An Exploratory Study Towards "Machines that Learn to Read"

Elizabeth Boschee, Vasin Punyakanok, Ralph Weischedel

BBN Technologies
10 Moulton Street, Cambridge, MA 02138, USA
eboschee@bbn.com, vpunyaka@bbn.com, weischedel@bbn.com

Abstract

This paper reports early results at the intersection of knowledge and language acquisition. Humans learn much by reading, a capability largely absent from machines. We assume that (1) some conceptual structure exists, represented in an ontology, (2) a handful of examples of each concept and relation is provided, and (3) the machine knows the grammatical structure and semantic structure of the language. The task is to learn the many ways that the concepts and relations are expressed so that a machine can automatically map from source text to the knowledge base. As a case study we looked at the relations invent(inventor, invention), employ(employer, employee), and located-at(entity, location). Our results show that structural features, e.g., dependency parses and propositions (predicate argument structure), outperform non-structural features, e.g., strings of words.

Introduction

One key element of human cognition is the ability to learn by reading. The time-honored approach to machine reading is to handcraft rules to map text to a knowledge base, for example dating back to experiments by Ross Quillian (1968). Our approach is to "learn to read", i.e., to employ learning algorithms, e.g., algorithms that learn to

1. parse text from supervised training data (several well known techniques in the state of the art),
2. resolve coreference from supervised training (Pradhan et al., 2007),
3. recognize the terminology for concepts and relations from a handful of instances (this paper).

In this paper, we report our initial results on task 3. Given a set of concepts, relations, and a few examples of each, can a system expand the set of examples such that it can automatically map from text to those concepts and relations?

Suppose the system is to learn the notion of X inventing Y, by giving the system three example pairs of inventor and invention, namely

- (Thomas Edison, the light bulb)
- (Alexander Graham Bell, the telephone)
- (Ben Franklin, the lightning rod).

The system then iteratively scours a collection of online documents to find sentences where those three relation instances are stated, induces patterns over textual features found in those instances, and iteratively performs cycles of finding additional pairs of (inventor, invention), finding text sentences, and inducing more rules for recognizing the relation of X inventing Y. We have explored three kinds of features:

- Surface structure patterns:
  - X *patented* Y
- Proposition (predicate-argument) patterns:
  - *developed*(subject:X, object:Y)
- Dependency tree patterns:
  - X → *invented* ← Y

In addition to describing the algorithm, we present an evaluation of the performance of the technique. We find it interesting that both the predicate-argument features and the dependency tree features outperform surface structure patterns, which were used in the context of factoid question answering (Ravichandran and Hovy 2002). We also include qualitative results on the importance of selecting the seed examples so as not to lead the learning algorithm astray and the challenge of relations that have few clear lexical signals, e.g, X located at Y.

Though not a model of a human learning to read, it seems likely that a person could use multiple online texts and/or the web to learn to add to a knowledge base in the way the BBN system does. Prior to the web and search tools, it seems unlikely that humans would have developed such a strategy to mine masses of text such as assumed in our approach, except for special cases, e.g., empirically based lexicography.

Objective

Our goal is to create a set of rules (i.e. patterns) that for any span of text can indicate whether or not a concept is present, and for any textual connection between two elements can determine whether a relation holds. We begin with only a few seed examples for any concept or relation.

Technical Approach

The system will "learn to read" a new relation or concept by iteratively mining the Web. The building blocks of this process are rules and examples.

As discussed above, we have explored three kinds of rules in this pilot study: surface structure patterns, dependency tree patterns, and proposition (predicate-argument) patterns. The surface patterns used in this pilot study are strings, e.g. "X *is employed by* Y". The dependency trees and predicate-argument propositions are both generated for this study by automatically transforming syntactic parse trees, which are themselves generated by a trained statistical parser. The propositions also incorporate additional automatically-generated information such as trace resolution.

A proposition consists of a predicate (typically a noun, verb, or adjective) and its associated arguments with their syntactic roles. For instance, the phrase "*The president of France decided to leave*" would generate the following propositions:

- P_1 = *president* (ref:n_1, of:n_2)
- P_2 = *France* (ref:n_2)
- P_3 = *decided* (subject:n_1, object:P_4)
- P_4 = *leave* (subject:n_1)

The elements n_i refer to noun phrases in the sentence; note that n_1 serves as the subject for both P_3 and P_4 due to the trace resolution incorporated in the proposition extraction process.

