# Identifying Relational Knowledge for Textual Entailment


## Abstract

In the context of Textual Entailment, it has been argued (e.g.,(Maccartney and Manning, 2008)) that many inferences are largely compositional and depend on the ability to recognize relations between entities, noun phrases, verbs, adjectives etc. For example, it may be important to know that a *blue Toyota* is not a *red Toyota* nor a *blue Honda* but that all are cars, and even Japanese made cars. This is a different problem than variations of relation extraction studied in the literature, that aim at extracting relations between entities that co-occur in a given snippet of text.

In this paper, we propose a novel approach to detect and classify relations between entities and other phrases in support of textual entailment. Given a pair of entities or phrases we identify relations that might exist between them and give them labels if possible. Our method makes use of Wikipedia as a source for background knowledge and disambiguates among concepts referring to input entities, along with a notion of prominence with respect to a given text collection.

We evaluate our system on a large set of pairs of instances taken from 40 classes, and significantly achieve an improvement over 29% in average F1-score compared to systems that use existing resources. We also show that, our system also significantly outperforms the existing resources even when input entities are covered in those resources.


## 1 Introduction

Inference in natural language requires the use of large amounts of background knowledge. For example, it may be important to know that a *blue Toyota* is not a *red Toyota* nor a *blue Honda* but that all are cars, and even Japanese made cars. This is a different problem than variations of relation extraction studied in the literature that aim at extracting relations between entities that co-occur in a given snippet of text. While the extraction of knowledge of this sort has also been discussed in the literature, it has been studied mostly in the context of large scale knowledge acquisition - extract all *easy to find* facts in a given corpus (Banko and Etzioni, 2008; Davidov and Rappoport, 2008; Paşca and Van Durme, 2008; Bunescu and Mooney, 2007). This knowledge is typically existential; e.g., while it is true that A is of type B, say, it is not clear if it is commonly B; moreover, if the textual inference system needs to know if A and B are related, and how, this relation will appear in the acquired knowledge base only if A and B have occurred in close proximity in a sentence (and in a easy to understand way). Consequently, it is difficult for a textual inference system to benefit from the knowledge acquired this way. Indeed, we know of no successful application of the large scale existential knowledge acquisition efforts to textual inference.

In the context of Textual Entailment, for example (Dagan et al., 2006; Haghighi et al., 2005; Braz et al., 2005), it has been argued, e.g., (Maccartney and Manning, 2008)) that many inferences are largely compositional and depend on the ability to recognize specific relations between entities, noun phrases, verbs, adjectives etc. For example, it is often necessary to know of an *ancestor* relation (and its directionality) in order to deduce that a statement with respect to the *child* (e.g., George Bush) holds for an *ancestor* (e.g., a republican leader). This is illustrated in the following example, taken from the RTE4 test suite:

> **T**: Nigeria's National Drug Law Enforcement Agency (NDLEA) has seized 80 metric tonnes of cannabis in one of its largest ever hauls, officials say.
>
> **H**: Nigeria seizes 80 tonnes of drugs.

Similarly, it is often important to know of a *cousin* relation to infer that a statement about Sun may *contradict* an identical statement with respect to HP (at least without additional information) since these are *different* companies. This is illustrated in the following example (where inference requires both identifying a cousin and an ancestor relation):

> **T**: A strong earthquake struck off the southern tip of Taiwan at 12:26 UTC, triggering a warning

from Japan's Meteorological Agency that a 3.3 foot tsunami could be heading towards Basco, in the Philippines.

**H**: An earthquake strikes Japan.

This paper proposes to address the problem of relation identification and classification in a form that is directly applicable to textual inference. Specifically, our system accepts two input arguments (entities or noun phrases) and detects the relation between them along with its possible label (e.g. *economic problems* is a possible class for *global warming* and *food crisis*). We focus here on the *ancestor* relation and the *cousin* relation, that were identified as key relations also in (Maccartney and Manning, 2008) (the *cousin* relation is call an *alternation* there, and the *ancestor* relation is call *forward entailment* and *backward entailment*). Following the inference logic developed there, we expect that the resource we develop in this work can be used compositionally to support robust textual inference.

We develop an approach that makes use of Wikipedia to recognize the classify the relations between a pair of arguments on the fly. While Wikipedia is vast resource that is rapidly being updated, our approach needs to take into account that it is developed by a large number of people, and is thus very noisy and non-uniform. Our algorithmic approach therefore treats Wikipedia and its category structure as an open resource and uses statistical text mining techniques to get robust information from this noisy resource.

