Alex Gruenstein
CS224N
Spring 2001
Final Project

Learning Hypernyms from Corpora

Abstract

Building on previous work by Sharon Caraballo and Marti A. Hearst, I describe an algorithm which extracts hypernym relations among nouns extracted from parsed corpora. Specifically, I investigate how automatically acquiring more hypernym data can be used to increase the accuracy of such a system.

Introduction

A word *w* is said to be a hypernym of another word *v* if the sentence "*v* is a (kind of) *w*" is true. For instance, *tool* is a hypernym for *hammer* since most people would agree that a hammer is indeed a type of tool. A fair amount of effort has already been put into creating databases of words and their hypernyms. Wordnet (Fellbaum, 1998), for example, allows for the lookup of all hypernyms for a word (in fact, it is more specific than this, in that one can actually look up hypernyms for specific senses of a given word).

Wordnet has proved to be a useful resource – however, expanding Wordnet to include new words (and their hypernym relationships) takes a large amount of human effort. Moreover, it is unrealistic to expect that Wordnet should contain hypernym relationships for specialized, domain-specific vocabularies. As Riloff and Shepherd point out, "Many domains are characterized by their own sublanguage containing terms and jargon specific to the field" (1997).

Such information, however, is often extremely useful for natural language processing tasks. Natural language processing tasks are often limited to specific domains. Moreover, ontologies describing items in these domains must often be hand-crafted so that natural language understanding can occur.[1] Extracting hypernym-hyponym relationships automatically for specific domains could often expedite the process of creating natural-language tools for those domains.

In my project, I build a noun hierarchy via bottom up clustering of nouns that appear as heads of noun phrases in *The Wall Street Journal*. I make use of the parsed and tagged corpus of *The Wall Street Journal* in the Penn Treebank. I then attempt to assign appropriate hypernyms to each cluster, again based on data obtained from *The Wall Street Journal*. My project is built, more or less, on top of the work of Sharon A. Caraballo (Caraballo 1999). I first create a baseline system modeled after the one she describes. Next, I attempt to improve upon her system by extracting more hypernym data from the corpus and putting it to better use. Finally, I compare the baseline system modeled after hers with the final system I arrive at after attempting to make improvements I describe below.

[1] A specialized domain I am especially interested in, and one which motivated me to work on this project in the first place, is the newly emerging domain of conversations with autonomous robots. I am a member of a team at Stanford's Center for the Study of Language and Information which is creating spoken-language dialogue systems which facilitate human commanding of autonomous robots. Our goal at the WITAS project is to create a multi-modal interface for commanding an unmanned aerial vehicle. In order to do this, we have to build a specialized ontology so that the system can "understand" the domain in question. One part of this involves coding such specialized facts that a "Yamaha" and a "Scandicraft" are both types of helicopters – a hypernym-hyponym relationship. In order to help facilitate the automatic construction of similar dialogue systems in the future, we would like to automate the creation of ontologies with facts like this as much as possible. Being able to extract hypernym-hyponym relationships from transcripts of real conversations in a domain would facilitate one step in this complex process.

I. Baseline

For a jumping-off point, I use the technique described by Sharon Caraballo with some of my own tweaks. The algorithm works as follows. First, we cluster nouns that appear in the text using a bottom-up hierarchical clustering technique. After such clustering, we end up with a binary tree, with nouns at the leaves. Next, we assign the best hypernyms that we can to each internal node in the tree. Finally, we compress the tree: we do away with the artificial restriction that it be a binary tree and try to give it more meaningful internal nodes – each internal node will contain the (up to) three best hypernyms for the nouns appearing as leaves in the subtree for which it is the root.

I will describe this baseline system first, as well as the previous work which motivates building a system to extract hypernym-hyponym relationships in this way. In the next section, I will discuss the specific improvements I made to the system.

Step 1: Extracting noun co-occurrence data

In order to assign a hypernym to a group of nouns, we first need an algorithm for clustering nouns into groups. We want to create clusters of nouns that all belong to the same category so that we can than assign hypernyms for each category based on data we will collect on known hypernyms for some percentage of the nouns in that category. If a noun is a hypernym for a large portion of the nouns in the group, then the chances should be good that it is a hypernym for the entire group. With this goal is mind, we aim to find an algorithm that will categorize the nouns in such a way that making such hypernym assignments will be valid.

