Exploiting Background Knowledge to Build Reference Sets for Information Extraction*

Matthew Michelson†
Fetch Technologies
841 Apollo St, Ste. 400
El Segundo, CA 90245 USA
mmichelson@fetch.com

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

Abstract

Previous work on information extraction from unstructured, ungrammatical text (e.g. classified ads) showed that exploiting a set of background knowledge, called a "reference set," greatly improves the precision and recall of the extractions. However, finding a source for this reference set is often difficult, if not impossible. Further, even if a source is found, it might not overlap well with the text for extraction. In this paper we present an approach to building the reference set directly from the text itself. Our approach eliminates the need to find the source for the reference set, and ensures better overlap between the text and reference set. Starting with a small amount of background knowledge, our technique constructs tuples representing the entities in the text to form a reference set. Our results show that our method outperforms manually constructed reference sets, since hand built reference sets may not overlap with the entities in the unstructured, ungrammatical text. We also ran experiments comparing our method to the supervised approach of Conditional Random Fields (CRFs) using simple, generic features. These results show our method achieves an improvement in F_1-measure for 6/9 attributes and is competitive in performance on the others, and this is without training data.

1 Introduction

The Web is full of unstructured, ungrammatical data such as online classified ads, auction listings, and forum postings. We call such unstructured, ungrammatical text "posts." However, querying posts beyond keyword search is difficult because posts are not typically structured to support queries such as joins, selections, and aggregations. Information extraction can structure posts by identifying the attributes embedded within each post. For instance, the post of Figure 1 shows a post for a car classified, with some of the useful attributes (such as the make, model, and trim) identified. After extraction, we can support queries such as "select all Hondas."

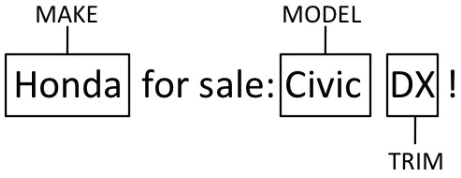

Figure 1: A car post with extracted attributes

However, the structure of posts varies greatly, so information extraction techniques that exploit the similarity of structures in text (such as consistent HTML tags) will not work [Kushmerick *et al.*, 1997]. Further, posts are not grammatical (they are largely unparsable), so natural language processing approaches to extraction are not useful either [Ciravegna, 2001]. Recent work on extraction from posts instead uses "reference sets" [Michelson and Knoblock, 2007; 2008]. A reference set is a list of entities with the associated attributes. Using the car example of Figure 1, a reference set could be a full set of cars from a car information website, such as Edmunds.com. Each reference-set tuple has a make attribute, a model attribute, etc. To exploit a reference set for extraction, first the algorithms find the best match for each post from the reference set, and then that reference set tuple provides clues for extraction.

Although previous work successfully uses reference sets, the construction of a reference set poses problems. For example, although in some cases it is not hard to manually create the reference set by pulling the data from a website, sometimes it is not possible to find a single comprehensive source.

*This research is based upon work supported in part by the National Science Foundation under award number IIS-0324955; in part by the Air Force Office of Scientific Research under grant number FA9550-07-1-0416; and in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004. The United States Government is authorized to reproduce and distribute reports for Governmental purposes not withstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

†Work done while at USC Information Sciences Institute.

For instance, finding a single source of car data to scrape is easy. Yet, if a user needs a reference set of laptops, containing a manufacturer (Dell), a model (Latitude), and a model number (d600), finding a single source that enumerates all of the laptop model numbers is challenging, if not impossible. There are websites that contain all of the Dell Latitude model numbers, but those many not contain the list of Acer Aspire model numbers or IBM Thinkpad model numbers. Therefore, a user must find multiple sources to produce an exhaustive list of laptop model numbers, and then combine the results (e.g., eliminating duplicates). Further, and more importantly, a user might build a complete reference set that covers all IBM, Apple, and Dell laptops, only to discover that the auction listings for the laptops the user wants to extract from contain only Apple and Acer laptops. This is a coverage issue because the reference set does not overlap well with the set of posts, since the Acer laptops are excluded from the reference set. Coverage issues also occur because of the dynamic nature of online data. For instance, a source of laptops may update their catalog every few weeks to incorporate new models, rendering the previously built reference set as incomplete.

