there is a car with the model Civic Rx, for instance. In isolation, the token "(ri)" could be the "Rx" of the model. Comparing this extracted car model to the reference attribute "Civic" generates a Jaccard similarity of 0.5 and a Jaro-Winkler score of 0.83. This is shown at the top of Figure 8.

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. After all 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.

Shown as "Iteration 1" in Figure 8, the cleaning method finds that "(ri)" is a removal candidate since removing this token from the extracted car model yields a Jaccard score of 1.0 and a Jaro-Winkler score of 1.0, 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 (since there are none), the process terminates, yielding a more accurate attribute value. This is shown as "Iteration 2" in Figure 8. Note that this process would keep iterating, until no tokens can be removed that improve the scores over the baseline. The pseudocode for the algorithm is shown in Figure 9.

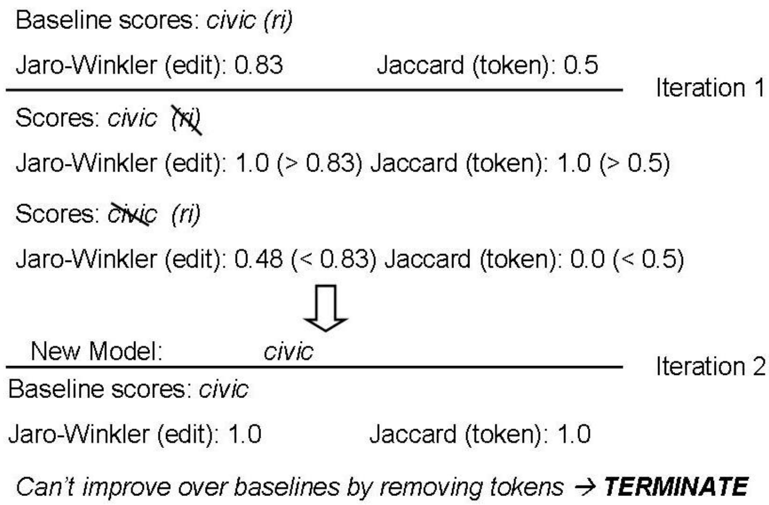

Figure 8: Improving extraction accuracy with reference set attributes

Note, however, that we do not limit the machine learning component of our extraction algorithm to SVMs. Instead, we claim that in some cases, reference sets can aid extraction in general, and to test this, in our architecture we can replace the SVM component with other methods. For example, in our extraction experiments we replace the SVM extractor with a Conditional Random Field (CRF) (Lafferty, McCallum, & Pereira, 2001) 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. Results

The Phoebus system was built to experimentally validate our approach to building relational data from unstructured and ungrammatical data sources. 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 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 to test the extraction algorithm. This data comes from the Bidding For Travel website[9] 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, with 1028 of these posts having a match in the reference set. The reference set comes from the Bidding For Travel hotel guides, which are special

9. www.biddingfortravel.com

Algorithm 3.1: CleanAttribute(E, R)

comment: Clean extracted attribute E using reference set attribute R

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

$$\textbf{do} \begin{cases} X_t \leftarrow \textsc{RemoveToken}(t, E) \\ JaroWinkler_{X_t} \leftarrow \textsc{JaroWinkler}(X_t, R) \\ Jaccard_{X_t} \leftarrow \textsc{Jaccard}(X_t, R) \\ \textbf{if} \begin{cases} JaroWinkler_{X_t} > JaroWinkler_{Baseline} \\ and \\ Jaccard_{X_t} > Jaccard_{Baseline} \end{cases} \\ \quad \textbf{then} \begin{cases} C \leftarrow C \cup t \end{cases} \end{cases}$$

$$\textbf{if} \begin{cases} C = null \\ \textbf{return}\ (E) \end{cases}$$
$$\textbf{else} \begin{cases} E \leftarrow \text{RemoveMaxCandidate(C,E)} \\ \textsc{CleanAttribute}(E, R) \end{cases}$$

Figure 9: Algorithm to clean an extracted attribute

posts listing all of the hotels ever posted about a given area. These special posts provide hotel names, hotel areas and star ratings, which are the reference set attributes. Therefore, these are the 3 attributes for which the standardized values are used, allowing us to treat these posts as a relational data set. This reference set contains 132 records.

The experimental data for the *comic* domain comes from posts for items for sale on eBay. To generate this data set, eBay was searched by the keywords “Incredible Hulk” and “Fantastic Four” in the comic books section of their website. (This returned some items that are not comics, such as t–shirts and some sets of comics not limited to those searched for, which makes the problem more difficult.) The returned records contain the attributes comic title, issue number, price, publisher, publication year and the description, which are extracted. (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, of which 697 have matches. The *comic* domain reference set uses data from the Comics Price Guide[10], which lists all the Incredible Hulk and Fantastic Four comics. This reference set has the attributes title, issue number, description, and publisher and contains 918 records.

