Semantic annotation of unstructured and ungrammatical text

Matthew Michelson and Craig A. Knoblock
University of Southern California
Information Sciences Institute,
4676 Admiralty Way
Marina del Rey, CA 90292 USA
{michelso,knoblock}@isi.edu

Abstract

There are vast amounts of free text on the internet that are neither grammatical nor formally structured, such as item descriptions on Ebay or internet classifieds like Craig's list. These sources of data, called "posts," are full of useful information for agents scouring the Semantic Web, but they lack the semantic annotation to make them searchable. Annotating these posts is difficult since the text generally exhibits little formal grammar and the structure of the posts varies. However, by leveraging collections of known entities and their common attributes, called "reference sets," we can annotate these posts despite their lack of grammar and structure. To use this reference data, we align a post to a member of the reference set, and then exploit this matched member during information extraction. We compare this extraction approach to more traditional information extraction methods that rely on structural and grammatical characteristics, and we show that our approach outperforms traditional methods on this type of data.

1 Introduction

The Semantic Web will revolutionize the use of the internet, but the idea faces some major challenges. First, construction of the Semantic Web requires a lot of extra markup on documents, but this work should not be forced upon everyday users. Second, there is a lot of information that would be more useful if it were annotated for the Semantic Web, but the nature of the data makes it difficult to do so. Examples of this type of data are the text of EBay posts, internet classifieds like Craig's list, bulletin boards such as Bidding For Travel,[1] or even the summary text below the hyperlinks returned after querying Google. We call each piece of text from these sources a "post." It would be beneficial to add semantic annotation to such posts, like that shown in Figure 1, but the annotation task should carry no burden to human users.

Information extraction (IE) can be used to extract and semantically annotate pieces of some text. However, IE on posts is especially difficult because the data is neither structured enough to use wrapper technologies such as Stalker [Muslea *et al.*, 2001] nor grammatical enough to exploit Natural Language Processing (NLP) techniques such as those used in Whisk [Soderland, 1999].

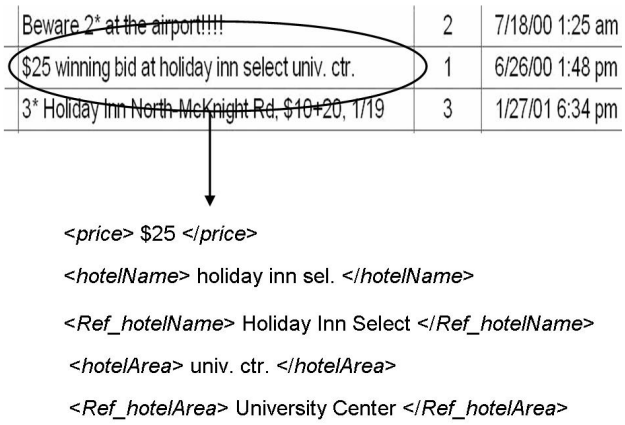

Figure 1: A post from Bidding For Travel

This lack of grammar and structure can be overcome by adding knowledge to IE. This extra knowledge consists of collections of known entities and their common attributes, which we call "reference sets." A reference set can be an online set of reference documents, such as the CIA World Fact Book. It can also be an online (or offline) database, such as the Comics Price Guide.[2] With the Semantic Web we envision building reference sets from the numerous ontologies that already exist. Continuing with our hotel example from Figure 1, assume there is an ontology of U.S. hotels, and from it we build a reference set with the following attributes: city, state, star rating, hotel name, area name, etc.

To use reference sets for semantic annotation we exploit the reference set to determine which, if any, of the attributes appear in the post. To do this, we first determine which member of the reference set best matches the post. We call this the record linkage step. Then we exploit the attributes of this

[1] www.biddingfortravel.com

[2] www.comicspriceguide.com

reference set member for the information extraction step by identifying and labeling attributes from the post that match those from the matching member of the reference set. We annotate the post in this manner.

For instance, the circled hotel post in Figure 1 matches the reference set member with the hotel name of "Holiday Inn Select" and the hotel area of "University Center." Using this match we label the tokens "univ. ctr." of the post as the "Hotel Area," since they match the hotel area attribute of the matching reference set record. In this manner we annotate all of the attributes in the post that match those of the reference set. Figure 2 illustrates our approach on the example post from Figure 1. For purposes of exposition, the reference set shown only has 2 attributes: name and area.