Examples are simply instances of the target relation or concept. For instance, if the target is the BODYPART concept, examples might be *arm* or *foot*. If the target is the CEO relation, examples might be (*Steve Jobs, Apple*) or (*Steve Ballmer, Microsoft*).

For each iteration of the learning process, we begin with a set of examples the system believes to be instances of the target relation or concept. (For the first iteration, these examples are the seed set.) We then generate a set of rules given these examples, and use these rules to identify new high-confidence examples of the target. These examples are then used to feed the next iteration.

The two major components of each iteration are rule generation and example identification. We describe these in detail below, assuming the target is a relation; the process is analogous for learning a concept.

Rule Generation

For each iteration, a set of rules *R* is generated using examples the machine considers to be likely instances of the target relation.

For each such example pair (x_i, y_i), we retrieve a large set of web documents containing both x_i and y_i. We then extract the sentences where x_i and y_i both occur. We use the BBN software engine Serif to generate a semantic and syntactic analysis of each sentence. This analysis includes named entity identification, parse structure, entity co-reference, and other information. We use this to represent the connection between x_i and y_i in multiple ways; as discussed above, for this pilot study we represent it as a surface string, as a predicate-argument proposition (where possible), and as a dependency tree. From these representations we generalize to a set of hypothesized rules.

For instance, assume we have an example (x_i, y_i) = (*Steve Jobs, Apple*). Our web search might retrieve the following sentence: "*Steve Jobs led Apple to 31% of the education market*." From this sentence, we would extract the following representations of the connection between *Steve Jobs* and *Apple*:

- surface string: X *led* Y
- proposition: *led* (subject:X, object:Y)
- dependency tree: X → *led* ← Y

In each case the noun phrase associated with the first element of the pair is replaced by X, and likewise the second is replaced by Y.

The set of all such representations, derived from all sentences for all examples, become our candidates for inclusion in the rule set *R*. The next step is to prune this set of hypothesized rules, keeping only those that are most likely to be reliable given the evidence available. Pruning for this pilot study was done according to a set of simple heuristics.

First, we discard rules that do not contain an adjective, noun, or non-copula verb. This eliminates very general rules like "X, Y" and "X *in* Y", which are typically too general to be safely applied during an iterative process.

Second, we discard rules generated by fewer than α examples. Patterns that were found in multiple, independent contexts are the most likely to be reliable indicators of the relation. For instance, the surface pattern "Y *CEO* X" would certainly have been generated by both (*Steve Jobs, Apple*) and (*Steve Ballmer, Microsoft*), even in a small experiment. On the other hand, the surface pattern "X*, born in 1955, runs* Y" could only have been generated by the (*Steve Jobs, Apple*) example. The multiply-attested rules will be more reliable; the precision of the rule set should increase with the value of α.

Similarly, we discard rules that were seen fewer than β times, regardless of the examples with which the patterns were associated. To best feed the iterative process, we want our rules to be at least somewhat frequent, assuming they are judged sufficiently reliable according to the primary pruning criteria.

Note that the values of α and β should probably increase for each iteration. For instance, when there are only three examples being used, we may deem rules generated by two of them to be worth keeping, but when there are forty examples, we would likely consider a setting of α=2 far too lenient. One obvious solution is to make the values of α and β a function of the size of the example set.

Example Identification

After a set of rules *R* is generated for a particular iteration, a new set of examples *E* must be identified from the Web.

For each rule, we begin by retrieving a large set of web documents that are likely to contain sentences where this rule will apply. (The current web search looks simply for documents that contain the same lexemes as the target rule; a more sophisticated retrieval strategy would increase the yield here.) All of the rules in *R* are then applied to these documents, and all possible examples of the relation are extracted.

