NameNet: a Self-Improving Resource for Name Classification

Paul Morarescu and Sanda Harabagiu

Human Language Technology Research Institute
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083-0688
{paul, sanda}@hlt.utdallas.edu

Abstract

This paper presents a semantically structured resource of more than 1,600 Name Classes. This structure is based on the noun hypernymy hierarchies in WordNet, expanded and validated by corpus evidence collected from the World Wide Web. The set of seed examples provided by WordNet is boostrapped and the used to automatically construct an annotated training corpus for each Name Class. The resulting Named Entity resource enables a supervised Named Entity Recognizer to identify all the encoded Name Classes with high accuracy and without any human intervention.

1. Introduction

Names are widely used in written and spoken communication. Some name classes are used more frequently than others. (Przbocki et al., 1999) reports that PERSONS, LOCATIONS and ORGANIZATIONS represent 88% of all names used in a typical news story. Besides these three name types, four others have been considered important by the Message Understanding Conferences MUC-6 and MUC-7. These classes were DATE, TIME, MONEY and PERCENT. Many named entity recognizers were reported to accurately identify these seven classes. For example, IdentiFinder (Bikel et al., 1999) obtains 94.92% on MUC-6 data. More recently, the Conference on Computational Natural Language Learning (CoNLL-2003) focused on four name classes: PERSON, LOCATION, ORGANIZATION and miscellaneous entities that do not belong to the previous three classes. To our knowledge, there are no named entity recognizers that identify numerous other classes of names, such as a large number of biological genii, military operations, terrorist organizations, etc. The main cause is that there are no resources that indicate how many other name classes exist and how they inter-relate. Furthermore, there is little annotated data which is required by named entity recognition methods based on supervised learning.

In this paper we present **NameNet**, a structured resource of name classes that contains both (1) annotations of name instances and (2) semantic relations between name classes. NameNet has an associated **Named Entity Recognizer (NER)**, implementing $IdentiFinder^{TM}$ (Bikel et al., 1999), that is (a) able to identify instances of any name class encoded in NameNet; and (b) bootstrapped by new names discovered on the Internet. The paper presents our methodology of creating NameNet and reports on our current results of using it for extracting information from text.

2. Identifying new name classes

2.1. WordNet as a Name Class Hierarchy

Our starting point in creating NameNet was **WordNet**, the lexico-semantic databases encoded at Princeton (Fellbaum, 1998). WordNet encodes a vast majority of the English nouns, verbs, adjectives and adverbs. Only nouns are relevant for the task of Named Entity Recognition. Synonym words share the same semantic meaning and are encoded in a synonym set, or **synset**.

Formally, we define a **Named Entity (NE)**, or simply **Name**, as a WordNet lexeme that contains one or more capital letters. Examples are “Europe”, “European Union”, “M1” and “al-Qaeda”. We define a **Name Synset (NS)** a WordNet synset containing one or more Named Entities. For example, {Japan current, Kuroshio current, Kuroshio} is a Name Synset. We define a **Name Class (NC)** a WordNet synset that has at least a Name Synset, among its hyponyms. If the Name Class has at least $n = 3$ seeds, we call it a **Seeded Name Class**. Otherwise, we call it a **Non-Seeded Name Class**. For example, {naval battle} is a Seeded Name Class with 17 Name Synsets among its hyponyms, whereas {event(1)}[1] is a Non-Seeded Named Class with one Named Synset among its hyponyms. Similarly, {syndicalism} is not a Named Class because it has no hyponyms.

We found in WordNet 2.0 a total of 45,408 NEs originating in 25,794 NSs which serve as constituents for 4,884 NCs. The distribution of the NCs and NSs over the nine WordNet noun hierarchies is illustrated in Table 1

We are interested in finding the semantic classes of the NEs in WordNet, and to this end we use the hierarchical structure of noun synsets. As an example, we present the derivation of a semantic hierarchy of NCs contained in the WordNet noun hierarchy rooted at the {event(1)} synset.

The initial structure of the hierarchy, as extracted from WordNet, illustrated in Figure 1, is the input of our top-level procedure which has three goals: (1) to bootstrap the training data available for each NC (2) to identify new NCs in WordNet; and (3) to restructure and validate the semantic hierarchy of NCs based on corpus evidence retrieved from the Internet.