Riloff and Sheperd (1997) and then Roark and Charniak (1998) assert that words often appear in the context of other words in the same category. Riloff and Shepherd bring to light the examples of conjunctions, lists, appositives, and nominal compounds. The assertion is that nouns which appear together in conjunctions, lists, appositives, and nominal compounds often belong to the same category. By finding nouns that appear with the same sorts of other nouns in conjunctions, lists, appositives, and nominal compounds, and then grouping them together, one can hope to create semantically valid groups of nouns. That is, one hopes to create groups of nouns which are similar enough in meaning that known hypernyms for some of the words can be generalized to all of the words in a cluster.

For my baseline system, I follow the lead of Caraballo and extract the head nouns that appear in conjunctions (for example, "the horse and the cow") and appositives (for example, "the cow, a spotted animal") from the parsed and tagged *Wall Street Journal* corpus in the Penn Treebank (Marcus, et al 1993). When the head noun phrase is a noun compound (like "Alex Gruenstein"), I consider each word from the phrase as a distinct noun. So, for example, a conjunction of multi-word proper names like "Hong Kong and Taiwan" would be stripped down to the individual nouns "Hong," "Kong," and "Taiwan." (One possible improvement to such a system would be to recognize instances of compound nouns like "Hong Kong" and leave them together, while still stripping apart compound noun phrases that ought to be separated). Each noun is stemmed using a Porter Stemmer (thanks to Fotis Lazarinis for his Java implementation of the stemmer I found at http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/porter.java).

I implemented the algorithm for pulling out such nodes in java – *tgrep* proved not to be quite powerful enough to pull out every case of what I wanted. I used ANTLR to create a parser/lexer to read in the tree structure of the parsed/tagged data and then custom java rules to examine these trees. My algorithm extracted 11,533 distinct (after stemming) head nouns.

I then applied a frequency threshold, and created a co-occurrence vector for only those nouns that appeared at least twice in the corpus. There were 5,016 such distinct nouns. Each entry in the co-occurrence vector corresponded to the number of times that noun was seen in an appositive or conjunction with another head noun (which might itself have frequency less than two) in the corpus.

Step 2: Bottom-up hierarchical clustering

The algorithm next performs bottom-up hierarchical clustering on the nouns, as described, for example, in chapter 14 of Manning and Shütze. The measure of similarity used was cosine. So the similarity between two vectors is defined as such (see Manning and Shütze or Caraballo):

$$sim(\mathbf{v}, \mathbf{w}) = \cos(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \bullet \mathbf{w}}{|\mathbf{v}||\mathbf{w}|}$$

In order to do bottom-up hierarchical clustering we must first put each noun in its own node. We then create a list of all such nodes – call this list the *nodeList*. We then calculate the similarity between every two nodes in the *nodeList* and cluster together the two most similar nodes by joining them under a common parent. The two nodes that are clustered together are removed from the *nodeList*, while the newly created parent node is added to the *nodeList*. This process continues until we have a single node (the root of the binary tree) in the *nodeList*. At this point, we stop.

In order to cluster nodes which don't themselves contain nouns, we must define a measure of similarity that will include such nodes. We define each such a node in terms of its leaves – we say that the cluster defined by a node consists of all the leaves under that node in the tree. We then calculate the similarity between two such nodes by defining it to be the average of the similarity of each noun in one group to each noun in the other group. Specifically, as defined in (Caraballo 1999), we calculate the similarity between two such groups as:

$$sim(A, B) = \frac{\sum_{\mathbf{v} \in A, \mathbf{w} \in B} \cos(\mathbf{v}, \mathbf{w})}{size(A)size(B)}$$

where size(A) is the number of leaves under the node A and size(B) is the number of leaves under B.

When actually implementing the system, however, we can calculate the similarity between two nodes more efficiently by using the property that that if we combine two nodes A and B under a new node C, we can calculate the similarity between C and any other node *i* like so (again, the notation is borrowed from Caraballo, and the ideas come both from her and Manning and Shütze):

$$sim(C,i) = \frac{sim(A,i)size(A) + sim(B,i)size(B)}{size(A) + size(B)}$$

I designed the clustering algorithm in Java. Using multi-threading techniques, I was able to do a large portion of the calculation in parallel. Running on Stanford's *junior* machine, I was able to do the clustering relatively quickly, even using an inherently slow language like Java. *junior*'s large amount of a memory and large number of processors meant that all the data could be kept at once in memory and that, via multi-threading, a few hours worth of cpu work could be done in just a few minutes (so long as junior wasn't too busy doing work for other people).