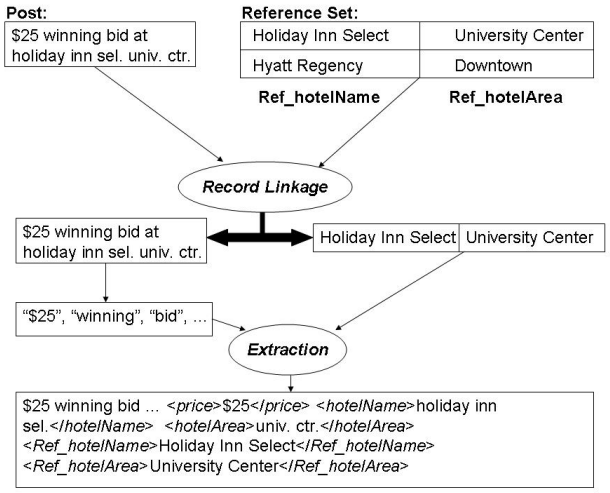

Figure 2: Annotation Algorithm

In addition to annotating attributes in the post from the reference set, we also annotate attributes that are identifiable, but not easily represented in reference sets. Examples of such attributes include prices or dates. This is shown by the price of Figure 1. Also, we include annotation for the attributes of the matching reference member. (These are called "Ref_hotel..." in Figure 1.) Since attribute values differ across posts, these reference member attributes provide a set of normalized values for querying. Also, the reference set attributes provide a simple visual validation to the user that the IE step identified things correctly. Lastly, by including attributes from the matching reference member, we can provide values for attributes that were not included in the post. In our example post of Figure 1, the user did not include a star rating. By including the reference set member's attribute for this star rating, we add useful annotation for information that was not present initially.

This paper describes our algorithm for semantic annotation using reference sets. Section 2 describes the record linkage step and Section 3 describes the extraction step. Section 4 presents experimental evaluation, and section 5 presents discussion of results. Section 6 presents related research, and section 7 describes our conclusions.

2 Aligning Posts to a Reference Set

To correctly parse the attributes from the post, we need to first decide what those attributes are. To aid in this process, we match the post to a member of the reference set.

Since it is infeasible to compare the post to all members of the reference set, we construct a set of candidate matches in a process known as "blocking". Many blocking methods have been proposed in the record linkage community (see [Baxter *et al.*, 2003] for a recent survey of some), but the basic idea is to cluster candidates together around a blocking key. In our work, a candidate for a post is any member of the reference set that shares some n-gram with that post. The choice of algorithm here is independent of the overall alignment algorithm.

Next we find the candidate that best matches the post. That is, we must align one data source's record (the post) to a record from the other data source (the reference set candidates). This alignment is called record linkage [Fellegi and Sunter, 1969].

However, our record linkage problem differs 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. Yet the attributes of the posts are embedded within a single piece of text. We must match this text to the reference set, which is already decomposed into attributes and which does not have the extraneous tokens present in the post. With this type of matching traditional record linkage approaches do not apply.

Instead, we create a vector of scores, V_{RL}, for each candidate. V_{RL} is composed of similarity scores between the post and each attribute of the reference set. We call these scores *RL_scores*. V_{RL} also includes *RL_scores* between the post and all of the attributes concatenated together. In the example reference set from Figure 2, the schema has 2 attributes *<Hotel Name, Hotel Area>*. If we assume the current candidate is <"Hyatt", "PIT Airport">. Then we define V_{RL} as:

V_{RL}=<*RL_scores(post,"Hyatt")*,
RL_scores(post,"PIT Airport"),
RL_scores(post,"Hyatt PIT Airport")>

Each *RL_scores(post,attribute)* is itself a vector, composed of three other vectors:

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

The three vectors that compose *RL_scores* represent different similarity types. The vector *token_scores* is the set of token level similarity scores between the post and the attribute, including Jensen-Shannon distance (both with a Dirichlet prior and a Jelenik-Mercer mixture model) and Jaccard similarity. The vector *edit_scores* consists of the following edit distance functions: Smith-Waterman distance, Levenstein distance, Jaro-Winkler similarity and character level Jaccard similarity. (All of the scores in *token_scores* and *edit_scores* are defined in [Cohen *et al.*, 2003].) Lastly, the vector *other_scores* consists of scores that did not fit into either of the other categories, specifically the Soundex score between the post and the attribute and the Porter stemmer score between the two.