For example, the entity *Ford* appears many times in Wikipedia, and is part of a large number of categories. We need to disambiguate it and determine which category it belongs to and, within this category, which specific entity is intended. We make use of a notion of prominence with respect to a given text collection (in this case, with respect to Wikipedia itself) under the assumption that in textual entailment applications we are in search of some notion of "common sense" knowledge.

We evaluate our system by comparing its performance over a large number of pairs chosen from over 40 classes, with a large scale effort done in forming the extended WordNet (Snow et al., 2006). We show significantly better results in terms of coverage and accuracy. Furthermore, we show that even when all entities are covered by the extended WordNet, our system still significantly outperforms the baseline.

The rest of this paper is organized as follows. In Section 2, we briefly mention about previous work that inspires the proposal of our approach. Section 3 formalizes the problem and describes our algorithmic approach to relation detection and classification. Our experiments and results are described in Section 4. Discussion and future work are in Section 5. We give the concluding remarks of our paper in Section 6.

## 2 Previous Work

(Snow et al., 2005) constructs a hypernym-only classifier using the dependency path patterns discovered in the sentences that contain noun pairs in hypernym/hyponym relation. Furthermore, they show that using coordinate terms can help improve the recall of the hypernym classifier. The best system in their work is a hypernym-only classifier additionally trained with hundred thousands of dependency paths extracted from Wikipedia corpus. The system outperforms the best WordNet classifier in identifying coordinated terms by relatively improving over 54% in F-score.

In the other hand, (Ravichandran et al., 2005) addresses the challenge of dealing with very large amount of data by employing efficient randomized algorithms to quickly cluster the noun similarity lists. The system first collects all nouns and their features, and after that constructs feature vectors for these nouns. The system then hashes all nouns into a hash table using the Locality Sensitive Hash functions, which can put similar feature vectors close to each other in the hash table. By using this approach, they can discover thousands of distinct clusters of English nouns by exploring over 70 million webpages. Recently, (Snow et al., 2006) apply this algorithm and also the data of noun similarity lists to train their ($m$,$n$)-cousin relationship classifier. They combine their hypernym-only classifier and the ($m$,$n$)-cousin classifier with the evidence features to recognized the hypernym and cousin relationships among nouns. Their inferred taxonomy achieves the best performance after adding 30,000 novel hyponyms compared to those in WordNet-2.1. They show that their system relatively improves 23% in F-score over the WordNet-2.1 hypernym classifier.

Along this research direction, (Paşca, 2007; Paşca and Van Durme, 2008) proposed approaches to automatically acquire open-domain classes of entities and attributes by using very little super-

vised seed information. Web documents and query logs are used in the acquisition process. By applying their method, they are able to extract 4,583 classes associated with an average of 189 instances for each class. However, each instance has one parent and no further ancestor. There is no hierarchical structure among classes and instances.

All of these work utilizes either supervised data to train their classifier or a very large amount of data in web-scale to mine the interested facts. On one hand, supervised approaches requires a lot of well labeled data to train a good classifier. It is infeasible to use a large amount of available unannotated text to support the classification task. On the other hand, approaches exploring very large amounts of data may potentially have problem because of their inflexibility when one wants to add newly appearing documents into the corpus and re-generate the facts. We address these issues by using Wikipedia and propose an efficient light-weight approach using the Wikipedia category structure to identify the relations between entities.

## 3 Relation Identification

In this section we first give our definitions of ancestor and cousin relationships. After that in section 3.2, we describe the Wikipedia category structure. We show that the Wikipedia category structure is naturally fit with the demand of identifying hierarchical relationship between concepts. We then present our algorithms and its extension in sections 3.3 and 3.4.

### 3.1 Ancestor and Cousin Relationships

We follow the definitions in (Snow et al., 2006) to define the hypernym and $(m, n)$-cousin relationships between senses in WordNet. In their paper, a hypernym relationship is denoted by $H_{ij}^n$ if a sense $j$ is the $n$-th ancestor of a sense $i$ in the hypernym hiararchy. The notation is simplified to $H_{ij}$ to denote that sense $j$ is an ancestor of sense $i$ at some level. On the other hand, the sibling relationship between senses $i$ and $j$ is generalized to $(m, n)$-cousin relationship which defines that sense $i$ and sense $j$ have their *least common subsummer* (LCS) in within exact $m$ and $n$ links, respectively.