The *cars* data consists of posts made to Craigslist regarding cars for sale. This dataset consists of classifieds for cars from Los Angeles, San Francisco, Boston, New York, New

10. http://www.comicspriceguide.com/

Jersey and Chicago. There are a total of 2,568 posts in this data set, and each post contains a make, model, year, trim and price. The reference set for the Cars domain comes from the Edmunds[11] car buying guide. From this data set we extracted the make, model, year and trim for all cars from 1990 to 2005, resulting in 20,076 records. There are 15,338 matches between the posts to Craigslist and the cars from Edmunds.

Unlike the hotels and comics domains, a strict 1-1 relationship between the post and reference set was not enforced in the 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, we can still use this make, model and year accurately for extraction. So, in this case, as mentioned previously, we pick one of the matches. This way, we exploit the attributes that we can from the reference set, since we have confidence in those.

For the experiments, 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. We believe that this is too much data to have to label, especially as data sets become large, so our experiments instead focus on using less 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%. We believe 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 us 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 we only run experiments training on 10% of the data. All experiments are performed 10 times, and the average results for these 10 trials are reported.

4.1 Record Linkage Results

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

4.1.1 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 we use two common methods. The first, which we 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 (McCallum, Nigam, & Ungar, 2000) and Bigram indexing (Baxter, Christen, & Churches, 2003), is slightly misaligned because the algorithms solve different

11. www.edmunds.com

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, we 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, we 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. We 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 (Baxter et al., 2003), we use a threshold of 0.3 for Bigram indexing, since that works the best. We also compare BSL to running a disjunction over all attributes using the simple token method only. In our results, we call this blocking rule "Disjunction." This disjunction mirrors the idea of picking the simplest possible blocking method: namely using all attributes with a very simple method.

As stated previously, the two goals of blocking can be quantified by the Reduction Ratio (RR) and the Pairs Completeness (PC). Table 5 shows not only these values but also how many candidates were generated on average over the entire test set, comparing the three different approaches. Table 5 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 6 shows a few example blocking schemes that the 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 our previous work on the topic (Michelson & Knoblock, 2006).

The results of Table 5 validate our 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 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. We 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. We 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 we can learn from the data, across the posts than we initially surmised.

Another interesting result is the poor reduction ratio of the Comic domain. This happens because most of the rules contain the disjunct that finds a common token within the comic

	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
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
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
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

Table 5: Blocking results using the BSL algorithm (amount of data used for training shown in parentheses).

Hotels Domain (30%)
({hotel area,token} $\wedge$ {hotel name,token} $\wedge$ {star rating, token}) $\cup$ ({hotel name, ngram3})
Hotels Domain (10%)
({hotel area,token} $\wedge$ {hotel name,token}) $\cup$ ({hotel name,ngram3})
Comic Domain (30%)
({title, token})
Comic Domain (10%)
({title, token}) $\cup$ ({issue number,token} $\wedge$ {publisher,token} $\wedge$ {title,ngram3})
Cars Domain (10%)
({make,token}) $\cup$ ({model,ngram3}) $\cup$ ({year,token} $\wedge$ {make,ngram3})

Table 6: Some example blocking schemes learned for each of the domains.

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." 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 Hotels domain we get a lower RR but the same PC when we use less training data. This happens because our 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 we said, maximizing the PC is the most important thing, so we choose not to do this. We want BSL to cover as many true positives as it can, even if that means losing a bit in the reduction.

In fact, we next test this notion explicitly. We 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.

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

Table 7: A comparison of BSL covering all training examples, and covering 95% of the training examples

Table 7 shows that when we use a threshold in the Hotels and Cars domain we see a statistically significant drop in Pairs Completeness with a statistically significant increase in Reduction Ratio.[12] 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 BSL to retain a rule with high RR, but lower PC. However, when we look at the record linkage results, we see that this threshold does in fact have a large effect.[13] Although there is no statistically significant difference in the F-measure for record linkage in the Hotels domain, the difference in Cars domain is dramatic. *When we use 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.*[14] Therefore, since the F-measure results differ by so much, we 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 we see in Table 7, this is especially true in the Cars domain where matching is much more difficult than in the Hotels domain.

Interestingly, in the Comic domain we do not see a statistically significant difference in the RR and PC. This is because across trials we almost always learn the same rule whether we use 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.

Our results using BSL are encouraging because they show that the algorithm also works for blocking when matching unstructured and ungrammatical text to a relational data source. This means the algorithm works in this special case too, not just the case of traditional record linkage where we are matching one structured source to another. This means our overall algorithm for semantic annotation is much more scalable because we are using fewer candidate matches than in our previous work (Michelson & Knoblock, 2005).

4.1.2 Matching Results

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

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

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

12. Bold means statistically significant using a two-tailed t-test with α set to 0.05
13. Please see subsection 4.1.2 for a description of the record linkage experiments and results.
14. 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.