Using *RL_scores* of just the reference attributes themselves gives a notion of the field similarity between the post and the match candidate. The *RL_scores* that uses the concatenation of the reference attributes gives an idea of the record level match. Since the post is not yet broken into attributes, this is how we determine these field and record level similarities. We could use only the concatenated scores since all we desire is a record level match. However, it is possible for different records in the reference set to have the same record level score, but different scores for the attributes. If one of these records had a higher score on a more discriminative attribute, we would like to capture that.

After all of the candidates are scored, we then rescore each V_{RL}. For each element of V_{RL}, the candidates with the maximum value at that index map this element to 1. The rest of the candidates map this element to 0.

For example, assume we have 2 candidates, with the vectors $V1_{RL}$ and $V2_{RL}$:

$$V1_{RL} = <.999,1.2,...,0.45,0.22>$$
$$V2_{RL} = <.888,0.0,...,0.65,0.22>$$

After rescoring they become:

$$V1_{RL} = <1,1,...,0,1>$$
$$V2_{RL} = <0,0,...,1,1>$$

The rescoring helps to determine the best possible candidate match for the post. Since there might be a few candidates with similarly close values, and only one of them is a best match, the rescoring separates out the best candidate as much as possible.

After rescoring, we pass each V_{RL} to a Support Vector Machine (SVM) [Joachims, 1999] trained to label them as matches or non-matches. When the match for a post is found, the attributes of the matching reference set member are added as annotation to the post.

3 Extracting Data from Posts

To exploit the reference set for extraction we use the attributes of the best match from the reference set as a basis for identifying similar attributes in the post.

To begin the extraction process, we break the post into tokens. In our example from Figure 2, the post becomes the set of tokens, {"$25", "winning", "bid",...}. Each of these tokens is then scored against each attribute of the record from the reference set that was deemed the match. This scoring consists of building a vector of scores which we call V_{IE}. Although similar to V_{RL}, the V_{IE} vector does not include the vector *token_scores* because we are comparing single tokens of the post. Instead, we include a vector *common_scores*. The *common_scores* vector includes user defined functions such as regular expressions, which help identify the attributes that are not present in the reference set, such as price or date. So, V_{IE} consists of the vector *common_scores* and an *IE_scores* vector between each token and each attribute from the reference set.

Our *RL_scores* vector from the record linkage step,

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

becomes,

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

So, using our example from Figure 2, we match the post to the reference set member {"Holiday Inn Select","University Center"}. So, if we generate the V_{IE} for the post token "univ." it would look like:

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

Since each V_{IE} is not a member of a cluster where the winner takes all, there is no binary rescoring.

Each V_{IE} is then passed to a multiclass SVM [Tsochantaridis *et al.*, 2004] trained to give it a class label, such as hotel name, hotel area, or price. Intuitively, similar attribute types should have a similar V_{IE}. We expect that hotel names will generally have high scores against the reference set attribute of hotel names, and small scores against the other attributes, and the vector will reflect this.

The SVM learns that any vector that does not look like anything else should be labeled as "junk", which can then be ignored. This is an important idea because without the benefits of a reference set this would be an extraordinarily difficult task. If features such as capitalization and token location were used, who is to say "Great Deal" is not a car name? Also, many traditional IE systems that work in this unique domain of ungrammatical, 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 when users are entering text.

However, by treating each token in isolation there is a chance that a junk token will be mislabeled. For example, a junk token might have enough letters to be labeled as a *hotel area*. Then when we extract the *hotel area* from the post, it will have a noisy token. Therefore, labeling each token individually gives an approximation of the data to be extracted.

To improve extraction, we can exploit the power of the reference set by comparing the whole extracted field to its analogue reference set attribute. Thus, once all of the tokens from a post are processed and we have whole attributes labeled, we take each attribute and compare it to the corresponding one from the reference set. Then we can remove the tokens that introduce noise in the extracted attribute.