For instance, assume the following rule set *R* (for simplicity we will use only proposition rules here):

1. *CEO* (ref:X, premodifier:Y)
2. *company* (ref:Y, possessive:X)
3. *led* (subject:X, object:Y)

These rules might retrieve the following set of pairs (the rules that generated them are indicated in parentheses):

- *Henry Golub, American Express* (1)
- *Jeffrey Skilling, Enron* (1, 2, 3)
- *Richard Parsons, Time Warner* (1, 3)
- *Bono, U2* (3)

We then apply several pruning heuristics to the set of extracted pairs.

First, we eliminate examples that were generated only by surface structure rules. This heuristic was implemented after empirical evaluation indicated that these were typically less reliable.

Second, we eliminate all examples that were generated by fewer than γ distinct rules.[1] As in the rule generation step, examples that are supported by multiple, independent pieces of evidence are going to be more reliable. It is easy to come up with a counter-example for almost any individual rule, i.e. an example where the rule might generate a pair that does not fit the relation. For instance, we see such an example in the list above: (*Bono, U2*) is not an example of the CEO relation, despite having been selected by rule #3 (from the sentence "*Bono led U2 through two of their biggest hits*"). It is more difficult, however, to come up with counter-examples that could be generated by several distinct rules independently. It is unlikely, for instance, that the Internet contains any references to "*U2 CEO Bono*" or "*Bono's company, U2*".

Obviously aggressive pruning will remove many examples that are actually correct. For instance, in the list above, γ=2 will eliminate (*Henry Golub, American Express*), which is actually a valid example of the CEO relation. However, the iterative process is best served by keeping precision as high as possible and only gradually expanding the example set as more evidence becomes available.

Test Case: INVENT

The primary experiment undertaken for this pilot study was the learning of the relation INVENT. We ran two full iterations of the process, with manual evaluation at both stages. The process therefore had four steps:

1. Manually create a set of three seed examples
2. Use seed examples to generate the rule set R_1
3. Use rule set R_1 to identify a larger set of pairs P_1
4. Use pair set P_1 to generate an expanded rule set R_2

With only the resources to manually evaluate two iterations, we prune minimally at each step for this initial experiment: we cannot afford to expand *too* slowly. A more conservative approach (i.e. more pruning) would likely yield better results in the long run.

Step 1: Seed Examples

The seed examples we selected were three common instances of invention in the domain of our test corpus (described later):

- (*George Stephenson, steam locomotive*)
- (*Karl Benz, boxer engine*)
- (*Nicolaus Otto, internal combustion engine*)

Step 2: Generating R_1

We retrieved 1000 Web documents for each seed example. After extracting a set of rule candidates and pruning minimally (α=2 and β=1), we were left with 21 rules in our first rule set R_1. Most of these are reasonable (though not perfect) indicators of INVENT. For instance, there were seven proposition rules included in the set, shown here with their frequency counts:

RULE	frequency
built (sub:X, obj:Y)	30
invented (sub:X, obj:Y)	19
inventor (ref:X, of:Y)	15
patented (sub:X, obj:Y)	8
developed (sub:X, obj:Y)	6
engine (ref:X, premod:Y)	3
created (sub:X, obj:Y)	2

The low-frequency rule *engine* (ref:X, premod:Y) looks the oddest and is probably the result of overgeneralization from "*the Karl Benz boxer engine*" and "*the Nicolaus Otto internal combustion engine*".[2]

Step 3: Generating P_1

Applying rule set R_1 to Web documents (and then pruning minimally with γ=2) identifies 34 new pairs for the set P_1. Here are the first ten:

- *James Watt, the steam engine*
- *Walter Hunt, the safety pin*
- *Tim Berners-Lee, the World Wide Web*
- *James Naismith, basketball*
- *Alexander Graham Bell, the telephone*
- *Alfred Nobel, dynamite*

[1] For this pilot study, "distinct" rules are those which do not share any stemmed words. We do not want "X leads Y" and led(subject:X, object:Y) to be considered independent for the purposes of example identification.

[2] Still, this rule is surprisingly reliable when applied to our test corpus.

- *Percy Spencer, the microwave oven*
- *Ruth Handler, the Barbie doll*
- *Ralph Teetor, cruise control*
- *Arthur Sicard, the snowblower*
- *Douglas Engelbart, the computer mouse*
- *Ann Moore, the Snugli baby carrier*

All of the 34 pairs are at least arguably valid, with some borderline cases that are too generic to be useful (though fortunately not terribly misleading). For instance, the pairs (*Nikola Tesla, prototypes*) and (*Shakespeare, words*) refer to categories of things rather than a specific invention.