We adopt these definitions to define the ancestor and cousin relationships between entities by using Wikipedia articles and Wikipedia category structure, which we will describe in Section 3.2. We *loosely define* that entity $X$ is an ancestor of entity $Y$ if the title of one of the articles about $X$ in Wikipedia *matches* one of the categories or ancestor categories of the articles about $Y$ in Wikipedia. We also follow the $(m, n)$-cousinhood definition to define the cousin relationship. Entities $X$ and $Y$ are cousins if the articles about $X$ in Wikipedia *share* one or more categories with the articles about $Y$, also in Wikipedia.

For instance, follow our definition, we can verify that *color* and *red* have an ancestor relationship because there is an article entitled "Red" (for entity *red*) that belongs to the category **Color** which is the title of another article about entity *color*. We also can verify that *red* and *green* belong to two articles (one entitled "Red", and the other one entitled "Green") which share a common caretory **Color**. Therefore, they are cousins.

### 3.2 Category Structure of Articles in Wikipedia

Wikipedia[1] is a freely available encyclopedia on the web developed by the contribution of a huge number of internet users around the world. Recently, more and more research that focuses on exploiting the valuable knowledge resources in Wikipedia has been carryied out (Chang et al., 2008; Richman and Schone, 2008; Suchanek et al., 2007; Gabrilovich and Markovitch, 2007). There are millions of articles in Wikipedia; and this number is increasing rapidly day by day. At the time of February, 2009, there are 2,738,772 articles in the English Wikipedia[2]. Each article in this encyclopedia is collaboratively written by volunteers and categorized into different categories by the authors of the article. One can observe that, although the content of articles in Wikipedia may contain false information at some point in its development, the categories of an article are quite determinable.

Essentially, as a nature of things, a certain concept may belong to several topics. Similarly, each article in Wikipedia will often be in several categories. And each category itself is also an article in Wikipedia; so it has its own supercategories. For example, the article with the title $t$ = *George W. Bush* belongs to a set of categories at the fist level $\mathcal{C}^1$ = {*Category:George W. Bush*, *Category:Presidents of the United States*,

[1] http://www.wikipedia.org

[2] http://en.wikipedia.org/wiki/Size_of_Wikipedia

*Category:Harvard University alumni*, ...}; at the second level of $t$ we have $\mathcal{C}^2 = \{$*Category:Heads of government by country*, *Presidents by country*, *American politicians*, ...$\}$.

From the properties of categories, they do not form a strict hierarchy or tree of categories. This allows multiple categorization schemes to co-exist simultaneously. As a guideline in constructing categories of Wikipedia articles, there is no constraints from constructing loops in the category space, but this is strongly discouraged. In our work, we consider the Wikipedia category structure as a directed acyclic graph.

Interestingly, as a matter of fact, articles in the same category or in its sub-categories are considered siblings. This fact is stated clearly in the guideline of the category structure in Wikipedia[3]. For one who wants to find the siblings of an article, he or she can not only look at the articles in the category but also can find more in its sub-categories. This valuable property of Wikipedia category directly suggests an efficient approach to identify relations between entities as long as they have their own article in Wikipedia.

### 3.3 Identifying Entity Relations Using Wikipedia Categories

In this section, we present our algorithms for identifying the possible ancestor and cousin relationships between entities.

For any entity $X$, we define the followings variables.

- $\mathcal{T}_\mathcal{X}$: the set of the top ranked articles returned by searching $X$ in in the pool of all *article titles* in Wikipedia.
- $\{c^L_{t_X}\}$: the set of all categories extracted for the titles $t_X \in \mathcal{T}_\mathcal{X}$ by going up to $L$ levels of ancestor.
- $\mathcal{C}_X = \bigcup_{t_X \in \mathcal{T}_X} \{c^L_{t_X}\}$: the union set of categories extracted for all titles in $\mathcal{T}_\mathcal{X}$ by going up to $L$ levels of ancestor.

Futthermore, for a category, we can possibly determine its generalized name which we call generalized category, and its domain by analyzing the category using lexical and syntactic information. For instance, with category $c$ = *Cities in the United States*, we locate its generalized name as *Cities*, and its domain as *the United States*. By this, we define the followings variables.

- $\mathcal{G}_X$: the union of all generalized category names of the categories of entity $X$, in $\mathcal{C}_X$.
- $\mathcal{D}_X$: the union of all domains of the categories of entity $X$, in $\mathcal{C}_X$.