To do this, we first get two baseline scores between the extracted attribute and the reference set attribute. One is a Jaccard similarity, which demonstrates token level similarity. However, since there are many misspellings and such, we also need an edit-distance based similarity metric. For this we use the Jaro-Winkler metric. These baselines give us an idea of how accurately we performed on the approximate extraction. Using our post from Figure 2, assume the phrase "holiday inn sel. univ. ctr." was "holiday inn sel. ***in*** univ. ctr.". In this case, we might extract "holiday inn sel. in" as the *hotel name*. In isolation, the token "in" could be the "inn" of a hotel name. Comparing this extracted hotel name to the reference attribute, "Holiday Inn Select," we get a Jaccard similarity of 0.4 and a Jaro-Winkler score of 0.87.

Next, we go through the extracted attribute, removing one token at a time and calculating the new Jaccard and Jaro-Winkler similarities. If both of these new scores are higher than the baseline, then that token is a candidate for removal. Once all of the tokens are processed in this way, the candidate for removal that has the highest scores is removed, and we repeat the whole process. The process ends when there are no more tokens that yield improved scores when they are removed. In our example, we find that "in" is a removal candidate since it yields a Jaccard score of 0.5 and a Jaro-Winkler score of 0.92. Since it has the highest scores after the iteration, it is removed. Then we see that removing any of the remaining tokens does not provide improved scores, so the process ends.

Aside from increasing the accuracy of extraction, this approach has the added benefit of disambiguation. For instance, a token might be both a *hotel name* and a *hotel area*, but not both at the same time? As an example, "airport" could be part of a *hotel name* or a *hotel area*. In this case, we could label it as both, and the above approach would remove the token from the attribute that it is not. However, our implementation currently assigns only one label per token, so we did not test this disambiguation technique.

Thus, the whole extraction process takes a token of the text, creates the V_{IE} and passes this to the SVM which generates a label for the token. Then each field is cleaned up and we add the annotation to the post. This produces the output shown at the end of Figure 2.

4 Results

To validate our approach, we implemented our algorithm in a system named Phoebus and tested the technique in two domains: hotel postings and comic books.

In the *hotel* domain, we attempt to parse the hotel name, hotel area, star rating of the hotel, price and dates booked from the Bidding For Travel website. This site is a forum where users share successful bids for Priceline. We limited our experiment to posts about hotels in Sacramento, San Diego and Pittsburgh. As a reference set, we use the Bidding For Travel hotel guides. These guides are special posts that list all of the hotels that have ever been posted about in a given area. These posts provide the hotel name, hotel area and the star rating, which are used as the reference set attributes.

The *comic* domain uses posts from EBay about comic books for sale searched by keyword "Incredible Hulk" and "Fantastic Four". Our goal is to parse the title, issue number, price, condition, publisher, publication year and the description from each post. (Note: the description is a few word description commonly associated with a comic book, such as *1st appearance the Rhino*.) As a reference set for this domain, we used the Comics Price Guide[3] for lists of all of the Incredible Hulk and Fantastic Four comics, as well as a list of all possible comic book conditions. In this case, the reference set included the attributes title, issue number, description, condition and publisher.

Experimentally, we split the posts in each domain into 2 folds, one for training and one for testing, where the training fold was 30% of the total posts, and the testing fold was the remaining 70%. We ran 10 trials and report the average results for these 10 trials.

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

4.1 Alignment Results

Our approach hinges on exploiting reference sets, so the alignment step should perform well. We report the results of the alignment step in Table 1. According to the usual record linkage statistics we define:

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

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

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

We compare our record linkage approach to WHIRL [Cohen, 2000]. WHIRL is record linkage system that does soft joins across tables by computing vector-based cosine similarities between the attributes. All other record linkage systems require matching based on decomposed fields, so WHIRL served as a benchmark because it does not have this requirement. As input to WHIRL, one table was the test set of posts (70% of the posts) and the other table was the reference set with the attributes concatenated together. As in our record linkage step, using the concatenation of attributes can best mirror finding a record level match. This was also done because joining across each reference set attribute separately leaves no way to combine the matches for these queries. For example, would we only count the reference set members that score highest for every attribute as matches?