Step 3: Extracting hypernym data

Hearst (1992) discusses techniques for automatically finding hypernym relationships among nouns in English-language corpora. Hearst makes the definition that an item w_2 is a hypernym for an item w_1 if native speakers of English will accept the sentence "A w_1 is a (kind of) w_2" (and, in turn, w_1 is a hyponym for w_2). Hearst suggests several patterns that appear in English that imply hypernym relationships among words. By looking for the patterns that Hearst describes, we should be able to algorithmically discover hypernym relationships between nouns that appear in our corpus.

For the baseline system, I followed Caraballo's work and focused exclusively on conjunctions and disjunctions of noun phrases. Specifically, the algorithm looks for phrases in the text of the form '*NP {, NP}* {,} {or | and} other NP*' (corresponding to Hearst's patterns numbered 3 and 4). So, for example, when we find a noun phrase like "hammers, drills, {and | or } other tools," we conclude that *tool* is a hypernym for both *hammer* and *drill*.

As in my previous corpus search, I designed a Java program that traversed *The Wall Street Journal* corpus in the Penn Treebank. It looked for noun phrases that matched this pattern, and then pulled out the head of each child noun phrase. Each *hyponym* that was identified was stemmed so that it would match the previous data. However, each *hyperym* was not stemmed. This breaks a bit from the system Caraballo describes – in her system, she stems all of the nouns discovered in the search. However, few of the hypernyms appear as multiple forms of the same stem, since in the constructions in question hypernyms most often appear as plurals. Moreover, the entire compound noun phrase is sometimes much more specific; consider "show business people," for example, in comparison to just "people." In consideration of these factors, I did not stem the hypernyms identified in the search.

Using the technique just described, my Java program identified 209 distinct candidate hypernyms. It identified 504 words as appearing with at least one of these hypernyms.

Step 4: Assigning hypernyms to the internal nodes of the tree

In this next step, we attempt to find the best possible hypernym for each internal node of the tree. Recall that each internal node in the tree defines a unique cluster of nouns – specifically, the group of all nouns that appear as the leaves of the tree rooted at that internal node. Since the nouns in each cluster are supposed to be semantically related, the hope is that if many of the nouns share a hypernym in common, this hypernym should correctly describe the entire cluster.

In following this logic (and Caraballo 1999), we start by creating a hypernym vector for each node in the tree. Each slot in the vector corresponds to one of the distinct hypernyms we pulled out in the previous step. For every leaf in the tree, we iterate over each slot in the vector and put a 1 in it if the hypernym in this slot appeared as a hypernym for the word in question, or a 0 if it did not occur as a hypernym for the word in question. The hypernym vectors for each internal node is then defined to be the sum of its children's vectors.

To choose the best hypernym for each group of nouns defined by each internal node, we simply choose the hypernym corresponding to the slot in the hypernym vector with the largest number. In other words, the best hypernym for a cluster of nouns is simply the hypernym that appeared in the text in a hypernym relationship with the most number of nouns in that cluster.

To account for times when good hypernyms might alternate (as in the example of "country" and "nation" given in (Caraballo 1999)) or when a single hypernym might not be adequate in describing all of the nouns below it, we also label each internal node with the second and third best hypernyms (if there are any) for that node. To choose the second and third best hypernyms, we again just choose the words with the second and third largest counts in the hypernym vector for a given node.

After this process, each internal node is labeled with 0, 1, 2, or 3 best hypernyms describing the cluster of nouns defined by that node. A node is labeled with 0 hypernyms when its hypernym vector consists entirely of zeros.

At the end of this step, we have a tree whose root is labeled with the three hypernyms: *areas*, *products*, and *investors*.

Step 5: Compressing the tree, or Bye Bye Binary

The tree that comes out of the bottom-up clustering is necessarily binary. However, this fact doesn't accurately reflect the fact that nouns can be classified in groups of arbitrary size; there is no reason to assume that they should be clustered in a binary way. Before assigning the hypernyms to each internal node, we might have found it difficult to cut up the tree into separate clusters – we had no criterion for removing internal nodes. However, after assigning the (up to) three best hypernyms to each internal node, we now have a straightforward way to eliminate redundant internal nodes. As in (Caraballo 1999), we do a depth-first search of the tree. When we come across an internal node, we examine each of its children in turn. If the child is also an internal node and it has the same (up to) three best hypernyms as the parent, or the child has no hypernyms, we remove the child from the tree and attach its children directly to its parent node.