[3]http://en.wikipedia.org/wiki/Wikipedia:Category

**Algorithm 1** AncestorRelationIdentification

```
1: Input: Wikipedia index I_W
2:       K: number of top documents
3:       L: number of levels of ancestor.
4:       (X, Y): two input entities.
5: Output: 0: no ancestor relationship.
6:         1: X is an ancestor of Y.
7: T_X ← SearchByTitle(X, I_W, K);
8: T_Y ← SearchByTitle(Y, I_W, K);
9: C_Y = {};
10: for each t_Y ∈ T_Y do
11:    {c^L_{t_Y}} ← RecursivelyExtractCate(t_Y, L)
12:    C_Y = C_Y ∪ {c^L_{t_Y}};
13: end for
14: G_Y ← CategoryGeneralization(C_Y);
15: D_Y ← DomainExtraction(C_Y);
16: E_Y = C_Y ∪ G_Y ∪ D_Y
17: for each t ∈ T_X do
18:    if (Match (t, E_Y)) then
19:       return 1;
20:    end if
21: end for
22: return 0;
```

Algorithm 1 describes our approach to identifying the ancestor relationship between two input entity $X$ and $Y$. The algorithm returns **1** as an indicator for $X$ to be an ancestor of $Y$. In lines 7 and 8 of the algorithm, we respectively search for articles in Wikipedia that contain all tokens of $X$ and $Y$ in their titles. Only top $K$ most relevant titles are returned. From line 9 to line 13, all categories up to $L$ levels of ancestor of the retrieved articles of $Y$ are collected. We extract the generalized category names and domains of all extracted categories of $Y$ by calling functions *CategoryGeneralization* and *DomainExtraction*, respectively. After that, from line 16 to 22, the union set $\mathcal{E}_Y$ of all categories, their generalized category names, and their domains is produced and compared with the article titles of $X$. If there is any *match* happens between titles in $\mathcal{T}_X$ and elements in $\mathcal{E}_Y$, the algorithm returns the answer indicating that $X$ is an ancestor of $Y$ and terminates, otherwise, the algorithm concludes that there is no ancestor relationship between $X$ and $Y$.

The cousin relation identifier is described in Algorithm 2. In lines 8 and 9, the articles in Wikipedia which have the titles containing all

**Algorithm 2** CousinRelationIdentification

1: Input: Wikipedia index $I_W$
2: $K$: number of top documents
3: $L$: number of levels of ancestor.
4: $(X, Y)$: two input entities.
5: Output: $(0, \{\})$: no cousin relationship.
6: $(1, \mathcal{N})$: $X$ and $Y$ are cousin.
7: $\mathcal{N}$ is the list of possible class names.
8: $\mathcal{T}_X \leftarrow$ SearchByTitle($X$, $I_W$, $K$);
9: $\mathcal{T}_Y \leftarrow$ SearchByTitle($Y$, $I_W$, $K$);
10: $\mathcal{C}_X \leftarrow$ CategoryExtraction($\mathcal{T}_X$, $L$);
11: $\mathcal{C}_Y \leftarrow$ CategoryExtraction($\mathcal{T}_Y$, $L$);
12: $\mathcal{G}_X \leftarrow$ CategoryGeneralization($\mathcal{C}_X$);
13: $\mathcal{G}_Y \leftarrow$ CategoryGeneralization($\mathcal{C}_Y$);
14: $\mathcal{N} =$ Match($\mathcal{C}_X, \mathcal{C}_Y$) $\cup$ Match($\mathcal{G}_X, \mathcal{G}_Y$)
15: **if** ($\mathcal{N} \neq \emptyset$) **then**
16: **return** $(1, \mathcal{N})$;
17: **else**
18: **return** $(0, \{\})$;
19: **end if**

**Algorithm 3** RelationIdentification

1: Input: Wikipedia index $I_W$
2: $K$: number of top documents
3: $L$: number of levels of ancestor.
4: $(X, Y)$: two input entities.
5: Output: $(0, \{\})$: $X$ and $Y$ have no relationship.
6: $(1, \{\})$: $X$ is an ancestor of $Y$,
7: $(2, \{\})$: $Y$ is an ancestor of $X$,
8: $(3, \mathcal{N})$: $X$ and $Y$ are cousin,
9: $\mathcal{N}$ is the list of possible class names.
10: **if** (AncestorRelationIdentification($X$,$Y$)) **then**
11: **return** $(1, \{\})$;
12: **else**
13: **if** (AncestorRelationIdentification($Y$, $X$)) **then**
14: **return** $(2, \{\})$;
15: **end if**
16: **end if**
17: $(I, \mathcal{N})$ = CousinRelationIdentification($X$,$Y$)
18: **if** ($I == 1$) **then**
19: **return** $(3, \mathcal{N})$;
20: **else**
21: **return** $(0, \{\})$;
22: **end if**