In this paper we present a technique to construct the reference set from the posts themselves. This alleviates the need to find sources or databases as reference sets. Further, using the posts themselves to build the reference set deals with the coverage issue since the entities in the reference set are generated directly from the posts and so overlap by definition.

Our approach starts with a set of "seed" values. These seeds are the smallest (and most obvious) knowledge a user can provide for a given domain. For instance, if a user wanted to construct a reference set of laptops, the seeds would be the manufacturers such as IBM, Dell, Apple, etc. It is not hard to find websites that enumerate reference sets at the most obvious attribute level, such as a list of car makes or laptop manufacturers. Using this list as starting seeds will constrain what tuples can be constructed from the posts themselves, and build a cleaner reference set. By using the posts themselves to fill in the more specific values in the reference set, we still have the benefits of coverage between the reference set and the posts, and we eliminate the difficulty in finding multiple sources to cover the most specific attributes.

Our results show that the reference set generated by our seed-based technique is cleaner (and hence more accurate for extraction) than using the same approach without the constraint of the seeds. Also, since our method uses the posts themselves it has better coverage (and hence more accurate extraction) than full, manually constructed reference sets, especially for the more specific attributes which are precisely those that are hard to find enumerated in a single source. Although we use the reference set for extraction, reference sets can be used in a number of other tasks such as taxonomy creation, text understanding, and information retrieval.

The rest of this paper is organized as follows. Section 2 describes our method for seed-based reference set construction. Section 3 presents our experimental results, where our technique outperforms manual reference sets for extraction, and largely outperforms state-of-the-art machine learning for this information extraction task. Section 4 describes related work, and Section 5 presents our conclusions and future directions.

Make	*Model*
Honda	Accord
Honda	Civic
Ford	Focus

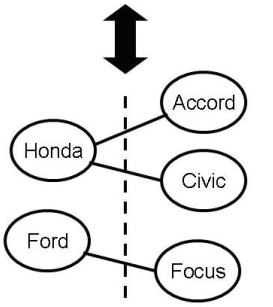

Figure 2: A reference set and its entity trees

2 Seed-Based Reference Set Construction

The intuition for constructing reference sets from posts is that reference set tuples often form hierarchy-like structures, which we call "entity trees." Consider the simple reference set of three cars in Figure 2. Each tuple in the reference set of cars has two attributes, a make and a model. The model is a more specific version of the make (i.e. an Accord is a Honda), and so the Accord attribute value is a child node to the Honda value. The same can be said of the Civic value, and so we can generate an entity tree rooted on the value Honda, with two children: Civic and Accord. The same argument applies to turn the Ford tuple into its own entity tree. However, the entity trees rooted on Honda and Ford are disjoint, since they do not share any ancestors. So a reference set is really a forest of entity trees. So, once an algorithm constructs the set of entity trees, it can then traverse them, outputting a tuple for each path from root to leaf, and the union of all these tuples creates a reference set. So, to construct a reference set from posts, the goal is really to build entity trees from the posts.

To build entity trees we use a modified version of the Sanderson and Croft (1999) heuristic for finding token subsumptions, with the notion that if token x subsumes y, then y is a child node under x in the entity tree that contains x. We define the rule for subsumption given terms x and y as:[1]

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

As an example, consider the four example posts shown in Table 1. If we consider the token pair "Honda" with "Civic" we see that the conditional probability of Honda given Civic (P(Honda|Civic)) is 1.0, while P(Civic|Honda) is 0.5 (since Honda occurs in four posts while Honda occurs with Civic in two). Therefore, the subsumption rule fires, and Civic becomes a child node to Honda. Similarly, Accord becomes a child of Honda, resulting in the entity tree of Figure 2.