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

The record linkage approach in this article is compared to WHIRL (Cohen, 2000). 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 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 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 8. 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 Cars domain, and the precision between Phoebus trained on 10% and 30% of the training data in the Comic 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 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 our 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. We expect this behavior because Phoebus has a notion of both field and token level similarity, using many different similarity measures. This justifies our use of the many similarity types and field and record level information, since our goal is to find as many matches as we can.

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

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

Table 8: Record linkage results

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 we want this technique to be widely applicable. In fact, we faced this practical issue ourselves in the Cars domain where we were 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. So, the fact that we can report good results with just 10% training data allows us to extend this work to the much larger Cars domain.

While our 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, we ran several experiments isolating each of these ideas.

First, we ran Phoebus matching on only the concatenation of the attributes from the reference set, rather than the concatenation and all the attributes individually. Earlier, we stated that we use the concatenation to mirror the idea of record level similarity and we also use each attribute to mirror field level similarity. It is our 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, we can test how the concatenation influences the matching in isolation. Table 9 shows these results for the different domains.

For the Cars and Comic domains we see an improvement in F-measure, indicating that that using the attributes and the concatenation is much better for matching than using the concatenation alone. This supports our notion that we also need 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

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

Table 9: Matching using only the concatenation

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 Hotels domain, we do not find 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 we see a huge improvement, and we never lose 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, we conclude that part of the reason Phoebus can outperform WHIRL is the use of the individual attributes for matching.

Our next experiment tests how important it is to include all of the string metrics in our feature vector for matching. To test this idea, we compare using all the metrics to using just one, the Jensen-Shannon distance. We choose the Jensen-Shannon distance because it outperformed both TF/IDF and even a “soft” TF/IDF (one that accounts for fuzzy token matches) in the task of selecting the right reference sets for a given set of posts (Michelson & Knoblock, 2007). These results are shown in Table 10.

As Table 10 shows, using all the metrics yielded a statistically significant, large improvement in F-measure for the Comic and Cars domains. This means that some of the other string metrics, such as the edit distances, were capturing similarities that the Jensen-Shannon distance alone did not. Interestingly, in both domains, using Phoebus with only the Jensen-Shannon distance does not dominate WHIRL’s performance. Therefore, the results of Table 10 and Table 9 demonstrate that Phoebus benefits from the combination

	Precision	Recall	F-Measure
Hotels			
Phoebus (30%)	87.70	93.78	90.63
Jensen-Shannon Only	89.65	92.28	90.94
WHIRL	83.61	83.53	83.13
Comic			
Phoebus (30%)	87.49	95.46	**91.30**
Jensen-Shannon Only	65.36	69.96	67.58
WHIRL	73.89	81.63	77.57
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

Table 10: Using all string metrics versus using only the Jensen-Shannon distance

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 Hotels 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 we include all of the metrics. Hence, either these missed matches are very difficult to discover, or we do not have a string metric in our 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, we conclude that Phoebus outperforms WHIRL because it combines multiple string metrics, it uses both individual attributes and the concatenation, and, as stated in Section 2.2, the SVM classifier is well suited for our record linkage task. These results also justify our inclusion of many metrics and the individual attributes, along with our use of SVM as our classifier.

Our last matching experiment justifies our binary rescoring mechanism. Table 11 shows the results of performing the binary rescoring for record linkage versus not performing this binary recoring. We hypothesize earlier in this paper 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 11 shows this to be the case, as across all domains when we perform the binary rescoring we gain 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.

	Precision	Recall	F-Measure
Hotels			
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
Comic			
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
Cars			
Phoebus (10%)	69.98	85.68	**77.04**
No Binary Rescoring	39.78	48.77	43.82

Table 11: Record linkage results with and without binary rescoring

4.2 Extraction Results

This section presents results that experimentally validate our approach to extracting the actual attributes embedded within the post. We also compare our approach to two other information extraction methods that rely on the structure or grammar of the posts.

First, the experiments compare Phoebus against a baseline Conditional Random Field (CRF) (Lafferty et al., 2001) extractor. 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. Therefore, by representing the state-of-the-art probabilistic graphical model, they present a strong comparison to our approach to extraction. 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 (Wellner, McCallum, Peng, & Hay, 2004). These experiments use the Simple Tagger implementation of CRFs from the MALLET (McCallum, 2002) suite of text processing tools.

Further, as stated in Section 3 on Extraction, we also created a version of Phoebus that uses CRFs, which we 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. We include PhoebusCRF to show that extraction in general can benefit from our reference set matching.

Second, the experiments 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 system (Ciravegna, 2001), which uses shallow NLP for extraction, has been shown to outperform other symbolic systems in extraction tasks, and so we use Amilcare as the other system to compare against. Since Amilcare can exploit gazetteers for extra information, for our experiments Amilcare receives the reference data as a gazetteer to aid the extraction. Both Simple Tagger and Amilcare are used with default settings.