words in two entities are retrieved, respectively. After that, the union set of categories are extracted by going upto $L$ levels of ancestor, for all titles in $\mathcal{T}_\mathcal{X}$, line 10, and $\mathcal{T}_\mathcal{Y}$, line 11. Function *CategoryExtraction* has been actually showed in Algorithm 1 from line 9 to 13. After that, the generalized category names of all extracted categories are induced for $X$ and $Y$. Finally, the matching functions will look for matches between two sets of categories and generalized category names. If there is any match happening, it will be put into a list of all possible class names for $X$ and $Y$. The algorithm is going to return the class name list along with an indicator indicating that $X$ and $Y$ have cousin relationship. Otherwise, it returns **0** for no relation between $X$ and $Y$.

After having algorithms to identify ancestor and cousin relations between entities, we design the relation identification algorithm, which is described in Algorithm 3. Input of the algorithm is a pair of entities ($X$, $Y$). The algorithm outputs the predict relationships between $X$ and $Y$, including ancestor, cousin, or no relation. If two entities bind a cousin relationship, the algorithm also returns a list of possible class names of the two entities. For instance, the list of class names would be {*American film actors*, *film directors*, ...} for the input pair (*Mel Gibson*, *Tom Cruise*). From line 10 to 16, the algorithm first checks if there is ancestor relationship between two entities using Algorithm 1. The algorithm immediately returns the answer if there is indeed an ancestor relationship between $X$ and $Y$, otherwise, the algorithm continues identifying the possible cousin relationship between two entities. Otherwise, the algorithm conclude that there is no relation for two input entities.

### 3.4 Prominence-based Search

As far as we observe the algorithms, we can see that the *SearchByTitle* function guide the identification prediction heavily. If the function returns irrelevant or unimportant articles about entities, we easily get wrong relations. Therefore, designing a good entity search function is important. In this section, we propose an approach called prominence-based search to improve the search results of input entities. Although our initial approach for prominence-based search is simple, it can serve as an disambiguation tool for entities.

We guide the search engine based on an assumption that, without special purpose, people often want to see information known or important entities retrieved when they search for information. We translate this assumption into our search engine by returning only top relevant articles about prominent entities. We measure the prominence of entities by their occurrence in a large text collection. For instance, if one wants to search for entity *bush*, the search engine may retrieve thousands of articles including *Kate Bush*, *William Bush*, *Laura Bush*, *George W. Bush*, *Bush family*, and so on. Among these articles, our system retrieves the ones about *George W. Bush*, *George H. W. Bush* in the top of the returned article set because their frequency of occurrence dominates other entities.

## 4 Experiments and Results

### 4.1 Experiment Data and Setup

To evaluate our approach in identifying ancestor and cousin relationships between entities, we use the data of 40 target classes of almost 11,000 instances in (Paşca, 2007). Each target class is an incomplete set of representative instances. Table 4.1 shows a snippet of this dataset. Interestingly, we have both types of closed word class (e.g. ChemicalElem, Country), and open word class (e.g. BasicFood, Hurricane). Moreover, there are classes with common nouns (e.g. BasicFood with *rice*, *milk*, *eggs*), and there are also classes with proper nouns (e.g. Actor with *Mel Gibson*, *Sharon Stone*).

| Class (Size) | Examples of Instances |
|---|---|
| BasicFood (155) | rice, milk, eggs, beans, fish |
| ChemicalElem (118) | lead, copper, aluminum, calcium |
| City (589) | San Francisco, Dubai, Chicago, |
| Disease (209) | arthritis, hypertension, influenza |
| Actor (1500) | Kate Hudson, Mel Gibson |

Table 1: This table shows a snippet of 40 target classes with instances.

We create experiment data by the following guidelines.

- Examples of ancestor relation: we randomly pair a class with one of its instances. The order of the class and the instance in the example defines the direction of the relation.
- Examples of cousin relation: we randomly pair two instances in a class.
- 3 types of negative examples (i.e. examples with two entities having no ancestor or cousin relation): we can randomly pair either: (1) a class and another class, (2) a class and an instance of another class, or (3) an instance in one class and instances in other classes.