Therefore, our first step for generating a reference set is to break the set of posts into bigrams, and then check each

[1]Note, we require terms x and y to co-occur at least once for this rule to be considered.

Table 1: Four example posts

Honda civic is cool
Look at cheap Honda civic
Honda accord rules
A Honda accord 4 u!

bigram pair using the subsumption rule. So, the first post of Table 1 yields the bigrams: {Honda, civic}, {civic, is}, etc.[2]

Building entity trees based on the Sanderson and Croft heuristic does require a few assumptions. First, by considering the ordered bigrams, we assume that the order in the entity tree is reflected in the posts. That is, users tend to order the attributes from more general (e.g. car makes) to more specific (e.g. car models). Also, this ordering needs to hold in at least enough of the posts to make the subsuption rule fire.

Yet, once our approach builds the entity trees, it uses them effectively for extraction, as shown in our experiments. Therefore, our approach leverages the little ordering it does find in some the bigrams to build the reference set, which we can then use to extract values from posts where the ordering assumption does not hold. Further, given that our approach finds effective entity trees at all reflects the notion that users do tend to order attributes from general to specific. If this were not the case, the discovered entity trees would have little utility for extraction later.

Also, the Sanderson and Croft heuristic above is defined for single tokens x and y, yet not all attribute values are unigrams. Therefore, to handle bigrams, we add the constraint that if x subsumes y and y subsumes x, we merge x and y into a single node (attribute value). For instance, given attribute values "Crown" and "Victoria" if "Crown" SUBSUMES "Victoria" and "Victoria" SUBSUMES "Crown" we merge them into a single value "Crown Victoria" (which is subsumed by the common parent "Ford"). We note this bigram was actually merged using our approach. To extend this to n-grams, one simply checks all token pairs against each other for subsumption. We note, that in our testing, 5.49% of the discovered attributes are bi-grams.

The above assumptions aside, applying the subsumption rule above can also lead to noisy entity trees. A common occurrence in auction listings, for instance, is to include the term "Free Shipping" or "Free Handling." If such phrases occur frequently enough in the posts, the subsumption rule will fire, creating an entity tree rooted on Free with the children Shipping and Handling. Clearly this is a noisy tree and it would introduce noisy extractions.

To deal with this problem we use a small amount of background knowledge to focus the construction of the entity trees. Our approach constrains the construction of entity trees such that each entity tree must be rooted on a seed value. If we only gave "Honda" as a seed, then only one entity tree rooted on Honda would be constructed. Even if other entity trees are discovered, they are discarded. However, it is easy to discover an exhaustive list of seeds on the Web, and including too many seeds is not a issue as our algorithm simply removes any entity tree that consists solely of a root from the constructed reference set (i.e. a singleton set).

One of the key intuitions behind our approach is that the set of root nodes for entity trees are generally much easier to discover online than nodes farther down the trees. For instance, if we consider laptops, the manufacturers of laptops (the roots) are fairly easy to discover and enumerate. However, as one traverses farther down the entity trees, say to the model numbers, it becomes hard to find this information. Yet, just this small sets of seeds is enough to improve the process of reference set construction substantially, as we show in the results where we compare against reference sets constructed without the seed-based constraint. Also, importantly, the attributes farther down the tree change more over time (new model numbers are released often), while the seeds infrequently change (there are few new computer makers). So, coverage becomes less of an issue when only considering the roots versus all of the attributes in a reference set tuple.

Once we have constructed the initial entity trees, we then iterate over the posts to discover possible terms that occur across trees. Specifically, subsumption is determined by the conditional probabilities of the tokens, but when the second token is much more frequent than the first token, the conditional probability will be low and not yield a subsumption. This occurs when the attribute value appears across entity trees (reference set tuples). Since the second term occurs more frequently than the first across tuples, we call this the "general" token effect.