Using this method, we reduce the redundancy of having a node's children all labeled with the same hypernyms as the node itself. Moreover, since the root of our tree is labeled, every internal node in the tree is now guaranteed to be labeled, meaning that we no longer have any unlabeled nodes.

Baseline results:

The final tree created by the algorithm described above has 29 internal nodes and a depth of 12. The three best hypernyms of the root are *areas*, *products, investors*. When we keep in mind that the data for this study is taken from *The Wall Street Journal* corpus, this assignment of hypernyms for the root node seems more reasonable. The children, grand-children, and great-grand-children internal nodes of the root appear in Table 1 below..

Hypernymns	Nouns in cluster
areas	5
problems	429
show business people / establishments	57
→ show business people	34
→ establishments	23
areas / investors countries	3682
→ investors / areas / institutions	3486
→ → areas / investors / issues	2125
→ → officers /creditors /committee members	1361
→ countries / u.s. possessions / nations	102
→ → u.s. possessions	38
→ → countries / nations	25

Table 1: Child, grand-child, and great-grand-child internal nodes of the root for the baseline System

II. Improvements

The system described above is very similar to the one described in (Caraballo 1999), with a few exceptions noted above. It uses less data than her system, since her system uses *The Wall Street Journal* as well as some data acquired via an automatic parser. As such, my final tree is smaller than the one she produces. Moreover, since less data is used, we will see that it is a bit less accurate than her system.

In the context of these facts, I felt it prudent to try to look at the relative improvement that various tweaks could have on the system, rather than comparing my system directly to hers. Using the system developed above as a baseline standard, I experimented with ways of improving upon it in the context of the corpus data it in fact used. Specifically, I targeted both the relative amount of hypernym data collected from the corpus and the way in which this data was used in building the noun hierarchy.

Collecting more hypernym data

In pursuit of using hypernym data to improve the accuracy of the resulting tree, I first looked at ways of extracting more relevant data from the corpus. In her paper, Caraballo considers phrases of the form '*NP {, NP}* {,} {or | and} other NP*.' Hearst, however, identifies several other lexico-syntactic patterns which can be used to extract hypernyms from text corpora. In order to acquire more hypernym data from the corpus, I expanded my Java parser to look for more of the patterns identified in Hearst.

Specifically, my improved parser targeted the following patterns identified by Hearst as indicative of hypernym/hyponym relationships:

- *NP such as {NP ,}* and NP*
    - *e.g.* “tools such as hammers, drills, and pliers”
- *NP such as {NP ,}* or NP*
    - *e.g.* “tools such as hammers, drills, or pliers”
- *such NP as {NP,}* and* NP
    - *e.g.* “such tools as hammers, drills, and pliers”
- *such NP as {NP,}* or* NP
    - *e.g.* “such tools as hammers, drills, or pliers”
- *NP {, NP}* {,} and other NP*
    - *e.g.* “hammers, drills, pliers and other tools”
- *NP {, NP}* {,} or other NP*
    - *e.g.* “hammers, drills, pliers, or other tools”
- *NP* {,} *including* {*NP,*}* *and* NP
    - *e.g.* “tools, including hammers, drills, and pliers”
- *NP* {,} *including* {*NP,*}* *and* NP
    - *e.g.* “tools, including hammers, drills, or pliers”

In the examples above, the head noun *tool* would be identified by the parser as a hypernym for each of the head nouns *hammer*, *drill*, and *pliers*.

The improved parser extracted 504 unique hypernym nouns from the exact same *Wall Street Journal* corpus compared with the 209 unique hypernym nouns extracted by the baseline parser. This is a 241% increase in the number of unique hypernyms extracted. Moreover, it identified 1,625 nouns as hyponyms for these unique hypernyms, compared with 504 discovered by the baseline system. This is a 322% increase.

