

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

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 marked up 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 or internet classifieds like Craig's list, where we would like to add markup within these strings of text. It would be beneficial to semantically annotate such text so that it is searchable for agents, however, a user just wants to type a line or two, without the burden of adding the extra markup. The goal, then, should be to keep the web as familiar to the user as possible, while making the data as useful as possible behind the scenes.

As a solution we propose that an information extraction (IE) system work behind the scenes to parse the entered text into attributes that could then be automatically put into a format for the Semantic Web. For example, assume a user enters the following headline, "1988 Honda Accord for sale! Only 80k miles, Runs Like New, V6, 2WD...\$2,500 obo. SUPER DEAL." Once the user hits the submit button, the IE system would take over, parsing attributes such as make and price from the entered text, and then annotating them with the correct XML accordingly. The user did no extra work, and yet a used car agent now has one more searchable item to explore and consider. Table 1 shows the pieces we might extract from the post.

However, information extraction on this type of text is especially difficult because the data is neither structured enough to use wrapper technologies, such as Stalker [13] or Road-Runner [7] nor grammatical enough to exploit Natural Language Processing (NLP) techniques such as those used in Whisk [15] or Rapier [4]. However, by leveraging reference sets, these difficulties can be overcome.

A reference set is a collection of known entities and their common attributes, which can exist online as a set of reference documents, such as the CIA World Fact Book, or can exist as an online (or offline) database, such as the Comics Price Guide. In the case of the Semantic Web, numerous reference sets already exist as ontologies. Continuing with our used car example, there is an ontology of cars. It would have all of the makes and models, along with the range of years

they were produced, and the special characteristics of each model, such as engine and drive type.

We could then exploit this reference set to help determine what, if any, of these attributes appear in the user entered text. This would break down into a two step process. First, we would need to figure out which member of the reference set best relates to the entered text. We call this the alignment, or Record Linkage (RL) step. The second step, the Extraction step, would parse (or tag) the attributes from the entered text that match those from the matching member of the reference set. In our example, the entered text would match the Honda Accord member of the car ontology. Then, for instance, when the system saw Accord in the entered text, it could label it as the car model, since it is the car model attribute of the matching reference set record. In this manner we could annotate all of the pertinent information in the text entered by the user. Note that throughout this paper, to emphasize the point, we use the phrase "reference set" and ontology interchangeably.

This paper describes just such an algorithm for alignment and extraction based on this alignment. Sections 2 and 3 describe the two steps of the algorithm. Section 4 presents experimental results, and section 5 discusses related research. We end with a discussion in section 6.

Table 1: Extraction on posts

Post				
1988 Honda Accord for sale! Only 80k miles, Runs Like New, V6, 2WD...\$2,500 obo. SUPER DEAL				
Make	Model	Year	Price	mileage
Honda	Accord	1988	\$2,500	80k

2 Record Linkage

To correctly parse out attributes from the entered text, we need to first decide just what those attributes are. To aid in this process, we first align the entered text to a member of the reference set we have chosen to use. This alignment will provide us with the attributes that we should look for in the entered text for parsing.

It is infeasible to compare the text against all members of the reference set, so first we construct a set of candidate

matches from the reference set. This “blocking” step is simple and fast. Many methods have been proposed in the RL community, see [2] for a recent survey of some. The choice of algorithm here is really independent of the overall alignment algorithm.

Once a set of candidates is generated, we must find the best match amongst these candidates to the entered text. However, since the entered text is not already parsed into attributes, and since this text has many irrelevant tokens (from a matching perspective), we cannot rely on traditional record linkage techniques, such as those employed by Marlin [3].

Instead, we create a vector of scores for each candidate consisting of the similarity between the post and each attribute of the reference set, and also the similarity between the post and all of the attributes concatenated together. So, assuming our example ontology had 2 attributes, {make,model}, and the current cluster member was {“Toyota”,“Camry”}, our vector would look like:

$$V = \{ scores(text, \text{“Toyota”}), scores(text, \text{“Camry”}), scores(text, \text{“Toyota Camry”}) \}$$

Where text = “1988 Honda Accord for sale! Only 80k miles, Runs Like New, V6, 2WD \$2,500 obo. SUPER DEAL.”

Each $scores(text, \text{“attribute”})$ part of this vector is the whole set of similarity scores for this attribute (or all of them in the case of the concatenation.) These scores break down into 3 categories.

Some of these scores are token based similarity scores, such as Jensen-Shannon distance, which reflect the token level similarity. However, there might be an important similar token in the text that is misspelled. The token based metrics would not reflect this, and so we also include our second category of similarity scores, the edit based distance scores, such as Smith-Waterman similarity. This way, if Accord is misspelled in the post as Accd, we can still capture this similarity. Lastly, we include a category of similarity functions that are neither token nor edit based, such as a soundex score, which captures similarities not captured by the other two categories. So, our vector really looks something more like:

$$V = \{ Jensen-Shannon(text, \text{“Toyota”}), \dots, Smith-Waterman(text, \text{“Toyota”}), \dots, Soundex(text, \text{“Toyota”}), \dots, Jensen-Shannon(text, \text{“Camry”}), \dots \}$$

The idea here is that by using just attributes themselves, we get a notion of the field similarity between the entered text and the match candidate, while the concatenation of the attributes gives an idea of the record level match. Since our post is not yet broken into attributes, this is the only way in which we can try to determine these field and record level similarities. However, this begs the question, “if all we care about is a record level match, why not use the concatenation only?” The answer is that it is possible for different records in the ontology 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.