An example of this general token effect can be seen with the trim attribute of cars. For instance, consider the posts in Table 2 which show the general token effect for the trim value of "LE." These posts show the "LE" trim occurring with a Corolla model, a Camry model, a Grand AM model, and a Pathfinder. Since LE happens across many different posts in many varying bigrams, we call it a "general" token, and its conditional probability will never be high enough for subsumption. Thus it is never inserted into an entity tree.

Table 2: Posts with a general trim token: LE

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

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

To make this clear, consider again the 'LE' trim. By it-

[2]We only consider ordered bigrams, rather than all combination of token pairs. We measured an average post length of 8.6 tokens across thousands of posts, which would generate more than 40,320 possible token pairs to check, per post, if all pairs are considered.

erating, the algorithm considers the following conditional probability for subsumption, assuming the models of Camry, Corolla and Pathfinder have already been discovered:

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

Now, if this conditional probability fits the heuristic for subsumption, then LE is added as a child to the nodes CAMRY, COROLLA and PATHFINDER in their own respective entity trees. In this manner we construct a reference set directly from the posts, using the seeds to constrain the noise that is generated in the final reference set.

Table 3 formally describes our method for constructing entity trees using seeds, which are then turned into a reference set by outputting a tuple for each path from root to leaf in each tree.

Table 3: Entity tree construction using seeds

Input: Posts P, Seeds S

```
/* First, break posts into bigrams */
Bigrams B ← MAKEBIGRAMS(P)

/* Build the entity trees rooted on the seeds */
AddedNodes N ← {}
For each {x, y} ∈ B          /* {x, y} is a bigram */
   If x ∈ S                  /* check x is a seed */
      If x SUBSUMES y
         y is child of x in entity tree
         N ← N ∪ y

/*Find all children's children, and their children, etc.*/
While N is not null
   For each {s, t} ∈ B
   where s ∈ N
      N ← N - s
      If s SUBSUMES t
         t is child of s in tree
         N ← N ∪ t

/* Iterate to discover ``general'' token nodes */
/* Start with unique nodes already in the entity trees */
AllEntityNodes ← UNIQUELIST(All Entity Trees)
/* Keep iterating while find new terms */
FoundNewTerms ← true
While FoundNewTerms is true
   FoundNewTerms ← false
   /* Consider terms {p0, ..., pn} in entity the trees
      that all form bigrams with non-entity tree term q */
   For each ( {p0, ..., pn}, q) s.t. {p0, ..., pn} ⊂ AllEntityNodes
      If {p0, ..., pn} SUBSUMES q      /* consider P(⋃ pi|q) */
         q becomes child of each pi in trees
         AllEntityNodes ← AllEntityNodes ∪ q
         FoundNewTerms ← true
```

3 Experiments and Results

The goal is to build reference sets for use in information extraction, so we use information extraction as the task by which to compare the effectiveness of the different methods for building reference sets. We compare our seed-based approach to both full, manually constructed reference sets extracted from online sources (which we call the "manual" approach), and to a version of our algorithm that is exactly the same as outlined above, but that does not constrain the entity trees to be rooted on seed values. The version that ignores the seed-based constraint is a fully automatic method for building the reference set from just the posts themselves, and so we refer to it as "Auto" in our results below.

Then, for each of our experimental data sets, we build three different reference sets (one manual, one based on seeds, and one without the seeds) and then pass the reference sets to a system that can exploit them for extraction [Michelson and Knoblock, 2007]. We then compare the extraction results using the standard metrics of recall, precision, and F_1-measure. Since the only difference for the extraction algorithm is the reference set provided, the extraction results serve as a proxy for the quality of the reference set both in terms of how well it overlaps with the posts (the coverage) and how clean it is (since noise leads to poor extractions).