Lastly, we compare Phoebus trained using 30% of the data for training to Phoebus trained using 10% of the data. (We do this for PhoebusCRF as well.) In our experimental results, the amount of training data is put in parentheses.

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).

For the cars data set, not all of the posts were labeled for training and testing the extraction. For this domain, we only labeled 702 of the posts for extraction, and use these for training and testing the extraction algorithm. Note, however, that Phoebus does perform the extraction on all of the posts, it just is not able to report results for those. In fact, a running demo of Phoebus, in the Cars domain is live.[15]

The extraction results are presented using Precision, Recall and F-Measure. Note that these extraction results are "field level" results. This means that an extraction is 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, it more accurately models how useful an extraction system would be. Tables 12, 13 and 14 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 fields in the test set that have the associated label. Also, observe that a * means that results between the highest Phoebus score (Phoebus or PhoebusCRF) and the highest baseline (Amilcare or Simple Tagger CRF) F-Measure are *not statistically significant* using a two-tailed paired t-test with α=0.05.

Phoebus and PhoebusCRF outperform the other systems on almost all attributes (13 of 16), as shown in Table 15. In fact, there was only one attribute where the baseline system was the best: using Amilcare to extract the Date attribute in the Hotels domain. For this attribute, Phoebus and PhoebusCRF both use the common-score regular-expression as the main identifying feature. Since this regular expression is user supplied, we propose that a better regular expression could make Phoebus/PhoebusCRF extract these dates even more accurately, overcoming this baseline. Since both systems perform well using the reference set data to aid the extraction, these results show that using reference sets can greatly aid extraction. This is especially evident when we compare PhoebusCRF to the Simple Tagger CRF, since the difference between these two extraction methods is the reference set attribute similarity scores and the common scores.

15. http://www.isi.edu/integration/Phoebus/demos.html This demo uses an extraction model trained on the 702 labeled extraction examples, and has been running live for months as of the writing of this article.

Hotel					
		Recall	Precision	F-Measure	Frequency
Area	Phoebus (30%)	83.73	84.76	84.23	~580
	Phoebus (10%)	77.80	83.58	80.52	
	PhoebusCRF (30%)	85.13	86.93	**86.02**	
	PhoebusCRF (10%)	80.71	83.38	82.01	
	Simple Tagger CRF (30%)	78.62	79.38	79.00	
	Amilcare (30%)	64.78	71.59	68.01	
Date	Phoebus (30%)	85.41	87.02	86.21	~700
	Phoebus (10%)	82.13	83.06	82.59	
	PhoebusCRF (30%)	87.20	87.11	87.15	
	PhoebusCRF (10%)	84.39	84.48	84.43	
	Simple Tagger CRF (30%)	63.60	63.25	63.42	
	Amilcare (30%)	86.18	94.10	**89.97**	
Name	Phoebus (30%)	77.27	75.18	76.21	~750
	Phoebus (10%)	75.59	74.25	74.92	
	PhoebusCRF (30%)	85.70	85.07	**85.38**	
	PhoebusCRF (10%)	81.46	81.69	81.57	
	Simple Tagger CRF (30%)	74.43	84.86	79.29	
	Amilcare (30%)	58.96	67.44	62.91	
Price	Phoebus (30%)	93.06	98.38	95.65	~720
	Phoebus (10%)	93.12	98.46	**95.72**	
	PhoebusCRF(30%)	92.56	94.90	93.71	
	PhoebusCRF (10%)	90.34	92.60	91.46	
	Simple Tagger CRF (30%)	71.68	73.45	72.55	
	Amilcare (30%)	88.04	91.10	89.54	
Star	Phoebus (30%)	97.39	97.01	97.20	~730
	Phoebus (10%)	96.94	96.90	96.92	
	PhoebusCRF (30%)	96.83	98.06	**97.44**	
	PhoebusCRF (10%)	96.17	96.74	96.45	
	Simple Tagger CRF (30%)	97.16	96.55	96.85	
	Amilcare (30%)	95.58	97.35	96.46	