Such a dramatic increase in the number of hypernyms and words appearing as hyponyms for those hypernyms means that was have a dramatic increase in the amount of data we can use when labeling the internal nodes of the trees with the best hypernyms available and in compressing the tree. Using the algorithm described above with this improved hypernym data leads to what appears to be a much richer tree than the one produced by the baseline system. The greater number of hypernyms allows the algorithm to draw finer distinctions when it is cutting up the tree in the compression stage. The resulting tree has a depth of 15, and 101 labeled internal nodes (compared with 12 and 29 respectively in the baseline tree). The root of the tree is labeled with *products*, *areas*, and *markets* as the three best hypernyms. The increase in depth and number of internal nodes certainly seems indicative that at the very least finer distinctions are being drawn in the clustering – we definitively end up with more clusters than we did before. Before comparing my system to the baseline system, I made one additional improvement.

Improving clustering: getting even more out of hypernym data

As a result of the previous improvement, the system is now armed with much more data about nouns and appropriate hypernyms for these nouns. It seemed prudent to put this data to work at an earlier stage in processing: clustering. In the baseline system, the clustering of the nouns is based purely on co-occurrence statistics between head nouns appearing in appositives and conjunctions together. The idea behind using this data for clustering is that nouns that appear together often in conjunctions and appositives are semantically related. By clustering them together and then looking at the known hypernyms of words in the groups, we can hope to label nouns that have no hypernym data from the corpus with appropriate hypernyms because of their similarity to other words for which we have data.

Since my system is designed specifically to assign hyperynms to clusters of words, I felt that a natural extension to the clustering stage of the system would be to incorporate the hypernym data itself in the clustering stage. Rather than just using the hypernym vectors we created for the nouns in the tree in Step 4 (above) for assigning hypernyms to already formed clusters, my system also put this hypernym data to use in the clustering algorithm itself. The rationale behind this move is that each cluster is supposed to represent nouns that share a hypernym – so nouns we know a priori share a hypernym, should have a higher probability of being clustered together.

The question, then, is: how exactly should the hypernym data be incorporated into the clustering stage of the algorithm? I chose the following straightforward approach. First, Collect the hypernym data from the corpus and then assign the hypernym vectors to each noun, as we did in Step 4 (above). Next, we simply append these vectors to the co-occurrence vectors we previously used exclusively in clustering. Finally, we cluster in exactly the same hierarchical, bottom-up fashion as except we use the cosine of our new, merged vectors as our similarity measure. Using this technique, we now favor clustering together nouns which appear with the same hypernyms. My hope was that these nouns would cluster together first (since they would be clustered together before other similar nouns which had no hypernym data), and then nouns for which there was no hypernym data in the corpus would be added into these clusters based on their similarities to the words which already are known to share hypernyms.

The resulting tree has a depth of 18, with 109 labeled internal nodes. The root node is labeled with the three best hypernyms of *products*, *areas*, and *markets*. The children, grand-children, and great-grand-children internal nodes of the root can be found in table 2.

Hypernymns	Nouns in cluster
products / markets / companies	4887
→ products / companies / countries	4866
→ → fields	24
→ → disasters	24
→ → activities	75
→ → construction trade	36
→ → programs / problems / instruments	418
→ → show business people / establishments	42
→ → goods / toy line	41
→ → countries / companies / markets	3809
→ → products / estate / items	173
areas / provinces	6
→ areas	5

Table 2: Child, grand-child, and great-grand-child internal nodes of the root for my final system

III. Evaluation (and Conclusion)

There are a couple of factors which ought to be considered in evaluating my final version of the system against the baseline system. First, the depth of the tree and the number of internal nodes is an important objective consideration. A greater depth and more internal nodes mean that the system is creating more clusters and drawing finer distinctions between the clusters. More clusters can be a good thing because it can result in greater specificity of the hypernyms in its cluster. We could build a trivial system that extracted all nouns out of a corpus, put them in a single cluster, and labeled that cluster with the hypernym "thing." Such a system would have an accuracy of 100% (or very close to 100%), however it would not prove very useful.

The goal of our system ought to be to find the most specific hypernym for each cluster – that is, the hypernym which is not a hypernym for any other noun which is a hypernym for the same cluster. For example, *thing* is not the most specific hypernym for

{ *hammer*, *pliers*, *screwdriver* } because at least one more specific hypernym exists: *tools*, since *thing* is a hypernym for *tools*, which in turn is a hypernym for { *hammers, pliers, screwdriver* }. By always choosing the most specific hypernym for a group of nouns, we could hope to build a more coherent tree – one where the ancestors of internal nodes could themselves be hypernyms for the hypernyms labeling the internal nodes. This property emerges somewhat from the system developed here, but could more fully emerge with even more hypernym data and research into ways of using hypernyms of hypernyms data to form the tree.