Step 4: Generating R_2

For the second iteration, we generated a new set of rules using the expanded pair set P_1. Pruned minimally (α=3 and β=1), R_2 consists of 206 rules. Some of these rules are still reasonable within a domain with lots of inventions, :e.g.,

- *designed* (sub:X, obj:Y)
- *introduced* (sub:X, obj:Y)
- *demonstrated* (sub:X, obj:Y)
- *father* (ref:X, of:Y)
- *constructed* (sub:X, obj:Y)
- *perfected* (sub:X, obj:Y)

Some are clearly far too general, however, e.g. sold(sub:X, obj:Y) and used(sub:X, obj:Y), or just wrong, e.g. telephone (possessive:X, ref:Y). This level of pruning is almost definitely too lenient; expanding from 21 rules to 206 rules is too big a jump to make in a single iteration.

Results

To evaluate the machine's ability to learn INVENT, we apply a rule set to a corpus and identify all supposed inventors and their inventions. We then manually judge whether INVENT holds, in context, for each (inventor, invention) pair.[3]

Our primary evaluation corpus is a set of 2000 documents about engines (selected under the Möbius program).[4]

Evaluating R_1

We begin by evaluating the rule set R_1, generated on the first iteration using just the three seed examples. As discussed above, this rule set contains 21 rules and was generated with pruning parameter α=2. For comparison, we also evaluate the performance of R_1', generated with α=3. Here are the results:

	Correct	Incorrect	Percent Correct
R_1 (α=2)	407	135	75.1%
R_1' (α=3)	111	12	90.2%

As we can see, the precision of the rule set increases dramatically as α increases. The cost to recall is significant, however. In both cases, the system shows initial evidence of "learning to read".

Comparing R_1 and R_2

Next, we compare the performance of R_1 and R_2. This comparison is of course highly sensitive to the values of α and β at each iteration. As mentioned above, pruning minimally results in an R_2 with 206 rules—likely far too many to be reliable.

To find a suitable R_2, we select pruning parameters that generate a set with precision roughly comparable to that of our R_1. We will then compare the number of correct instances found to see whether progress has been made. In this case we increase α to 5 and β to 25. These seem reasonably proportional given the increase in the size of example set *E* (R_1 was generated from 3 examples, R_2 from 37 examples) and the resulting increase in typical rule frequency (the most frequent rule in R_1 fired 34 times; the most frequent in R_2 fired 947 times). Using these new parameters, R_2 contains 39 rules, a reasonable increase over the 21 rules in R_1.

	Correct	Incorrect	Percent Correct
R_1 (α=2, β=1)	407	135	75.1%
R_2 (α=5, β=25)	430	132	76.5%

Indeed, performance has improved with the second iteration: the new set of rules finds 23 more correct instances of the INVENT relation and eliminates 3 additional false alarms. This indicates the system is indeed continuing to "learn to read" INVENT.

Comparing Rule Types

We also wish to compare the performance of different rule types. R_1 conveniently contained exactly seven rules of each type. We can therefore easily compare rule type performance by breaking down the results of the R_1 evaluation:

	Correct	Incorrect	Percent Correct
Dependency Trees	75	10	88.2%
Propositions	315	86	78.6%
Surface Structure	222	68	76.6%

(Note that these numbers do not add up to the 407 correct instances found by R_1; some instances were found by more than one type of rule.)

[3] We do not double-check the objective accuracy of the source documents. This means, for instance, that (Al Gore, Internet) was judged correct in at least one document.

[4] For a more informative evaluation, we exclude from all our experiments the rule "engine(ref:X, premod:Y)", because it occurs so frequently in the Möbius corpus as to dwarf the occurrence of every other rule. However, since its precision exceeds the average precision of the other rules, this exclusion does not help the system in any way.

We see here that propositions are more accurate and more productive than surface-structure rules. (Dependency trees are even more accurate, but with much lower recall.) Adding some simple heuristics could somewhat improve precision for the surface-structure rules, since the pilot study implementation is somewhat simplistic. However, their recall would still fall short of proposition-based rules.