We ran a similarity join across these tables, which produced a list of matches, ordered by descending similarity score. For each post that had matches from the join, the reference set member with the highest similarity score was called its match. These results are also reported in Table 1. Since Phoebus is able to represent both an attribute level and record level similarity in its score, using more than just token based cosine similarity, it outperformed WHIRL.

	Prec.	Recall	F-measure
Hotel			
Phoebus	93.60	91.79	**92.68**
WHIRL	83.52	83.61	83.13
Comic			
Phoebus	93.24	84.48	**88.64**
WHIRL	73.89	81.63	77.57

Table 1: Record linkage results

4.2 Extraction Results

We also performed experiments to validate our approach to extraction. Specifically, we compare our technique to two other IE methods as baselines.

One baseline is Simple Tagger from the MALLET [McCallum, 2002] suite of text processing tools. Simple Tagger

is an implementation of Conditional Random Fields (CRF). CRFs have been effectively used in IE. As an example, one algorithm combines information extraction and coreference resolution using CRFs [Wellner *et al.*, 2004].

We also present our results versus Amilcare [Ciravegna, 2001], which uses shallow Natural Language Processing for information extraction. It has been empirically shown that Amilcare does much better in extraction versus other symbolic systems [Ciravegna, 2001]. It presents a good benchmark versus NLP based systems, which we expect will not do well on our domains. For the tests, we supplied our reference data as gazetteers to Amilcare.

Our IE technique includes scores that we deemed "common" scores, which are used to help identify attributes not in the reference set. We make these clear for each domain, since they are the only domain specific scores we include. For the hotel domain, our common scores are *matchPriceRegex* and *matchDateRegex*, which give a positive score if a token matches a price or date regular expression, and 0 otherwise. For the comic domain, we use *matchPriceRegex* and *matchYearRegex*.

We present our results using Precision, Recall and F-Measure as defined above. Tables 2 and 3 show the results of correctly labeling the tokens within the posts with the correct attribute label for the Hotel and Comic domains, respectively. Attributes in *italics* are attributes that exist in the reference set. The column Freq shows the average number of tokens that have the associated label.

Table 4 shows the results reported for all possible tokens, which is a weighted average, since some attribute types are more frequent than others. Also included in Table 4 are "field level" summary results. Field level results regard a piece of extracted information as correctly labeled only if all of the tokens that should be present are, and there are no extraneous tokens. In this sense, it is a harsh, all or nothing metric, but it is a good measure of how useful a technique would really be.

We tested the F-Measures for statistical significance using a two-tailed paired t-test with α=0.05. In Table 2 the only F-Measure difference that was not significant was the *Star* attribute between Phoebus and Simple Tagger. In Table 3 the F-Measures for Price were not significant between Phoebus and Simple Tagger and between Phoebus and Amilcare. In Table 4 all differences in F-Measure proved statistically significant.

Phoebus outperforms the other systems on almost all attributes, and for all summary results. There were 3 attributes where Phoebus was outperformed, and two of these warrant remarks. (We ignore hotel name since it was so similar.) On Comic titles, Phoebus had a much lower recall because it was unable to extract parts of titles that were not in the reference set. Consider the post, "The incredible hulk and Wolverine #1 Wendigo". In this post, Phoebus extracts "The incredible hulk," but the actual title is "The incredible hulk and Wolverine." In this case, the limited reference set hindered Phoebus, but this could be corrected by including more reference set data, such as other comic book price guides.