One indirect indication of such specificity is the number of internal nodes in the tree (as well as the number of hypernyms actually identified in the corpus). For the most part, more internal nodes should indicate that the system is finding more specific hypernyms for clusters of nouns. A greater depth of the tree should also be indication of this. Table 3 shows these statistics for the system, as it was improved over time. I take the final system's greater number of internal nodes and depth as an indication of an improvement over the baseline system, though I can't be absolutely certain that these numbers are an indication that the system is actually outputting more useful data.

System	**Number of Internal Nodes**	**Depth**
Baseline	29	12
More Hypernym Data	101	15
More Hypernym Data & Hypernym Data Used in Clustering	109	18

Table 3: Comparison of the structure of the tree created by the system as it progressed from baseline to its final version

In following (Caraballo 1992), I thought it also important to try to rate the accuracy of the system. In order to collect similar statistics to Caraballo, I compared the tree structure created by the baseline system to that created by my final system by asking human judges to rate the accuracy of the two systems. The three human judges were undergraduate students majoring in Human Biology, Chemical Engineering, and Computer Science respectively. None of them have studied much linguistics, and none of them have studied computational linguistics. They knew very little about my project, and were not told exactly why they were being asked to make these judgments. They were not told which set of clusters was from my final system and which was from the baseline system.

For each system, I randomly selected 10 internal nodes which were parents to at least 20 nouns. From each of these internal nodes, I chose 20 of their children nouns at random as well as 5 "noise" nouns taken randomly from elsewhere in the tree. The three judges then judged whether each of the 25 words (20 actual children and 5 noise nouns) was a hyponym of any or all of the (up to) 3 best hypernyms used to label the internal node. The 5 noise nouns in each group were inserted to ensure that the judges were being fair. Very few of them were ever chosen by the judges, and in the few cases they were actually chosen, they usually did appear to actually be valid hyponyms which had come in from other branches on the tree clustered under the same (or a similar) hypernym label. The results of the judges ratings for 10 random internal nodes chosen from both the baseline system and my final system appear in tables 4 and 5 below.

Baseline System				
Three best hypernyms	**Best Hypernym**		**Any Hypernym**	
	majority of judges	**any judge**	**majority of judges**	**any judge**
issues /air-freight companies	1	6	4	7
areas / issues / companies	4	7	5	12
problems	4	12	4	12
areas / issues /purposes	2	3	5	8
matters	5	13	5	13
areas	4	5	4	5
issues	1	2	1	2
show-business-people	0	0	0	0
establishments	2	3	2	3
issues / companies / defendents	1	6	2	7
AVERAGE	2.4 (12.0%)	5.7 (28.5%)	3.2 (16%)	6.9 (34.5%)

Table 4: Judges' evaluation of baseline system

My System				
Three Best Hypernyms	**Best Hypernym**		**Any Hypernym**	
	majority of judges	**any judge**	**majority of judges**	**any judge**
instruments / variables / securities	3	4	3	11
instruments / goodies / variables	3	8	5	13
construction trade	0	2	0	2
disasters	3	5	3	5
activities	5	7	5	7
instruments	1	4	1	4
goodies	3	9	3	9
programs / reforms	0	4	0	7
fields	2	14	2	14
instruments / securities / alternatives	0	5	5	9
AVERAGE	2 (10.0%)	6.2 (31.0%)	2.7 (13.5%)	8.1 (40.5%)

Table 5: Judges' evaluation of my final system

The numbers under the “Best Hypernym” section indicate the number of hypernym-hyponym pairs rated either by a majority of the judges or any of the judges to be valid in each category, where the hypernym in question is only the best hypernym chosen by the system for that cluster. The “Any Hypernym” section indicates the number of hypernym-hyponym pairs rated either by a majority of the judges or any of the judges to be valid in each category, where the hypernym in question is any one of the (up to) three best hypernyms assigned to the cluster by the system.

The data indicates that the baseline system actually performed better when we consider pairs picked only by a majority of the judges, whereas my final system performs better when we consider pairs picked by any of the judges.