[1]{event(1)} represents the first semantic sense of the {event} synset in WordNet

WordNet root	Seeded NCs	Seeded NSs	Non-seeded NCs	Non-seeded NSs
group (1)	141	6012	188	238
abstraction (6)	174	1455	385	494
entity (1)	1198	12768	2189	2880
psychological_feature (1)	67	826	156	183
state (4)	19	83	102	127
event (1)	11	182	54	61
act (2)	28	273	128	147
possession (2)	1	3	12	13
phenomenon (1)	5	18	26	31
TOTAL	1644	21620	3240	4174

Table 1: The distribution of Name Synsets and Name Classes over the WordNet noun hierarchies

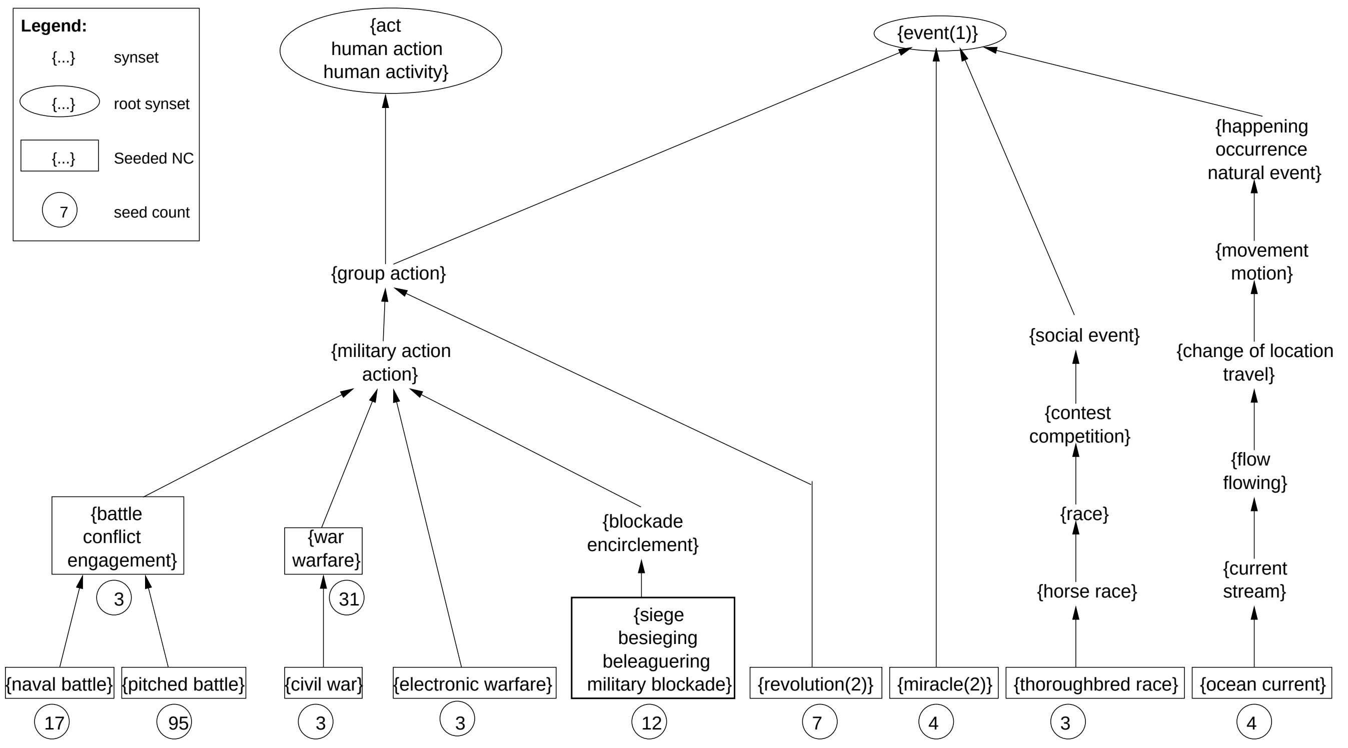

Figure 1: Example of an initial Name Class Hierarchy extracted from WordNet 2.0

2.2. Bootstrapping the data for Name Classes

The bootstrapping procedure is applied on each Seeded Name Class NC_i from WordNet. Based on documents retrieved by the Google search engine, the following procedure extends the set of NEs in NC_i and generates an annotated corpus that is used directly to train a supervised NER that identifies NC_i.

Step 1 Collect the seeds of the Name Class NC_i = $\{NE_1, NE_2, ..., NE_n\}$, where NE_j, $1 \leq j \leq n$, are the seeds of NC_i, i.e. the Named Entities in all its direct hyponyms.