We can take this further by directly comparing a proposition rule and its most common surface rephrasings. For instance, we take a single proposition—*invented*(sub:X, obj:Y)—and its most common surface rephrasings—"X *invented* Y" and "Y *was invented by* X". The results:

	Correct	Incorrect
invented(sub:X, obj:Y)	102	4
X *invented* Y Y *was invented by* X	94	17

We ignore the difference in precision assuming it could be at least partially improved by heuristics for the surface structure rules. But there is still a nontrivial gap in recall. More surface rephrasings could help close this gap; for instance, if the machine learned the rule "Y (*invented by* X)", it would find the instance "*the cotton gin (invented by Eli Whitney)*". But these cannot solve the entire problem: it would be difficult to generate surface patterns that could detect instances like "*Trevithick also developed steam engines for use in mines and invented a steam threshing machine*" or "*Invented by Australian engineer Alan Burns and developed in Britain by engineers at Pursuit Dynamics, this underwater jet engine uses high pressure steam*", even if we generalize our surface patterns to allow for part-of-speech information as in previous work.

Comparing Corpora

Finally, we evaluated the performance of our rule sets over a more generic corpus. Our target test corpus was rich in INVENT relations (the phrase "X *invented* Y" occurred 78 times in 2,000 documents). Our secondary test corpus is a set of 10,000 generic news documents (one month of the New York Times). In this much larger corpus, the phrase "X *invented* Y" occurs only 5 times, i.e. the INVENT relation is much rarer.

On this corpus, our rules perform poorly:

	Correct	Incorrect	Percent Correct
Möbius corpus	407	135	75.1%
generic corpus	22	187	10.5%

Our performance "out of domain" is very poor. Generic rules like *built* (sub:X, obj:Y) fire often and are usually wrong. In the Möbius corpus, which focuses on engines, "built" usually means "physically constructed". In a generic news corpus, however, "built" is often metaphorical, as in "built a relationship".

At the most basic level, we have not successfully distinguished between constructions that frequently occur when INVENT(x,y) is true (e.g. X *built* Y) and constructions that *only* hold when INVENT(x,y) is true (e.g. X *invented* Y). One way of addressing this problem is to integrate the learning of relations with the learning of concepts, so that each process might influence the other. For instance, if the system can create patterns that incorporate concept constraints, it could generate more precise patterns (for example, the pattern X *built* Y might be quite effective for INVENT if the type of Y were restricted; this could eliminate instances like "*they built trust*"). Another way to approach this problem might be to incorporate a minimal amount of human supervision at each iteration to identify "near misses" that could be fed back to the system as negative examples.

Beyond INVENT (Further Experiments)

To explore this method further, we experimented with several other relations. A few lessons learned are presented here.

First, seed examples clearly dictate what the system will learn. One experiment targeted the EMPLOYED relation. However, the initial seed set contained three company CEOs, and the system essentially learned a COMPANY-LEADER relation instead. Sample proposition rules found by the system were:

- *ceo* (ref:X, of:Y)
- *boss* (ref:X, premodifier:Y)
- *firm* (ref:Y, possessive:X)
- *led* (sub:X, obj:Y)

We observed similar behavior when we targeted the LOCATED-AT relation for people and places. In this case we included two seed instances where the relation was temporary (i.e. a visit) and two where the relation was relatively permanent (i.e. residence). However, the seed instances involving the visits were far more commonly mentioned on the Web, because they happened to involve more famous people. The rules generated during this experiment therefore dealt almost exclusively with temporary locations, e.g.:

- *arrived* (sub:X, in:Y)
- *visit* (possessive:X, to:Y)
- *visited* (sub:X,obj:Y)

We conclude that this algorithm is naturally quite sensitive to seed example selection.

Second, results are better when the relation is expressed in many different ways using many different words. When we targeted the LOCATED relation for facilities and places, almost all of the rules generated were very simple:

- *of* (ref:X, of:Y)
- *in* (ref:X, in:Y)
- X *in* Y
- Y *at the* X

Our process, however, focuses on rules involving nouns, verbs, and adjectives. (The assumption is that very simple, high frequency rules are too dangerous for an iterative process without human supervision; also, that such rules are so limited in number that they may be best handcrafted.) Our iterative process did not perform well on this relation, because it had so few distinct rules to work with. This process appears better suited to relations that are expressed in lexically diverse ways.