Table 12: Field level extraction results: Hotels domain

Comic					
		Recall	Precision	F-Measure	Frequency
Descript.	Phoebus (30%)	32.43	30.71	31.51	~90
	Phoebus (10%)	30.16	27.15	28.52	
	PhoebusCRF (30%)	26.02	33.03	28.95	
	PhoebusCRF (10%)	15.45	26.83	18.54	
	Simple Tagger CRF (30%)	32.30	34.75	**33.43***	
	Amilcare (30%)	8.00	52.55	13.78	
Issue	Phoebus (30%)	83.39	83.65	83.52	~510
	Phoebus (10%)	80.90	82.17	81.52	
	PhoebusCRF (30%)	87.77	88.70	**88.23**	
	PhoebusCRF (10%)	83.01	84.68	83.84	
	Simple Tagger CRF (30%)	78.31	77.81	78.05	
	Amilcare (30%)	77.66	89.11	82.98	
Price	Phoebus (30%)	68.09	90.00	**77.39***	~15
	Phoebus (10%)	39.84	60.00	46.91	
	PhoebusCRF (30%)	51.06	85.34	61.16	
	PhoebusCRF (10%)	29.09	55.40	35.71	
	Simple Tagger CRF (30%)	44.24	84.44	55.77	
	Amilcare (30%)	41.21	66.67	50.93	
Publisher	Phoebus (30%)	100.00	85.38	**92.09**	~60
	Phoebus (10%)	99.85	83.89	91.18	
	PhoebusCRF (30%)	77.91	88.30	82.50	
	PhoebusCRF (10%)	53.22	87.29	64.26	
	Simple Tagger CRF (30%)	78.13	88.52	82.72	
	Amilcare (30%)	63.75	90.48	74.75	
Title	Phoebus (30%)	89.34	89.34	89.34	~540
	Phoebus (10%)	89.37	89.37	89.37	
	PhoebusCRF (30%)	92.93	93.70	**93.31***	
	PhoebusCRF (10%)	90.64	92.13	91.37	
	Simple Tagger CRF (30%)	93.57	92.79	93.18	
	Amilcare (30%)	89.88	95.65	92.65	
Year	Phoebus (30%)	78.44	97.69	**86.99**	~100
	Phoebus (10%)	77.50	97.35	86.28	
	PhoebusCRF (30%)	76.24	93.46	83.80	
	PhoebusCRF (10%)	54.63	85.07	66.14	
	Simple Tagger CRF (30%)	39.93	72.89	51.54	
	Amilcare (30%)	77.05	85.67	81.04	

Table 13: Field level extraction results: Comic domain.

Cars					
		Recall	Precision	F-Measure	Frequency
Make	Phoebus (10%)	98.21	99.93	**99.06**	~580
	PhoebusCRF (10%)	90.73	96.71	93.36	
	Simple Tagger CRF (10%)	85.68	95.69	90.39	
	Amilcare (10%)	97.58	91.76	94.57	
Model	Phoebus (10%)	92.61	96.67	**94.59**	~620
	PhoebusCRF (10%)	84.58	94.10	88.79	
	Simple Tagger CRF (10%)	78.76	91.21	84.52	
	Amilcare (10%)	78.44	84.31	81.24	
Price	Phoebus (10%)	97.17	95.91	**96.53**	~580
	PhoebusCRF (10%)	93.59	92.59	93.09	
	Simple Tagger CRF (10%)	83.66	98.16	90.33	
	Amilcare (10%)	90.06	91.27	90.28	
Trim	Phoebus (10%)	63.11	70.15	**66.43**	~375
	PhoebusCRF (10%)	55.61	64.95	59.28	
	Simple Tagger CRF (10%)	55.94	66.49	60.57	
	Amilcare (10%)	27.21	53.99	35.94	
Year	Phoebus (10%)	88.48	98.23	**93.08**	~600
	PhoebusCRF (10%)	85.54	96.44	90.59	
	Simple Tagger CRF (10%)	91.12	76.78	83.31	
	Amilcare (10%)	86.32	91.92	88.97	

Table 14: Field level extraction results: Cars domain.

Domain	Num. of Max. F-Measures				Total Attributes
	Phoebus	PhoebusCRF	Amilcare	Simple Tagger	
Hotel	1	**3**	1	0	5
Comic	**2**	1	0	0	6
Cars	**5**	0	0	0	5
All	**8**	**4**	1	0	16

Table 15: Summary results for extraction showing the number of times each system had statistically significant highest F-Measure for an attribute.

Phoebus performs especially well in the 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 the Cars domain, they are good enough to yield very high 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 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 be extracted correctly, but the rest of the attributes can be extracted using the reference set member.

A couple of other interesting notes come from these results. One of the most intriguing aspects of these results is that they allow us to estimate some level of structure for different attributes within a domain. Since CRFs rely more on the structure of the tokens within a post than the structured SVM method, we hypothesize that in the domains with more structure, PhoebusCRF should perform best and in the domains with the least structure, Phoebus should perform best. Table 15 shows this to be the case. PhoebusCRF dominates the Hotels domain, where, for example, many posts have a structure where the star rating comes before the hotel name. So using such structure should allow the extractor to get the hotel name more accurately than not using this information. Therefore we see that overall there is structure within the Hotels domain because PhoebusCRF is the method that performs best, not Phoebus. Contrast this with the Cars domain, which is highly unstructured, where Phoebus performs the best across all attributes. In this domain there are many missing tokens and the order of attributes is more varied. The Comic domain is varied with both some attributes that exhibit structure and some that do not, and as Table 15 shows, so are the cases where Phoebus or PhoebusCRF dominates. However, although the Hotels data exhibits some structure, the important aspect of this research is that using Phoebus allows one to perform extraction *without assuming* any structure in the data.

Also, a result worth noting is that the price attribute in the Comic domain is a bit misleading. In fact, none of the systems were statistically significant with respect to each other because there were so few prices to extract that the F-Measures were all over for all the systems.