Both classes and instances are considered entities in the experiments. Table 4.1 shows some examples in our test data.

We set up three experiments two evaluate our algorithms in Section 3.3.

1. Experiment 1: Evaluating the ancestor identifier (*see* Algorithm 1). Positive examples are those of the ancestor relations. Cousin examples are considered negative examples and added into the test set with other types of negative examples. The identifier is correct if positive examples are recognized having ancestor relation, and no relation for other examples
2. Experiment 2: Evaluating the cousin relation identifier. Positive examples are those of the cousin relations and all types of negative examples are used to evaluate the cousin identifier. We do not use examples which have ancestor relation in this example. Given that the cousin identifier is going after the ancestor one in the decision tree as described in Algorithm 3, we take out the ancestor examples so that we can measure the upper bound in accuracy of the cousin identifier. If the identifier recognize two entities are cousins, it will also output a list of possible class of the entities.
3. Experiment 3: Evaluating the overall algorithm to identify relations. In this experiment, we evaluate the performance of the relation identifier described in Algorithm 3. All kind of examples including ancestor examples, cousin examples, and all types of negative examples are used in this experiment.

| Entity 1 | Entity 2 | Relation |
|---|---|---|
| Actor | Mel Gibson | A |
| BasicFood | rice | A |
| Wine | Champagne | A |
| Paris | Dubai | C{City} |
| copper | oxygen | C{ChemElem} |
| Nile | Volga | C{River} |
| Roja | C++ | N |
| egg | Vega | N |
| HotBot | autism | N |

Table 2: Some examples in the test data. A, C, and N denote ancestor relation, cousin relation, and no relation, respectively. The entities in curly brackets are the target class.

Beside these three experiments, we also report other evaluations which can be found in Section 4.3. In our experiments, all systems, including the baseline system described in Section 4.2, use 2 as the maxinum level of ancestor in the hierarchical structure one can go up from an entity. We use the Apache Lucene Information Retrieval library[4] to implement the search engine to search for the Wikipedia articles having titles that contain the input entities. Lucene is a high-performance text

[4]http://lucene.apache.org

search library that is written in Java and is a widely used off-the-shelf IR system.

### 4.2 Baseline system

We compare our system with the baseline system that uses the extended WordNet in the work of (Snow et al., 2006) as the background knowledge to infer relationships of entities. Starting from WordNet-2.1 (Fellbaum, 1998), the extended WordNet has augmented 400,000 synsets so far. Words that are added into the extended WordNet can be common nouns or proper nouns.

The baseline system identifies a pair of entities having ancestor relation if one of the entities is the $k$-th hypernym of the other one in the extended WordNet hierarchical tree. Two entities are cousins if they share common ancestors (synsets in WordNet) within at most the maximum level of ancestor in the WordNet tree, which is 2 in our setting. Similar to our framework in Algorithm 3, the cousin identifier goes after the ancestor one in the decision tree to recognize the relation between entities. If both ancestor and cousin identifiers fail to recognize the relation of a pair, the pair is decided as having no relation.

### 4.3 Results

In experiment 1 that evaluates the accuracy of our ancestor identifier, we use all 10,894 instances with their class to be the positive data. We automatically generate 10,000 negative examples including 2,000 cousin examples and 8,000 pairs of all types of negative examples. Table 3 shows the precision, recall, and the accuracy of our algorithm compare to the baseline system that uses the extended WordNet as source of knowledge.

| System | Precision | Recall | Accuracy |
|---|---|---|---|
| Baseline | 73.3 | 50.0 | 63.8 |
| Ours | **98.6** | **75.0** | **86.4** |

Table 3: Performance of the ancestor relation identifier of both baseline and our systems.

Our ancestor identifier outperforms the baseline system with almost 23% of improvement in accuracy. Especially, our ancestor relation identifier has very high precision, which indicates that it can serve as a very reliable filter for our whole relation identification system.

Next, we evaluate our cousin relation identifier separately with the ancestor one. In this experiment, we randomly generate three sets of testing data. Each test set contains 2,000 positive examples (i.e. cousin examples), and 2,000 negative. We do not use the ancestor examples in this experiment. Table 4 presents the average precision, recall, and accuracy over three test sets of our system and the baseline.

| System | Precision | Recall | Accuracy |
|---|---|---|---|
| Baseline | **74.3** | 31.5 | 60.3 |
| Ours | 68.2 | **74.1** | **69.8** |

Table 4: Average precision, recall and accuracy of the cousin relation identifier of both the baseline and our system. Note that the ancestors examples are not included in the test sets of this experiment.