Finally, we have shown that proposition rules show promise for improving system performance. However, although retrieving Web documents that satisfy surface patterns is easy, it is less trivial to retrieve Web documents that satisfy a particular proposition rule. Our pilot study did not attempt to handle this problem, and we simply searched for documents containing the same words as the rule. For instance, if the rule was "*worked*(sub:X, at:Y)", the documents returned would have the word "worked" but not necessarily the construction "PER worked at". Our yield for proposition rules was therefore relatively low, which meant they were less useful in the pruning stage; this should be addressed as further work is pursued.

Related Work

There has been a great deal of research to improve concept and relation detection given large amounts of supervised training. For instance, the Automatic Content Extraction (ACE) program regularly evaluates system performance at detecting particular concepts (e.g. Person, Weapon, Vehicle) and relations (e.g. Located, Employed, Citizen), given a substantial amount of supervised training, e.g., 250k words of annotated data. Approaches to this task frequently incorporate a great deal of structural syntactic information. Their obvious weakness is their reliance on manually annotated examples, which are expensive and time-consuming to create.

Co-training attempts to circumvent this problem by using large quantities of unlabeled data (along with a few examples of labeled data) to improve the performance of a learning algorithm (Mitchell and Blum 1998). However, this process requires two distinctly different views of the data which are played off against each other. In our approach, we do use different types of rules to detect concepts and relations, but these would be too similar to be effective in a co-training framework.

The idea of automatically generating patterns from unsupervised text has also been explored in numerous contexts, classically in (Riloff 1996). Deepak Ravichandran and Eduard Hovy (2002) specifically reported on the possibility of automatically generating surface patterns for relation identification; others have explored similar approaches (e.g. Agichtein and Gravano 2000 or Pantel and Pennacchiotti 2006). We build on the pre-existing work based on surface patterns by adding the structural features of proposition and dependency tree patterns and our particular approach to pruning/filtering.

Conclusion

The goal of this pilot study was to explore unsupervised methods for "learning to read" a new concept or relation, specifically focusing on the possibility of using structural features (e.g. dependency parses and predicate argument structure) to improve performance by mining the web. We have described an initial algorithm that incorporates these syntactic features, alongside simpler surface string features. The overall result is promising. The rules based on predicate-argument propositions outperform surface rules in terms of both recall and precision: the top proposition rules find almost half again as many instances as the top surface structure rules, while maintaining higher precision. Compared head to head with lexically similar surface structure rules, syntax-based rules clearly offer increased recall across the board, a crucial advantage for a system that must be able to map from any source text to a knowledge base. Next steps include the integration of concept and relation learning, as well as further research into techniques to increase pattern precision. These preliminary results also suggest that there is much to be extended and refined to improve performance and concrete ways to address the current limitations.

References

Agichtein E. and Gravano L. 2000. Snowball: extracting relations from large plain-text collections. In *Proceedings of the ACM Conference on Digital Libraries*, 85-94.

Blum A. and Mitchell T. 1998. Combining Labeled and Unlabeled Data with Co-Training. In *Proceedings of the 1998 Conference on Computational Learning Theory*.

Pantel P. and Pennacchiotti M. 2006. Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations. In *Proceedings of Conference on Computational Linguistics / Association for Computational Linguistics (COLING/ACL-06)*, 113-120.

Pradhan S., Ramshaw L., Weischedel W., MacBride J., Micciulla L. 2007. Unrestricted Coreference: Identifying Entities and Events in OntoNotes. In *Proceedings of the IEEE International Conference on Semantic Computing*.

Quillian R. 1968. Semantic Memory. *Semantic Information Processing* (Cambridge, MA: MIT Press): 227-270.

Ravichandran D. and Hovy E. 2002. Learning surface text patterns for a question answering system. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, 41–47.

Riloff E. 1996. Automatically generating extraction patterns from untagged text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1044-1049.