The goal of our extraction task is to extract the values for given attributes. For instance, using Figure 1, we should extract the make={Honda}, model={Civic}, and trim={DX}. However, our approach to constructing reference sets does not supply these attribute names. Our method discovers attribute values such as "Honda" and "Civic," but it internally labels their associated attribute names as attribute_0 and attribute_1, instead of Make and Model. Therefore, to clarify the results we manually label the attribute names as given by the manually constructed reference sets. We do not feel this is much of a hindrance in our method. If a user can find a set of seeds, the user should also be able to find appropriate attributes names. In fact, it is exceedingly more challenging to discover the attribute values than just finding their names.

We used three real-world data sets as our experimental data. The first set contains classified ads for used cars for sale from the classified site Craigslist.org. Of these, we labeled 1,000 posts to test the extraction of the make (e.g. Honda), model (e.g. Civic), and trim (e.g. DX) attributes. The second set consists of classified ads for used laptops from Craigslist.org as well. Again we labeled 1,000 for extracting the maker (e.g. IBM), model (e.g. Thinkpad), and model number (e.g. T41). Our last data set contains posts about skis for sale on eBay. We labeled 1,000 for extraction of the brand (e.g. Rossignol), the model (e.g. Bandit), and the model specification (e.g. B3, called the "spec"). The data is summarized in Table 4.

Table 4: Three experimental data sets

Name	Source	Attributes	Num. Posts
Cars	Craigslist	make, model, trim	2,568
Laptops	Craigslist	maker, model, model num.	2,921
Skis	eBay	brand, model, model spec.	4,981

We need full, manually constructed reference sets for comparison. For the Cars domain, we collected 27,000 car tuples by pulling data from the Edmunds.com car buying site for cars and combining it with data from a classic car site, SuperLambAuto.com. For the Laptops domain, we scraped 279 laptops off of the online retailer Overstock.com. Lastly, for the Skis domain, we built 213 ski tuples from the skis.com website, and cleaned them to remove certain stop-words.

The seeds for our seed-based method also came from freely available sources. For the car domain, the seeds consist of 102 car makes, again from Edmunds. The laptop seeds are 40 makers, culled from Wikipedia, and the ski seeds are 18 ski brands pulled from Skis.com.

Table 5 shows the field-level extraction results for each attribute in each domain, comparing the three methods.[3] Again, the "manual" method uses the full reference set constructed from online sources (shown in parentheses in the table), the "auto" method is the fully automatic method without seeds, and our full technique is called "seed-based."

Table 5: Extraction results for different reference sets

Cars			
Make	Recall	Prec.	F_1-Meas.
Manual (Edmunds)	92.51	99.52	95.68
Auto	79.31	84.30	81.73
Seed-based	89.15	99.50	94.04
Model	Recall	Prec.	F_1-Meas.
Manual (Edmunds)	79.50	91.86	85.23
Auto	64.77	84.62	73.38
Seed-based	73.50	93.08	82.14
Trim	Recall	Prec.	F_1-Meas.
Manual (Edmunds)	38.01	63.69	47.61
Auto	23.45	54.10	32.71
Seed-based	31.08	50.59	38.50
Laptops			
Maker	Recall	Prec.	F_1-Meas.
Manual (Overstock)	84.41	95.59	89.65
Auto	51.27	46.22	48.61
Seed-based	73.01	95.12	82.61
Model	Recall	Prec.	F_1-Meas.
Manual (Overstock)	43.19	80.88	56.31
Auto	54.47	49.52	51.87
Seed-based	70.42	77.34	73.72
Model Num.	Recall	Prec.	F_1-Meas.
Manual (Overstock)	6.05	78.79	11.23
Auto	25.58	77.46	38.46
Seed-based	34.42	86.05	49.17
Skis			
Brand	Recall	Prec.	F_1-Meas.
Manual (Skis.com)	83.62	87.05	85.30
Auto	60.59	55.03	57.68
Seed-based	80.30	96.02	87.46
Model	Recall	Prec.	F_1-Meas.
Manual (Skis.com)	28.12	67.95	39.77
Auto	51.86	51.25	51.55
Seed-based	62.07	78.79	69.44
Model Spec.	Recall	Prec.	F_1-Meas.
Manual (Skis.com)	18.28	59.44	27.96
Auto	42.37	63.55	50.84
Seed-based	50.97	64.93	57.11

Table 6 summarizes the results, showing the number of attributes where our seed-based method outperformed the other techniques in terms of F_1-measure. It also shows the number of attributes where the seed-based technique is within 5% of the other method's F_1-measure (including the attributes where the seed-based method outperforms the other method). An F_1-measure within 5% is a "competitive" result.