The baseline system is better in the precision, but our system outperforms the baseline in general with almost 10% improvement in accuracy.

After experiencing our identifiers separately, we are going to evaluate them together in the framework (*see* Algorithm 3) that can detect whether a pair of entities has ancestor relation, cousin relation, or neither of them. In this experiment, we follow the second experiment by generating 3 sets of testing data. Each test set has 6,000 examples including 2,000 ancestor examples, 2,000 cousin examples, and 2,000 negative examples of all types. We report the details of the evaluation in Table 5. Our system achieves significant improvements over the baseline system, in both the ancestor and cousin identifiers. Our ancestor identifier gets a 34.6% improvement in average F1-score over the baseline ancestor identifier; and our cousin detector significantly outperforms the baseline system by over 50% in average F1-score in the overall system. Combining with the F1-score on identifying examples having no relation, our system achieves a significant improvement of over 29% in average F1-score from the baseline system.

We go one more step further to evaluate the baseline and our system against a test set where all entities and classes exist in the extended WordNet, which is the knowledge source the baseline uses. Out of 10,894 entities in the 40 class data set, we first extract all entities which exist in the extended WordNet. Moreover, for classes whose the name does not exist in the extended WordNet, we remove those classes and their instances. After this step, we have 9,158 (about 84% of the whole data set) entities remaining in the data set. We randomly generate positive and negative examples for ancestor and cousin relations. After generating the

| | Set 1 | | Set 2 | | Set 3 | | Average | | |
|---|---|---|---|---|---|---|---|---|---|
| | Prec. | Rec. | Prec. | Rec. | Prec. | Rec. | Prec. | Rec. | F1-Score |
| **Baseline** | | | | | | | | | |
| Ancestor | 49.1 | 38.75 | 48.7 | 39.8 | 48.7 | 40.2 | 48.8 | 39.6 | 43.7 |
| Cousin | 31.9 | 10.35 | 31.9 | 10.2 | 32.6 | 10.2 | 32.2 | 10.2 | 15.5 |
| None | 46.6 | 87.9 | 47.1 | 87.8 | 47.5 | 88.5 | 47.0 | 88.0 | 61.3 |
| **Ours** | | | | | | | | | |
| Ancestor | 95.5 | 66.1 | 96.2 | 66.4 | 94.9 | 66.7 | 95.5 | 66.4 | **78.3** |
| Cousin | 62.1 | 74.35 | 60.5 | 74.0 | 61.8 | 75.2 | 61.4 | 74.5 | **67.3** |
| None | 59.9 | 66.5 | 59.8 | 65.1 | 60.8 | 65.7 | 60.2 | 65.75 | **62.8** |

Table 5: Performance of our relation identification system and the baseline in details. *Ancestor*, and *Cousin* denote the performance of the ancestor and the cousin identifiers, respectively. *None* denotes no relation.

test set, we have 780 positive examples for ancestor relation, 780 positive examples for cousin relation, and 1000 negative examples. Both the baseline and our system are evaluated on this test set. Note that all entities in this test set are guaranteed to exist in the extended WordNet. Table 4.3 shows the results of this experiment.

| | Precision | Recall | F1-Score |
|---|---|---|---|
| **Baseline** | | | |
| Ancestor | 48.9 | 44.1 | 46.4 |
| Cousin | 39.9 | 13.6 | 20.3 |
| None | 58.4 | 92.9 | 71.7 |
| **Ours** | | | |
| Ancestor | 99.4 | 64.2 | **78.0** |
| Cousin | 67.6 | 76.2 | **71.6** |
| None | 73.8 | 86.9 | **79.8** |

Table 6: Performance of our system and the baseline in details. In this experiment, entities are guaranteed to be covered in the extended WordNet.

From the results of this experiment and the experiments above, it is clearly shown that our approach significantly outperforms the extended WordNet, which is one of the popular large scale knowledge bases.

## 5 Discussion

We experienced that the data set of 40 classes of instances we are using in this work mostly contains unambiguous entities (i.e., entities that do not share meanings with many other entities). We do not have ambiguous entities such as *Bush* or *Ford* in this dataset. Therefore, the prominence-based search (described in Section 3.4) did not have a chance to display its ability of disambiguating entities. We evaluated our ancestor identifier without prominence-based search on the same test set we used in Experiment 1. We got 86.2% in accuracy, compared to the accuracy of 86.4% of the identifier with prominence-based search. However, we manually selected a list of several commonly used name entities such as *Bush*, *Ford*, *Gates*, *Jobs*, etc., and tried our search engine on this list. We got the important people that we want to know about most of the time. This implies that our system potentially works well on other data sets which may contain ambiguous entities.