Step 2 Given the seeds NE_j and the lexemes WF_k, $1 \leq k \leq m$, in Name Class NC_i, we generate a set of queries for the Google search engine. The general syntax of each query is an OR boolean operation between all WF_k conjoined with an (implicit) AND boolean operation between all NE_j:

google_query = (WF_1 OR WF_2 OR ... OR WF_m) AND (NE_1 NE_2 ... NE_n)

For example, a Google query for the {battle, conflict, engagement} Name Class illustrated in Figure 1 is *("battle" OR "conflict" OR "engagement") AND ("Armageddon" "Battle of Britain" "Drogheda")*. A Google query for the {revolution(2)} Name Class is *("revolution") AND ("American Revolution" "Chinese Revolution" "Cuban Revolution" "English Revolution" "French Revolution " "Mexican Revolution" "Russian Revolution")*

Google returns only ten results for each query, thus we need to resubmit the query repeatedly to retrieve more than 10 results. Since (Calishain and Dornfest, 2003) report that the results retrieved by Google depend on the order of the query words, we shuffle the query words randomly each time we resubmit the query. Moreover, each alias of the same name in WordNet (such as {Passero, Cape Passero,

Passero Cape}) has the same probability of being used as a query term.

Step 3 Filter out the web-related content, such as HTML tags and JavaScript code, from all documents such that we are left only with the actual text. Also, filter out all-capital texts because our NER uses capitalization as a hint in identifying NEs. We paste all the remaining text content into a single document D.

Step 4 Split D into sentences using the sentence boundary detector described in (Ratnaparkhi-99). Discard all sentences that do not contain any NE.

Step 5 Generate the training corpus for our NER by from the set $\{S_j\}$ of all the sentences in D such that each S_j contains one or more NEs from NC_i and no NE that is not in NC_i. To provide negative examples, merge this set of sentences with a similar set of sentences collected for an arbitrary NC_k from a different WordNet noun hierarchy[2]. Train the NER on the resulting set of sentences.

Step 6 Generate the testing corpus for our NER from the set $\{S_j\}$ of all sentences in D such that each S_j contains one or more NEs which are not in NC_i. Test the NER on this set of sentences.

Step 7 All the NEs which are not in NC_i and have been annotated by the NER in the testing phase are added to NC_i seeds.

Step 8 The procedure is restarted from step 2 with the the new set of seeds, until we have *enough*[3] seeds or the last iteration ended with no NEs being added to NC_i.[4]

The result of this procedure is (1) a set of new NCs; and (2) annotated corpora than can be used to train our NER to identify instances of all the old and new NCs.

2.3. The identification of new name classes

As illustrated in Figure 1, semantic relations connect NCs. For example, a hyponymy (IS-A) relation exists between {naval battle} and {battle, conflict, engagement}. We know that *Trafalgar* is a NE of {naval battle} and it is important to decide whether it is also a NE of {battle, conflict, engagement}. We claim that a class NC_i must satisfy certain conditions for the names identified as its examples to be identified as examples of NC_j, where NC_i is a hyponym of NC_j. Namely, NC_i must tag with our NER the training instances of all hyponyms of NC_j, as well as all the NEs of NC_i, as defined in section 1.2.

Given two name classes $NC_i \neq NC_j$, such that either (NC_i is-a NC_j) or $\exists$ NC_k such that (NC_i is-a NC_k) and (NC_j is-a NC_k), the procedure to decide whether the names of NC_i and the names of NC_j are classified in the same NC by our NER is the following:

1) Split the corpus of NC_i into 90% for training and 10% for testing

2) Train the NER on the training corpus of NC_i

3) Test the NER on the testing corpus of NC_i using tenfold cross-validation

4) Test the NER on the corpus of NC_j

5) Use hypothesis testing to decide which of H_1: $NC_i = NC_j$ and H_2: $NC_i \neq NC_j$ is better supported by the data

3. Experimental Setup and Results

The procedures described in this paper require no human intervention. However, the users can set a few parameters according to their particular task:

(1) The minimum number (n) of seeds which defines a Seeded Name Class, which we set to 3 in the Introduction.

(2) The number of seeds that is considered satisfactory to stop the bootstrapping algorithm. We used fixed values like 30 or 50 as well relative values that were multipliers of the original number of seeds of each class.