Table 6: Summary results comparing our method to others

	Seed vs. Auto	Seed vs. Manual
Outperforms	9/9	5/9
Within 5%	9/9	7/9

The results show that our seed-based method builds a cleaner reference set than the fully automatic approach that ignores the seeds since the seed-based approach outperforms the "auto" approach on every single attribute. The seed-based method builds a cleaner, more effective reference set and that leads to more effective extraction.

The results also show that our seed-based method retains the benefits of constructing reference sets from the posts themselves, as compared to a full, manually constructed reference set. In particular, the results support the idea that using the posts themselves generates a reference set with better coverage than the single sources that were scraped manually. Not only does the seed-based method outperform the manual reference sets on a majority of attributes (5/9), the seed-based method does much better at building a reference set to represent the most specific attributes (ski model, ski model spec., laptop model, and laptop model num.), which are those attributes that are likely to cause coverage problems. In fact, for these attributes, only 53.15% of the unique attribute values in the seed-based reference set exist in the manually constructed reference set. Therefore, the coverage is quite different.

For example, it is important to note that Overstock sells *new* computers, while the laptops for sale on Craigslist are generally *used, older* laptops. So, while there is a match between the makers (since the laptop makers don't change quickly), even if the used laptops are six months older than the new ones for sale there will be a mismatch between some models and for many of the model numbers. This coverage mismatch using the "manual" reference sets is very clear for the laptop model numbers and ski model specifications. Both of these are attributes that change quite frequently over time as new models come out. This is in contrast to ski brands and laptop manufacturers (our seeds) which change much less frequently, and so can be enumerated with less of a concern towards coverage. We note that we chose Wikipedia because of its comprehensive list of laptop makers, but Wikipedia enumerates far fewer models and model numbers than the Overstock reference set, and so would be a worse choice for a manual reference set.

Also, we note that our seed-based technique is competitive on 7/9 attributes when compared to the full, manually constructed reference sets. Yet, the number of seeds is drastically smaller than the number of tuples manually constructed for those reference sets. So, even though we are starting with a much tinier set of knowledge, we still retain much of the benefit of that knowledge by leveraging it, rather than having to explicitly enumerate all of the tuple attributes ourselves. This is important as it is much easier to find just the seeds.

The one attribute where the manual reference set really out-

[3]Field-level results are strict in that an extraction is correct only if all the tokens that should be labeled are, and no extra tokens are labeled.

performs our seed-based method is the trim attribute for cars, where the difference is roughly 9% in F_1-measure. This is mostly due to the fact that we use field level results, and when the seed-based technique constructs the trim attribute it sometimes leaves out certain attribute tokens. For example, consider the example where the extracted trim should be "4 Dr DX." However, the seed-based technique only includes "DX" as the reference set tuple's attribute. Meanwhile, the manually constructed reference set contains all possible tokens since it is scraped from a comprehensive source (it's attribute value is "4 Dr DX 4WD"). So, although our seed-based technique finds the DX token and labels it correctly as a trim, it misses the "4 Dr" part of the extraction, so the whole extraction is counted as incorrect using field level results.

One limitation of our seed-based technique versus the manual construction of reference sets has to do with the inclusion of certain attributes. Surprisingly, there is not enough repetition in the posts for discovering the years of the cars as attributes. This is due to various factors including the variety of year representations (all four digits, two digits, strange spellings, etc.) and the placement in the posts of the years (since we consider bigrams for subsumptions). However, the full manual reference set does contain years and as such it can extract this attribute, while the seed-based method cannot. Nonetheless, although a manual reference set may include an attribute that can not be discovered automatically, the reference set might have terrible coverage with the posts, limiting its utility. So, we feel it is important to deal with coverage, as our seed-based method does.