Another aspect that came to light with statistical significance is the generalization of the algorithm. For the Hotels and Comic domains, where we were able to use both 30% and 10% of the data for training, there are not many cases with a statistically significant difference in the F-Measures for the extracted attributes using Phoebus. In the Hotels domain the name, the area and date had statistically significant F-Measures between training on 30% and 10% of the data, and in the Comic domain only the difference in F-Measure between the issue and description attributes were significant (though the description was borderline). This means of the 11 attributes in both domains, roughly half of them were insignificant. Therefore there is little difference in extraction whether we use 10% of the data for training or 30%, so the extraction algorithm generalizes very well. This is important since labeling data for extraction is very time consuming and expensive.

One interesting result to note is that except for the comic price (which was insignificant for all systems) and the hotel date (which was close), Phoebus, using either 10% or 30% training data, outperformed all of the other systems on the attributes that were not included in the reference set. This lends credibility to our claim earlier in the section that by training the system to extract all of the attributes, even those in the reference set, we can more accurately extract attributes not in the reference set because we are training the system to identify what something is not.

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 of posts: the auction sites, Internet classifieds and forum/bulletin boards.

5. Discussion

The goal of this research is to produce relational data from unstructured and ungrammatical data sources so that they can be accurately queried and integrated with other sources. By representing the attributes embedded within a post with the standardized values from the reference set, we can support structural queries and integration. For instance, we can perform aggregate queries because we can treat the data source as a relational database now. Furthermore, we have standardized values for performing joins across data sources, a key for integration of multiple sources. These standardized values also aid in the cases where the post actually does not contain the attribute. For instance, in Table 1, two of the listings do not include the make "Honda." However, once matched to the reference set, they contain a standardized value for this attribute which can then be used for querying and integrating these posts. This is especially powerful since the posts never explicitly stated these attribute values. The reference set attributes also provide a solution for the cases where the extraction is extremely difficult. For example, none of the systems extracted the description attribute of the Comic domain well. However, if one instead considers the description attribute from the reference set, which is quantified by the record linkage results for the Comic domain, this yields an improvement of over 50% in the F-Measure for identifying the description for a post.

It may seem that using the reference set attributes for annotation is enough since the values are already cleaned, and that extraction is unnecessary. However, 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 want to discover the most common spelling mistake or abbreviation for a attribute. Also, there are cases when the extraction results outperform the record linkage results. 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. For a strong example of this, consider the Cars domain. The F-measure for the record linkage results are not as good as those for the extraction results in this domain. This means most matches that were chosen where probably incorrect because they differ from the correct match by something small. For example, a true match could have the trim as "2 Door" while the incorrectly chosen match might have the trim "4 Door," but there would still be enough information, such as the rest of the trim tokens, the year, the make and the model to correctly extract those different attributes from the post itself. By performing the extraction for the values from the post itself, we can overcome the mistakes of the record linkage step because we can still exploit most of the information in the incorrectly chosen reference set member.

Extraction on all of the attributes also helps our system classify (and ignore) "junk" tokens. 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 improve the extraction results on items that are not in the reference set, such as prices and dates.

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 is confined 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 is 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 includes 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, choose the first match. Using this reference city, our system can then extract the city from the post, and remove it from the post. Then our 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 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 can 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 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 must be taken into consideration so that errant matches are not created. How to automate this is a line of future research. The other issue arises from trusting the correctness of the Edmunds source. We assume Edmunds is right 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[16] 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.

6. Related Work

Our research is driven by the principal that the cost of annotating documents for the Semantic Web should be free, that is, automatic and invisible to users (Hendler, 2001). Many researchers have followed this path, attempting to automatically mark up documents for the Semantic Web, as proposed here (Cimiano, Handschuh, & Staab, 2004; Dingli, Ciravegna, & Wilks, 2003; Handschuh, Staab, & Ciravegna, 2002; Vargas-Vera, Motta, Domingue, Lanzoni, Stutt, & Ciravegna, 2002). 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). This is not an option when the data is ungrammatical, like the post data. In a similar vein, there are systems such as ADEL (Lerman, Gazen, Minton, & Knoblock, 2004) 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 little emphasis on techniques that could label text that is both unstructured and ungrammatical.

Although the idea of record linkage is not new (Fellegi & Sunter, 1969) and is well studied even now (Bilenko & Mooney, 2003) 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 (Cohen, 2000) 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. We note with interest the EROCS system (Chakaravarthy, Gupta, Roy, & Mohania, 2006) 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 performing a similar filtering for larger documents and then applying the Phoebus 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

16. www.kbb.com

(Lee et al., 1999; Chaudhuri et al., 2003). 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.