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

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

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

Table 2: Extraction results: Hotel domain

Comic					
		Prec.	Recall	F-Measure	Freq
Condition	Phoebus	91.80	84.56	**88.01**	410.3
	Simple Tagger	78.11	77.76	77.80	
	Amilcare	79.18	67.74	72.80	
Descript.	Phoebus	69.21	51.50	59.00	504.0
	Simple Tagger	62.25	79.85	**69.86**	
	Amilcare	55.14	58.46	56.39	
Issue	Phoebus	93.73	86.18	**89.79**	669.9
	Simple Tagger	86.97	85.99	86.43	
	Amilcare	88.58	77.68	82.67	
Price	Phoebus	80.00	60.27	**68.46**	10.7
	Simple Tagger	84.44	44.24	55.77	
	Amilcare	60.0	34.75	43.54	
Publisher	Phoebus	83.81	95.08	**89.07**	61.1
	Simple Tagger	88.54	78.31	82.83	
	Amilcare	90.82	70.48	79.73	
Title	Phoebus	97.06	89.90	93.34	1191.1
	Simple Tagger	97.54	96.63	**97.07**	
	Amilcare	96.32	93.77	94.98	
Year	Phoebus	98.81	77.60	**84.92**	120.9
	Simple Tagger	87.07	51.05	64.24	
	Amilcare	86.82	72.47	78.79	

Table 3: Extraction results: Comic domain

5 Discussion

One drawback when using supervised learning systems is the cost to label training data. However, our entire algorithm gen-

	Hotel					
	Token level			Field level		
	Prec.	Recall	F-Mes.	Prec.	Recall	F-Mes.
Phoebus	93.60	91.79	**92.68**	87.44	85.59	**86.51**
Simple Tagger	86.49	89.13	87.79	79.19	77.23	78.20
Amilcare	86.12	86.14	86.11	85.04	78.94	81.88
	Comic					
	Token level			Field level		
	Prec.	Recall	F-Mes.	Prec.	Recall	F-Mes.
Phoebus	93.24	84.48	**88.64**	81.73	80.84	**81.28**
Simple Tagger	84.41	86.04	85.43	78.05	74.02	75.98
Amilcare	87.66	81.22	84.29	90.40	72.56	80.50

Table 4: Summary extraction results

eralizes well and can perform well with little training data. Table 5 presents summary, token level results for extraction. Here we trained Phoebus on 10% of the posts, and then tested on the other 90%. In both domains, the results are similar to those of the experimental set up, which used 30% of the data for training.

	Prec.	Recall	F-measure
Hotel (30%)	93.60	91.79	92.68
Hotel (10%)	93.66	90.93	92.27
Comic (30%)	93.24	84.48	88.64
Comic (10%)	91.41	83.63	87.34

Table 5: Phoebus: Trained on 10% of the data

Since we return the reference set attributes as annotation, we need to examine whether or not this annotation is valid. To do this, we need a way to link the results from performing record linkage to extraction results. Note we can consider the fact that a correct match during record linkage is the same as correctly identifying those attributes from the reference set in the post, at the field level. In our ongoing example from Figure 2, when we match the post to the reference member with a hotel name of "Holiday Inn Select", it is like extracting this hotel name from the post. Thus, we can consider our record linkage results to be field level extraction results for the attributes in the reference set.

Using the reference set attributes as annotation has some interesting implications. For instance, the attributes from the reference set provide a normalized platform of values for querying the data. Also, returning reference set attributes for types not found in the post provides information that might have been missing previously. As an example, a post might leave out the star rating. Yet, now we have one from the reference set record upon which to query. Another interesting implication arises as a solution for the cases where extraction is hard. None of the systems extracted the comic description well. However, by including the reference set description, we consider the record linkage results as how effectively we extracted (and thus labeled), a description. This yields an improvement of over 20% for precision, and almost 10% for recall.

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

Lastly, the more discriminative information that we can pass to the SVM, the better it will perform. Assume that all we wanted to do is extract price and date from hotel posts, and we would get the rest of the annotation from the reference set. We would still want to train the SVM to extract the attributes of the reference set, because it would then know that by classifying a certain piece of information as a certain attribute, it is not another piece of information. That is to say, it is just as important to recognize what a token is not. For example, consider a reference set hotel named "The $41 Inn". Then when post matches this reference set member, and we see the token "$41", we know that it is most likely a hotel name. If we did not train to extract all of the attributes, this would be classified as a price, since the SVM has no notion of a hotel name.

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

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

Consider the following two cases. If we want to extract conference names and cities but we only use one reference set, it would have to contain the power set of cities crossed with conference names. However, if we have two reference sets, one for each attribute, we can run the algorithm once with the conference name data, and once with a reference set of cities.