We also tested the effect of iterating to capture "general" tokens versus simply stopping after the first pass. Iterating results in an improvement in F_1-measure for all of the attributes except for one. This improvement is especially dramatic for the most specific attributes, the Car trims and Ski specification, which are attributes that occur across separate entity trees. The attribute where we note a decrease is the Laptop model numbers. In the laptop domain many discovered "general" tokens are placed as model numbers when they are actually different attributes such as cpu speeds. This results in reference set tuples that share a brand and model, but differ in that one has the correct model number while the other assigns the cpu as the model number. Then, during extraction, both of these tuples match a given post (e.g. the post has both the correct model number and the cpu speed in the text), resulting in erroneous extractions. Nonetheless, the dramatic increase for the other attributes warrants the inclusion of iterating, and these errors are more an artifact of the extraction algorithm than the reference set.

Although extraction experiments serve as the best metric for the actual utility of the seed-based reference sets, we also ran experiments on the generated entity trees themselves. We examined whether attribute values are consistent in their placement in the entity trees (the column homogeny). For instance, given known car models such as "Civic" we measure if the model values are mostly placed as car model attributes (second level in the tree) or misplaced as car trims (third level). However, measuring this directly without domain expertise is difficult. Instead, we compare the attribute values in the seed-based reference set to those in the manually constructed reference sets, and for those values that match, we measure if they are the for same attribute (i.e. their columns match) or not. This yields a measure of the column homogeny for the seed-based reference set, based on the manual reference set, which is assumed to be clean. However, it is an approximate measure because not all values in the seed-based reference set match those in the manual reference set since they differ in coverage.

Nonetheless, the results of this approximate measurement indicate a good level of homogeny amongst the seed-based attributes. For skis, only 1.7% of the found attribute values are in the wrong column, while for cars 2.9% of the values are in the wrong columns. Both the skis and cars had a common error of placing the specific attribute (model spec or car trim) one spot higher in the entity tree than it should have been. However, the approximation was misleadning for laptops. In the laptops domain, we had perfect column homogeny using this measure, but this is because we can only measure the column homogeny for attributes that match in both the seed-based and manual reference sets. Yet, there were obvious column homogeny errors, such as cpu speeds being placed as model numbers. Since these did not match into the manual reference set, they were ignored by our homogeny measuring experiment. Given that we have enough domain expertise, we did a manual calculation for this set, and determined that 8.09% of the tuples in the seed-based set have a cpu speed or other variant as a model number which is incorrect. However, even at 8% this is a good result for homogeny.

Lastly, we compare information extraction using our seed-based reference sets to a supervised machine learning approach to extraction: Conditional Random Fields (CRF) [Lafferty *et al.*, 2001]. We used MALLET [McCallum, 2002] to implement two different CRF extractors. One, called "CRF-Orth," uses orthographic features of the tokens for extraction, such as capitalization, number containment, etc. The second extractor, "CRF-Win," uses the same orthographic features, and also considers a two-word sliding window around a token as a feature. We perform 10-fold cross validation for each extractor (using 10% of the data for training, to reflect real-world human-cost requirements for labeling) noting that each fold is independent, and we use the average results for comparison. For brevity, Table 7 shows the summary extraction results, similar in format to those of Table 6.

Table 7: Summary results comparing the seed-based method to CRFs

	Seed vs. CRF-Win	Seed vs. CRF-Orth
Outperforms	7/9	6/9
Within 5%	9/9	7/9

Table 7 shows that our seed-based method outperforms the two CRF extractors on a majority of the attributes, even though the cost in creating a seed list is much less than the cost of labeling the data and creating features for training our specific CRFs. Further, the table shows that a technique that relies heavily on structure, such as CRF-Win, performs worse on extraction from posts as compared to other methods.