Although our work describes a technique for information extraction, many methods, such as Conditional Random Fields (CRF), assume at least some structure in the extracted attributes to do the extraction. As our extraction experiments show, Phoebus outperforms such methods, such as the Simple Tagger implementation of Conditional Random Fields (McCallum, 2002). Other IE approaches, such as Datamold (Borkar, Deshmukh, & Sarawagi, 2001) and CRAM (Agichtein & Ganti, 2004), 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 (2007) uses a reference set to train a CRF to extract data, which is similar to our PhoebusCRF implementation. However, there are two differences between PhoebusCRF and their work (Bellare & McCallum, 2007). First, the work of Bellare and McCallum (2007) mentions that reference set records are matched using simple heuristics, but it is unclear how this is done. In our 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 ours performs named entity recognition using "Semi-CRFs" with a dictionary component (Cohen & Sarawagi, 2004), 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. Our reference sets are relational data, so finding the match becomes a record linkage task. Further, their work on Semi-CRFs (Cohen & Sarawagi, 2004) 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 (Mansuri & Sarawagi, 2006). This work is similar to ours 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 we 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, we are not the first to consider structured SVMs for information extraction. Previous work used structured SVMs to perform Named Entity Recognition (Tsochantaridis et al., 2005) but their extraction task does not use reference sets.

Our method of aiding information extraction with outside information (in the form of reference sets) is similar to the work on ontology-based information extraction (Embley, Campbell, Jiang, Liddle, Ng, Quass, & Smith, 1999). 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 (Ding, Embley, & Liddle, 2006). However, in contrast to our 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. We believe 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 our work, we can simply add new tuples to our reference set. Lastly, in contrast to our work, this ontology based work assumes contextual and structural rules will apply, making an assumption about the data to extract from. In our work, we make no such assumptions about the structure of the text we are extracting from.

Yet another interesting approach to information extraction using ontologies is the Textpresso system which extracts data from biological text (Müller & Sternberg, 2004). 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.

In recent work on learning efficient blocking schemes Bilenko et al., (2006) 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 (Agrawal, Imielinski, & Swami, 1993). 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 (Mitchell, 1997). On the other hand, the work of mining association rules (Agrawal et al., 1993) 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.

7. Conclusion

This article presents an algorithm for semantically annotating text that is ungrammatical and unstructured. Unstructured, ungrammatical sources contain much information, but cannot support structured queries. Our technique allows for more informative use of the sources. Using our 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. Such semantic annotation is necessary as society transitions into the Semantic Web, where information requires annotation to be useful for agents, but users are unwilling to do the extra work to provide the required annotation.

In the future, our technique could link with a mediator framework (Thakkar, Ambite, & Knoblock, 2004) for automatically acquiring reference sets. This is similar to automatically incorporating secondary sources for record linkage (Michalowski, Thakkar, & Knoblock, 2005). The automatic formulation of queries 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 a more automatic 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 the attributes, as described in Section 3. This disambiguation should lead to much higher accuracy during extraction.

Future work could investigate the inclusion of thesauri for terms in the attributes, with the frequency of replacement of the terms taken into consideration. Also, exploring technologies that automatically construct the reference sets (and eventually thesauri) from the numerous ontologies on the Semantic Web is an intriguing research path.

The long term goal for annotation and extraction from unstructured, ungrammatical sources involves automating the entire process. If the record linkage and extraction methods could become unsupervised, then our approach 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.

Acknowledgments

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), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

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.

References

Agichtein, E., & Ganti, V. (2004). Mining reference tables for automatic text segmentation. In *the Proceedings of the 10th ACM Conference on Knowledge Discovery and Data Mining*, pp. 20 – 29. ACM Press.

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 207–216. ACM Press.

Baxter, R., Christen, P., & Churches, T. (2003). 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*, pp. 25–27.

Bellare, K., & McCallum, A. (2007). Learning extractors from unlabeled text using relevant databases. In *Proceedings of the AAAI Workshop on Information Integration on the Web*, pp. 10–16.

Bilenko, M., Kamath, B., & Mooney, R. J. (2006). Adaptive blocking: Learning to scale up record linkage and clustering. In *Proceedings of the 6th IEEE International Conference on Data Mining*, pp. 87–96.

Bilenko, M., & Mooney, R. J. (2003). Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining*, pp. 39–48. ACM Press.

Borkar, V., Deshmukh, K., & Sarawagi, S. (2001). Automatic segmentation of text into structured records. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 175–186. ACM Press.

Califf, M. E., & Mooney, R. J. (1999). Relational learning of pattern-match rules for information extraction. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pp. 328–334.

Chakaravarthy, V. T., Gupta, H., Roy, P., & Mohania, M. (2006). Efficiently linking text documents with relevant structured information. In *Proceedings of the International Conference on Very Large Data Bases*, pp. 667–678. VLDB Endowment.

Chaudhuri, S., Ganjam, K., Ganti, V., & Motwani, R. (2003). Robust and efficient fuzzy match for online data cleaning. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 313–324. ACM Press.