The next interesting case happens when a post contains more than one of the same attribute. For example, we want to extract two cities from some post. If we use one reference set, then it would include the cross product of all cities. However, we can use a single reference set of city names if we slightly modify the algorithm. We make 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, we just choose the first match. Using this reference city we then extract the city from the post, and remove it from the post. Then we simply run the process again, which will catch the second city, using the same, single reference set. This could be repeated as many times as needed.

6 Related Work

Our work is motivated by the goal that the cost of annotating documents for the Semantic Web should be free, that is, automatic and invisible to users [Hendler, 2001]. Many researchers have followed this path, attempting to automatically mark up documents for the Semantic Web, as we propose here [Vargas-Vera *et al.*, 2002; Handschuh *et al.*, 2002; Cimiano *et al.*, 2004; Dingli *et al.*, 2003]. 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 our post data. In a similar vein, there are systems such as ADEL [Lerman *et al.*, 2004] which rely on the structure to identify and annotate records in web pages. Again, the failure of our data to exhibit structure makes this approach inappropriate. So, while there is a fair amount of work in automatic labeling, there is not much emphasis on techniques that could do this on text that is unstructured and ungrammatical.

While the idea of record linkage is not new [Fellegi and Sunter, 1969] and is well studied even now [Bilenko and Mooney, 2003], most of the focus for this work matches one set of records to another set of records based on their decomposed attributes. There is little work on matching data sets where one record is a single string composed of the other data set's attributes to match on, as in our case. The WHIRL system [Cohen, 2000] allows for record linkage without decomposed attributes, but as shown in Section 4.1 we outperform WHIRL by exploiting a larger set of features to represent both a field and record level similarity.

Using the reference set's attributes as normalized values is similar to the idea of data cleaning. However, most of the data cleaning algorithms assume that there are tuple-to-tuple transformations [Lee *et al.*, 1999; Chaudhuri *et al.*, 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 our data, where all of the attributes are embedded within the post, which maps to a set of attributes from the reference set.

Information extraction can semantically annotate data, which is why we chose to compare our technique to other IE approaches, such as the Simple Tagger Conditional Random Field [McCallum, 2002]. Other IE approaches, such as Datamold [Borkar *et al.*, 2001] and CRAM [Agichtein and Ganti, 2004], segment whole records (like bibliographies) into attributes. However, both of these systems require that every token of a record receive a label, which is not possible with posts that are filled with irrelevant tokens. CRAM is also similar in its use of reference sets for extraction. However, they assume that the reference set members already match the data for extraction, while we do this record linkage automatically. Another IE approach similar to ours performs named entity recognition using a dictionary component [Cohen and Sarawagi, 2004]. However, this technique requires that entire segments have the same class label, while our technique can handle the case where an attribute is broken up in the middle by another attribute, say a hotel name interrupted by a hotel area.

7 Conclusion

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

In the future, we we would like to link this technique with a mediator [Thakkar *et al.*, 2004] framework for automatically acquiring reference sets. This is similar to automatically incorporating secondary sources for record linkage [Michalowski *et al.*, 2005]. How to automatically formulate a query to retrieve the correct domain reference set is a direction of future research. Also, our current implementation only gives one class label per token. Ideally we would give a token all possible labels, and then remove the extraneous tokens when we clean up the attributes, as described in Section 3.

8 Acknowledgements

We would like to thank William Cohen, Andrew McCallum and Fabio Ciravegna for allowing public use of their systems and code. Also, thanks to Kristina Lerman and Snehal Thakkar for their comments.

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

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

References

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

[Baxter *et al.*, 2003] Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for

record linkage. In *Proceedings of the ACM Workshop on Data Cleaning, Record Linkage, and Object Identification*, 2003.

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

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

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

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

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

[Cohen and Sarawagi, 2004] William Cohen and Sunita Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *Proceedings of the 10th ACM Int'l Conf. Knowledge Discovery and Data Mining*, Seattle, Washington, August 2004. ACM Press.

[Cohen *et al.*, 2003] William W. Cohen, Pradeep Ravikumar, and Stephen E. Feinberg. A comparison of string metrics for matching names and records. In *Proceedings of the Workshop on Data Cleaning and Object Consoliation*, 2003.

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

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

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

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

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

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

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

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

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

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

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

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

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

[Tsochantaridis *et al.*, 2004] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, August 2004.

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

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