We must look at these results in the context of the trees created by the two systems. I argue that even though the accuracy of the two systems are relatively similar, the fact that my final system creates more clusters and assigns more labels, while maintaining about the same accuracy, means that it is actually improved. On looking over the trees compared by the two systems, it seems to generally be the case that the baseline system assigns more general labels to each node than does my system. It’s difficult to quantify this, but if we assume it is true, then even though my system is not much more accurate than the baseline system, the fact that it manages to maintain the same accuracy while creating more clusters makes it arguably more useful.

My conclusion, then, is that gathering more hypernym data and using this data for clustering, arguably makes for a better tree.

IV. Other Paths

I conclude with a brief discussion of other paths I went down in working on the system that didn't make it into the final system.

One attempt I made at improving accuracy was in the stage where the tree was compressed. I considered using some additional information in deciding when to delete internal nodes, merging their children with their parent's children. Specifically, I wanted to use additional information to decide when a hypernym qualified as one of the three best hypernyms for a cluster. In addition to a hypernym having to have one of the three highest counts for in the hypernym vector for that node, I stipulated that one of the following two conditions must be met:

1. The number of words in the cluster for which this candidate appeared as a hypernym must be at least a certain percentage of the total number of words in the entire tree for which this candidate appeared as a hypernym

or

2. The candidate hypernym must have occurred for at least a certain percentage of the nouns in the cluster.

I felt that condition 1 was necessary because it wouldn't be fair to rule out a candidate hypernym just because it didn't appear with a high percentage of the words in the cluster, if all of the words which we knew were valid hyponyms appeared in the cluster. Condition 2 was requisite because it wouldn't be fair to rule out a candidate hypernym for a cluster if a high percentage of words in the cluster were known to be valid hyponyms.

Experimenting with these conditions proved difficult. I tried various combinations of parameters for the two percentages, but the trees they produced would either be radically different from one another or nearly identical. I didn't want to have to call in human judges to rate thousands upon thousands of relationships in order to rate the accuracy of each different tree. A study with more time and human judges would be useful, however, in determining how these 2 conditions could be used to improve performance. Various combinations did *seem* to improve performance, but rather than involving too many variables in my final evaluation, I felt it better to leave these criteria out of the final system.

V. Previous Work

As discussed throughout the paper, I modeled my baseline system on the work of (Caraballo 1999). (Roark and Charniak 1998) focus on noun-phrase co-occurrence statistics in order to automatically build semantic lexicons. Specifically, they use the same sort of noun-phrase constructions focused on here in order to cluster words semantically. (Hearst 1992) identifies noun-phrase structures which are indicative of hypernyms in text corpora.

VI. Acknowledgements

Thank you to Anna Liess, Lizzie Hammerman, and Adam Simpkins for acting as my human judges

Submitted Archive Description

The major components of the system are in *Similarity.java*. This module does all the clustering, hypernym assigning, and compressing. The extraction of the hypernyms and noun phrase data is done by Extract.java, ExtractHead.java, and ExtractHypernyms.java. In order to use these modules, you must install ANTLR.

I have included the output tree from the final version of the system in the archive, as well as the hypernyms extracted from the text by the final version of the system. To view it, compile Similarity.java and then run:

```
Similarity view tree.out hypernyms.txt
```

The tree will come up in a new window. Please use jdk1.2 or greater for compiling and running in java. The system should work fine on any sweet-hall Uix machine.

References

Biess, Ann, et al (1995). Bracketing Guidelines for Treebank II Style Penn Treebank Project. Technical Report, University of Pennsylvania.

Caraballo, Sharon A. (1999). Automatic Construction of a Hypernym-Labeled Noun Hierarchy from Text. In *37th Annual Meeting of the Association for Computational Linguistics: Proceedings of the Conference*, pages 120-126.

Hearst, Marti A. (1992). Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the Fourteenth International Conference on Computational Linguistics.*

Manning, Christopher D. and Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

Riloff, Ellen and Jessica Shepherd (1997). A Corpus-Based Approach for Building Semantic Lexicons. In *Proceedings of the Second Conference on Emperical Methods in Natural Language Processing*.

Roark, Brian and Eugene Charniak (1998). Noun-Phrase Co-Occurrence Statistics for Semi-Automatica Semantic Lexicon Construction. In *COLING-ACL '98: 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics: Proceedings of the Conference,* pages 1110-1116.