(3) The confidence interval used to decide whether two or more NCs are to be considered as a single NC for the practical NE recognition task. A fine tuning may be required here.

All the Seeded Name Classes in Figure 1 were successfully boostrapped by the procedure in section 2.2, as shown in Table 2

NC	New NEs
{battle, conflict, engagement}	141
{naval battle}	8
{pitched battle}	21
{civl war}	4
{war, warfare}	12
{electronic warfare}	0
{siege, besieging, beleaguering, military blockade}	3
{revolution(2)}	6
{miracle(2)}	10
{thoroughtbred race}	5
{ocean current}	1

Table 2: The Boostrapping Results

The 141 new NEs of the {battle, conflict, engagement} propagated from its two hyponyms {naval battle} and {pitched battle}.

A new Seeded Name Class represented by synset {military action, action} resulted from the unification of all its Name Class hyponyms illustrated in Figure 1.

There is no gold standard against which the precision and recall of the boostrapping procedure can be computed automatically. Thus we performed a manual validation of the NC hierarchy in Figure 1 after the boostrapping, and obtained the values of 81.4% and respectively 83.1%.

4. Applications

NameNet is useful for domain-specific Information Extraction. In a domain like Terrorism it is more useful to identify terrorist organizations than organizations in general. This enables the acquisition of more precise and efficient extraction patterns.

A hierarchical structuring of Name Classes like the one we propose in NameNet improves the recognition of instances of subsuming classes, as more corpus evidence is

[2]Since NCs from different WordNet noun hierarchies should not be recognized as the same NC by the NER

[3]"enough" is an application-dependent integer parameter that can be set by the user

[4]We used the latter as the stop condition in our experiments

provided by their subclasses. For example, recognizing *Syracuse* as an instance of the {siege} class, we produce a new positive example for the {military action} class and thus enhance the performance of the recognizer of that class.

New classes and new names are retrieved from the Internet in addition to those derived directly from WordNet. They are all integrated in the same semantic hierarchy.

5. Future Work

We plan to recognize the most typical tagging styles of Gazetteers, which proved useful to validate new Name Classes like {terrorist organization} (see for example the *Current List of Designated Foreign Terrorist Organizations (as of October 5, 2001)* by the US Department of State at http://www.state.gov/s/ct/rls/rpt/fto/2001/5258.htm, retrieved by Google in the first set of 10 results to our query)

We plan separate searches in on-line dictionaries such as HyperDictionary (http://www.hyperdictionary.com) and The Free Dictionary (http://www.thefreedictionary.com), which Google often ranks among the best results to our queries.

6. Related Research

The first set of Name Classes (Grishman and Sundheim, 1996) had the 7 instances mentioned in the Introduction. The number was suited for applications like Information Extraction on a small number of domains related to business activities. It soon became evident that each new domain or task, like Document Summarization or Question Answering, needed various new Named Classes. Some new Name Classes proved difficult to detect because they had sub-classes which featured significant variability.

(Sekine et al., 2002) propose a set of 150 Name Classes structured in a 3-level hierarchy, with a design merging newspaper corpus evidence, thesauri and existing systems and tasks. The paper also describes inherent ambiguities in the definition of the concept of Named Entity. For instance, Acura Integra is the name of a model of a car and not a name of a particular entity. Thus one may ask whether Acura Integra can still be treated as a NE.

7. Acknowledgements

We would like to thank our colleague Alessandro Moschitti for co-authoring our NE recognizer. Many thanks to John Asmuth for writing a simple and publicly available Python API for WordNet, as well as to Mark Pilgrim for writing a similar Python API for Google, both of which were very helpful.

8. References

Bikel, D. M., R. Schwartz, and R. M. Weischedel, 1999. An algorithm that learns what's in a name. *Machine Learning Journal Special Issue on Natural Language Learning*.

Calishain, Tara and Rael Dornfest, 2003. *Google Hacks*. O'Reilly & Associates, Inc., 1st edition.

Fellbaum, Christiane, 1998. *WordNet*. Bradford Books, 1st edition.

Grishman, R. and B. Sundheim, 1996. Message understanding conference - 6: A brief history. *COLING*.

Przbocki, M., J. Fiscus, J. Garofolo, and D. Pallett, 1999. 1998 hub4 information extraction evalation. *Proceedings of the DARPA Broadcast News Workshop*:13–18.

Sekine, S., K. Sudo, and C. Nobata, 2002. Extended named entity hierarchy. *LREC*.