Cimiano, P., Handschuh, S., & Staab, S. (2004). Towards the self-annotating web. In *Proceedings of the 13th International Conference on World Wide Web*, pp. 462–471. ACM Press.

Ciravegna, F. (2001). Adaptive information extraction from text by rule induction and generalisation.. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pp. 1251–1256.

Cohen, W., & Sarawagi, S. (2004). 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*, pp. 89–98, Seattle, Washington. ACM Press.

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

Cohen, W. W., Ravikumar, P., & Feinberg, S. E. (2003). 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*, pp. 13–18.

Crescenzi, V., Mecca, G., & Merialdo, P. (2001). Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pp. 109–118. VLDB Endowment.

Ding, Y., Embley, D. W., & Liddle, S. W. (2006). Automatic creation and simplified querying of semantic web content: An approach based on information-extraction ontologies. In *Proceedings of the Asian Semantic Web Conference*, pp. 400–414.

Dingli, A., Ciravegna, F., & Wilks, Y. (2003). Automatic semantic annotation using unsupervised information extraction and integration. In *Proceedings of the K-CAP Workshop on Knowledge Markup and Semantic Annotation*.

Elfeky, M. G., Verykios, V. S., & Elmagarmid, A. K. (2002). TAILOR: A record linkage toolbox. In *Proceedings of 18th International Conference on Data Engineering*, pp. 17–28.

Embley, D. W., Campbell, D. M., Jiang, Y. S., Liddle, S. W., Ng, Y.-K., Quass, D., & Smith, R. D. (1999). Conceptual-model-based data extraction from multiple-record web pages. *Data Knowledge Engineering*, *31*(3), 227–251.

Fellegi, I. P., & Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association*, *64*, 1183–1210.

Handschuh, S., Staab, S., & Ciravegna, F. (2002). S-cream - semi-automatic creation of metadata. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management*, pp. 165–184. Springer Verlag.

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

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

Jaro, M. A. (1989). Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, *89*, 414–420.

Joachims, T. (1999). *Advances in Kernel Methods - Support Vector Learning*, chap. 11: Making large-Scale SVM Learning Practical. MIT-Press.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, pp. 282–289. Morgan Kaufmann.

Lee, M.-L., Ling, T. W., Lu, H., & Ko, Y. T. (1999). Cleansing data for mining and warehousing. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications*, pp. 751–760. Springer-Verlag.

Lerman, K., Gazen, C., Minton, S., & Knoblock, C. A. (2004). Populating the semantic web. In *Proceedings of the AAAI Workshop on Advances in Text Extraction and Mining*.

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

Mansuri, I. R., & Sarawagi, S. (2006). Integrating unstructured data into relational databases. In *Proceedings of the International Conference on Data Engineering*, p. 29. IEEE Computer Society.

McCallum, A. (2002). Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu.

McCallum, A., Nigam, K., & Ungar, L. H. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the 6th ACM SIGKDD*, pp. 169–178.

Michalowski, M., Thakkar, S., & Knoblock, C. A. (2005). Automatically utilizing secondary sources to align information across sources. In *AI Magazine, Special Issue on Semantic Integration*, Vol. 26, pp. 33–45.

Michelson, M., & Knoblock, C. A. (2005). Semantic annotation of unstructured and ungrammatical text. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pp. 1091–1098.

Michelson, M., & Knoblock, C. A. (2006). Learning blocking schemes for record linkage. In *Proceedings of the 21st National Conference on Artificial Intelligence*.

Michelson, M., & Knoblock, C. A. (2007). Unsupervised information extraction from unstructured, ungrammatical data sources on the world wide web. *International Journal of Document Analysis and Recognition (IJDAR), Special Issue on Noisy Text Analytics*.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York.

Müller, H.-M., & Sternberg, E. E. K. P. W. (2004). Textpresso: An ontology-based information retrieval and extraction system for biological literature. *PLoS Biology*, *2*(11).

Muslea, I., Minton, S., & Knoblock, C. A. (2001). Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, *4*(1/2), 93–114.

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

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130–137.

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, *147*, 195–197.

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

Thakkar, S., Ambite, J. L., & Knoblock, C. A. (2004). 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*.

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the 21st International Conference on Machine Learning*, p. 104. ACM Press.

Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, *6*, 1453–1484.

Vargas-Vera, M., Motta, E., Domingue, J., Lanzoni, M., Stutt, A., & Ciravegna, F. (2002). MnM: Ontology driven semi-automatic and automatic support for semantic markup. In *Proceedings of the 13th International Conference on Knowledge Engineering and Management*, pp. 213–221.

Wellner, B., McCallum, A., Peng, F., & Hay, M. (2004). 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*, pp. 593–601.

Winkler, W. E., & Thibaudeau, Y. (1991). An application of the fellegi-sunter model of record linkage to the 1990 U.S. Decennial Census. Tech. rep., Statistical Research Report Series RR91/09 U.S. Bureau of the Census.

Zhai, C., & Lafferty, J. (2001). 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*, pp. 334–342. ACM Press.