4 Related Work

Our goal of creating reference sets for information extraction is somewhat similar to research on unsupervised information extraction from the Web [Cafarella *et al.*, 2005; Hassan *et al.*, 2006; Paşca *et al.*, 2006]. This work focuses on finding relations, such as person X was born in country Y by looking at webpages, which is similar to building a reference set. However, this research learns patterns to make extractions. Such patterns assume that similar structures will occur again to make the learned extraction patterns useful. We can not make such structural assumptions about posts beyond the redundancy of bigrams.

Building our entity trees using the Sanderson and Croft (1999) heuristic relates our approach to work in ontology creation [Cimiano *et al.*, 2005; Dupret and Piwowarski, 2006; Makrehchi and Kamel, 2007]. Our approach follows the current research on this topic in that we generate candidate terms (the bigrams in our case) and then plug these terms into an hierarchy (our entity trees). However, there are a number of differences between the other work on ontology creation and our approach to building entity trees. First, many of those methods use NLP to determine the terms, which we cannot do with posts. Next, these other approaches build up single, large concept hierarchies. In our case, we build multiple independent entity trees, which are then flattened into a reference set. So, we use hierarchy building in order to construct a set of relational tuples, but our attributes are not necessarily conceptually hierarchical. Our method is more similar to filling out a taxonomy, rather than building a conceptual ontology. Further, all of these methods use webpages as resources for discovery, while we use posts as our base data.

5 Conclusion

This paper presented a method for building a reference set from unstructured, ungrammatical data using a small amount of background knowledge, called a "seed set." Our method combines the utility of starting knowledge, which generates cleaner and more effective reference sets, with the ability to construct the rest of the reference set from the posts themselves, which ensures an alignment between the data to extract from, and the reference set used. Further, using only seeds is often a more viable choice than creating a full reference set manually, because of the difficultly in finding a single source that provides all of the required data (e.g., finding a single source of all laptop model numbers) and given the fact that certain attributes change over time. The seeds are high level information, however, and are easier to find and they change much less frequently.

There are a number of future directions for this work. First, we intend to combine our approach with more background knowledge, such as nonroot lists from services such as Google sets or even partial, manually constructed reference sets. Also, although this work shows that users do sometimes consider certain attributes as more general than others, we would like to investigate this more formally and examine what happens with attributes that should be siblings in an entity tree (such as model numbers and cpu speeds).

References

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

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

[Ciravegna, 2001] Fabio Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *Proceedings of IJCAI*, pages 1251–1256, 2001.

[Dupret and Piwowarski, 2006] Georges Dupret and Benjamin Piwowarski. Principal components for automatic term hierarchy building. *String Processing and Information Retrieval*, pages 37–48, 2006.

[Hassan *et al.*, 2006] Hany Hassan, Ahmed Hassan, and Ossama Emam. Unsupervised information extraction approach using graph mutual reinforcement. In *Proceedings of EMNLP*, pages 501–508, 2006.

[Kushmerick *et al.*, 1997] Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *Proceedings of IJCAI*, pages 729–737, 1997.

[Lafferty *et al.*, 2001] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML*, pages 282–289, 2001.

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

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

[Michelson and Knoblock, 2007] Matthew Michelson and Craig A. Knoblock. 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*, 10:211–226, 2007.

[Michelson and Knoblock, 2008] Matthew Michelson and Craig A. Knoblock. Creating relational data from unstructured and ungrammatical data sources. *Journal of Artificial Intelligence Research (JAIR)*, 31:543–590, 2008.

[Paşca *et al.*, 2006] Marius Paşca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain. Organizing and searching the world wide web of facts - step one: the one-million fact extraction challenge. In *Proceedings of AAAI*, pages 1400–1405, 2006.

[Sanderson and Croft, 1999] Mark Sanderson and Bruce Croft. Deriving concept hierarchies from text. In *Proceedings of ACM SIGIR*, 1999.