## 6 Conclusion

Our work was motivated by the need to support textual inference. This is a knowledge intensive task, but current knowledge acquisition efforts have not addressed the problem in a way that allows inference systems to use it, in tasks such as textual entailment. While most of the current information extraction systems work by extracting possible entities that appear in close proximity and their relations, from a large text collection, we have not seen any successful application of these large scale existential knowledge acquisition efforts to textual inference. In this paper, we propose an efficient approach that identifies a "knowledge need" on the fly. We focused on two key relations – the ancestor and the cousin relations – which have been shown to have a principle role in supporting compositional inference. Refining this is key future direction. Our approach makes use of Wikipedia and its category structure, which serves as a conceptual network and allow a quick identification of the relation between entities and the relevant class membership. We show that our approach significantly outperform systems that use existing large scale data as knowledge source. Furthermore, we show that even when all entities are covered by the extended WordNet used in the baseline, our system still significantly outperforms the baseline system.

## References


Michele Banko and Oren Etzioni. 2008. The tradeoffs between open and traditional relation extraction. In *Proceedings of ACL-08: HLT*, pages 28–36, Columbus, Ohio, June. Association for Computational Linguistics.

R. Braz, R. Girju, V. Punyakanok, D. Roth, and M. Sammons. 2005. An inference model for semantic entailment in natural language. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1678–1679.

Razvan Bunescu and Raymond Mooney. 2007. Learning to extract relations from the web using minimal supervision. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 576–583, Prague, Czech Republic, June. Association for Computational Linguistics.

M. Chang, L. Ratinov, D. Roth, and V. Srikumar. 2008. Importance of semantic represenation: Dataless classification. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, July.

I. Dagan, O. Glickman, and B. Magnini, editors. 2006. *The PASCAL Recognising Textual Entailment Challenge.*, volume 3944. Springer-Verlag, Berlin.

Dmitry Davidov and Ari Rappoport. 2008. Unsupervised discovery of generic relationships using pattern clusters and its evaluation by automatically generated SAT analogy questions. In *Proceedings of ACL-08: HLT*, pages 692–700, Columbus, Ohio, June. Association for Computational Linguistics.

C. Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

E. Gabrilovich and S. Markovitch. 2007. Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 6–12.

A. Haghighi, A. Ng, and C. Manning. 2005. Robust textual inference via graph matching. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 387–394, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

Marti A. Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, pages 539–545, Morristown, NJ, USA. Association for Computational Linguistics.

Bill Maccartney and Christopher D. Manning. 2008. Modeling semantic containment and exclusion in natural language inference. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 521–528, Manchester, UK, August. Coling 2008 Organizing Committee.

Marius Paşca and Benjamin Van Durme. 2008. Weakly-supervised acquisition of open-domain classes and class attributes from web documents and query logs. In *Proceedings of ACL-08: HLT*, pages 19–27, Columbus, Ohio, June. Association for Computational Linguistics.

Marius Paşca. 2007. Organizing and searching the world wide web of facts – step two: harnessing the wisdom of the crowds. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 101–110, New York, NY, USA. ACM Press.

Patrick Andre Pantel. 2003. *Clustering by committee*. Ph.D. thesis. Adviser-Dekang Lin.

Fernando Pereira. 1993. Distributional clustering of english words. In *In Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 183–190.

Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 622–629, Morristown, NJ, USA. Association for Computational Linguistics.

Alexander E. Richman and Patrick Schone. 2008. Mining wiki resources for multilingual named entity recognition. In *Proceedings of ACL-08: HLT*, pages 1–9, Columbus, Ohio, June. Association for Computational Linguistics.

Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. 2005. Learning syntactic patterns for automatic hypernym discovery. In *Advances in Neural Information Processing Systems (NIPS 2005)*, November. This is a draft version from the NIPS preproceedings; the final version will be published by April 2005.

Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. 2006. Semantic taxonomy induction from heterogenous evidence. In *ACL '06: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 801–808, Morristown, NJ, USA. Association for Computational Linguistics.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A Core of Semantic Knowledge. In *16th international World Wide Web conference (WWW 2007)*, New York, NY, USA. ACM Press.