Once all of the candidates are scored in this manner, they are rescored by binary normalization. In this step, for each index of the vector of scores, the candidates with the maximum value at that index are rescored to 1, while all of the

other candidates receive a 0 at that index. This mirrors the idea that although there may be a few candidates that are very close to each other, in that they have similarly close values for the indexes, only one of them is a best match, so it should be separated out as much as possible.

For example, if we had:

$$V1 = \{.999, 1.2, \dots, 0.45, 0.22\}$$

$$V2 = \{.888, 0, \dots, 0.65, 0.22\}$$

After binary rescaling we would get:

$$V1 = \{1, 1, \dots, 0, 1\}$$

$$V2 = \{0, 0, \dots, 0, 1\}$$

After the rescaling is done, the candidates are passed to a Support Vector Machine (SVM) [10] which will label them matches or not. These binary values of the vector are the features that the SVM use to learn the classification.

3 Extraction

Once a member of the reference set is chosen as the best match, we can use the attributes of this reference set record as a basis upon which to do our extraction. This is the key idea of the whole algorithm, namely that we can do the extraction because we benefit from knowing the attributes of the reference set.

However, it might seem that we are done after the alignment step. We could simply use the values from the matching ontology record, and the extraction step is needless. Yet, even if the correct match was not found, it is still possible to use this reference information to correctly parse as much valuable data as possible from the text. For example, what if our post said, “1988 Honda Acd for sale!...” This might lead the system to pick an incorrect ontology reference member, like Honda Civic, simply because it could not pick up on the Acd as Accord, but could find Honda. Yet, even if this happens, there is enough useful information in the incorrect ontology member that many of the useful pieces of the post could still be parsed out correctly. We would assume, for instance, that surely “Honda” would be labeled as the car make.

To begin the extraction process, the entered text is broken up into tokens, based on spaces. In our example, the entered text becomes the set of tokens, {“1988”, “Honda”, “Accord”, ...}. Each of these tokens is then scored against each attribute of the record from the ontology that was deemed the matching reference record.

Again, assuming only two attributes in the ontology, let’s assume the match found was the “Honda”, “Accord” record. We build a vector of scores between each token and all of the reference set attributes. These scores can be edit based scores, like in the alignment step, or not (again, such as Soundex), but the token based scores are removed, because we compare against single tokens. To further improve the system, you could also include scores not related to the reference set, such as token position in the entered text, the ratio of numbers to letters in the token, etc. We call these common scores. Common scores are especially useful for disambiguating similar attributes (such as street name “6th” vs. street num “612”) and/or parsing out attributes that are not easily represented in a reference set (such as a date or price).

We call the vector of all of the scores a Feature Profile, which looks like:

$$FP = \{ CommonScores(token), Score(token, attribute1), Score(token, attribute2), \dots \}$$

Since each FP is not a member of a cluster where the winner takes all, there is no binary rescaling. These FPs are then passed to a multiclass SVM [17] trained to give each FP a class label, where each class is an attribute type from the reference set, such as car make or year. The SVM can also label an FP as junk, meaning it does not match any attribute type.

Intuitively, similar attribute types should have similar FPs. We expect that car makes will generally have high scores against the reference set attribute of car makes, and small scores against the other attributes, and the vector will reflect this.

The SVM, then, 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 can not be made when users are entering text.

Once the extraction has finished, we replace the class labels (ignoring the junk) with semantic annotation and the Semantic Web has one more searchable entity.

4 Results

To verify our approach we tested the technique in two domains, comic books and hotel postings.

The comic books domain used posts from EBay about items for sale searched by keyword “Incredible Hulk” and “Fantastic Four”. Our goal was 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 (<http://www.comicspriceguide.com/>) for lists of all of the Incredible Hulk and Fantastic Four comics, as well as a list of all possible comic book conditions.

In the hotel domain, we attempted to parse the hotel name, hotel area, star rating of the hotel, price and dates booked from the Bidding For Travel website (www.biddingfortravel.com), which serves as a forum where users can share successful bids for Priceline on different items. 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, which are special posts within each city’s list of posts that list the hotel name, hotel area and the star rating within each city.

As one baseline comparison, we present our results versus those obtained by using the Simple Tagger tool of the MALLET [12] suite of text processing tools. The Simple Tagger tool is an implementation of Conditional Random Fields that

can be effectively used in segmentation/labeling tasks.

We also present our results versus the Amilcare system [6], 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 [6]. It thus presents a good benchmark versus other NLP based systems, which we expect will not do well on our data type. For the tests, we supplied our reference data as gazetteers to Amilcare.

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 of each technique, in this manner, and report the average results over the 10 trials.

We present our results using Precision, Recall and F-Measure as they are defined for information extraction tasks [14]. The table 2 shows the results of correctly labeling the tokens within the posts with the correct attribute label.

Table 2: Baseline Comparison

	Prec.	Recall	F-measure
Hotel			
Using ref. set	94.41	94.25	94.33
Simple Tagger	89.12	87.80	89.00
Amilcare	86.66	86.20	86.39
Comic			
Using ref. set	96.19	92.5	94.19
Simple Tagger	84.54	86.33	85.42
Amilcare	87.62	81.15	84.23

5 Related Work

This work is along the idea of Hendler [9] that the cost of marking up documents for the Semantic Web should be free, that is, automatic and invisible to users. Many researchers have followed this path, attempting to automatically mark up documents for the Semantic Web, as we propose here.

Cimiano et al. [5] perform semantic markup by identifying proper nouns in the text of Web pages, and then querying reference sets (in this case, Google), to discover hypothetical category labels for tagging. However, relying on lexical information, such as POS tagging, is not an assumption we can use with our data type since grammatical clues such as capitalization and word order are not reliable.

Along the lines of systems that use lexical information are the MnM system [18] and the S-CREAM system [8], that both rely on the Amilcare system internally for extraction. However, Amilcare exploits shallow NLP to aid in extraction, which is difficult to do on our data type. (Refer to section 4 for a comparison to Amilcare.)

The work of Dingli et al. [1] does unsupervised, automatic semantic annotation by querying reference sets with items (such as names to Citeseer) and then using the returned results to aid in tagging those items. However, it is not clear how such a system could deal with the myriad of misspellings of similar words as happens with user entered text, since this

could harm the query step. Note that this system also uses Amilcare for extraction.

One source of our data is lists of items, such as Ebay posts, which is comparable to processing records for entities and descriptors of these records. In this sense, our work is most similar in this area to the ADEL system [11], which marks up sets of records in web pages. Our work differs in that the records annotated by ADEL follow a similar structure, which is exploited for extracting the attributes, while our system does not use structural information because the data is not structured enough.

While there is a fair amount of work in the field of automatic annotation, there is not much emphasis on adding this annotation to text that is neither structured nor grammatical. Hence, our main contribution is a technique that addresses this specific type of textual data.

6 Conclusion

In this manner we have shown how to create more content for the Semantic Web from a type of textual data that would be difficult to do so traditionally, but is prevalent on the web. We have shown that this can be done without any extra effort on the user's behalf, and yet there would be large benefits for agents searching the Semantic Web that included such data.

As a future direction, we would like to link this technique with a mediator framework [16] for querying the Semantic Web. With this integration, a designer would not need to supply the reference sets, but rather, the mediator could query and discover them itself. How to automatically formulate a query to retrieve the correct domain ontology is a direction of future research.

References

- [1] Y. W. A. Dingli, F. Ciravegna. Automatic semantic annotation using unsupervised information extraction and integration. In *K-CAP 2003 Workshop on Knowledge Markup and Semantic Annotation*, 2003.
- [2] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.
- [3] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *the Proceedings of the 9th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD-2003)*, pages 39–48, Washington, DC, August 2003. ACM Press.
- [4] M. S. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, pages 328–334, Orlando, Florida, August 1999.
- [5] P. Cimiano, S. Handschuh, and S. Staab. Towards the self-annotating web. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 462–471. ACM Press, 2004.
- [6] F. Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, 2001.
- [7] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.
- [8] S. Handschuh, S. Staab, and F. Ciravegna. S-cream - semi-automatic creation of metadata. In *the 13th International Conference on Knowledge Engineering and Knowledge Management, EKAW02*. Springer Verlag, 2002.
- [9] J. Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.
- [10] T. Joachims. Learning to classify text using support vector machines, 2002. Kluwer.
- [11] K. Lerman, C. Gazen, S. Minton, and C. A. Knoblock. Populating the semantic web. In *Proceedings of the Workshop on Advances in Text Extraction and Mining (ATEM-2004)*, 2004.
- [12] A. K. McCallum. Mallet: A machine learning for language toolkit. <http://mallet.cs.umass.edu>, 2002.
- [13] I. Muslea, S. Minton, and C. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.
- [14] F. Peng and A. McCallum. Accurate information extraction from research papers using conditional random fields. In *Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL, 2004)*.
- [15] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.
- [16] S. Thakkar, J. L. Ambite, and C. A. Knoblock. A data integration approach to automatically composing and optimizing web services. In *Proceedings of 2004 ICAPS Workshop on Planning and Scheduling for Web and Grid Services*, Whistler, BC, Canada, 2004.
- [17] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. 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.
- [18] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. Mnm: Ontology driven semi-automatic and automatic support for semantic markup. In *The 13th International Conference on Knowledge Engineering and Management (EKAW 2002)*, 2002.