Unsupervised Resolution of Objects and Relations on the Web

Alexander Yates
Turing Center
Computer Science and Engineering
University of Washington
Box 352350
Seattle, WA 98195, USA
ayates@cs.washington.edu

Oren Etzioni
Turing Center
Computer Science and Engineering
University of Washington
Box 352350
Seattle, WA 98195, USA
etzioni@cs.washington.edu

Abstract

The task of identifying synonymous relations and objects, or Synonym Resolution (SR), is critical for high-quality information extraction. The bulk of previous SR work assumed strong domain knowledge or hand-tagged training examples. This paper investigates SR in the context of unsupervised information extraction, where neither is available. The paper presents a scalable, fully-implemented system for SR that runs in $O(KN \; log \; N)$ time in the number of extractions N and the maximum number of synonyms per word, K. The system, called RESOLVER, introduces a probabilistic relational model for predicting whether two strings are co-referential based on the similarity of the assertions containing them. Given two million assertions extracted from the Web, RESOLVER resolves objects with 78% precision and an estimated 68% recall and resolves relations with 90% precision and 35% recall.

1 Introduction

Web Information Extraction (WIE) systems extract *assertions* that describe a relation and its arguments from Web text (e.g., $(\text{is capital of}, \text{D.C.}, \text{United States})$). WIE systems can extract hundreds of millions of assertions containing millions of different strings from the Web (*e.g.*, the TEXTRUNNER system (Banko et al., 2007)).[1] WIE systems often extract assertions that describe the same real-world object or relation using different names. For example, a WIE system might extract $(\text{is capital city of}, \text{Washington}, \text{U.S.})$, which describes the same relationship as above but contains a different name for the relation and each argument.

Synonyms are prevalent in text, and the Web corpus is no exception. Our data set of two million assertions extracted from a Web crawl contained over a half-dozen different names each for the United States and Washington, D.C., and three for the "is capital of" relation. The top 80 most commonly extracted objects had an average of 2.9 extracted names per entity, and several had as many as 10 names. The top 100 most commonly extracted relations had an average of 4.9 synonyms per relation.

We refer to the problem of identifying synonymous object and relation names as Synonym Resolution (SR).[2] An SR system for WIE takes a set of assertions as input and returns a set of clusters, with each cluster containing coreferential object strings or relation strings. Previous techniques for SR have focused on one particular aspect of the problem, either objects or relations. In addition, the techniques either depend on a large set of training examples, or are tailored to a specific domain by assuming knowledge of the domain's schema. Due to the number and diversity of the relations extracted, these tech-

[1]For a demo see www.cs.washington.edu/research/textrunner.

[2]Ironically, SR has a number of synonyms in the literature, including Entity Resolution, Record Linkage, and Deduplication.

niques are not feasible for WIE systems. Schemata are not available for the Web, and hand-labeling training examples for each relation would require a prohibitive manual effort.

In response, we present RESOLVER, a novel, domain-independent, unsupervised synonym resolution system that applies to both objects and relations. RESOLVER clusters coreferential names together using a probabilistic model informed by string similarity and the similarity of the assertions containing the names. Our contributions are:

1. A scalable clustering algorithm that runs in time $O(KN\ log\ N)$ in the number of extractions N and maximum number of synonyms per word, K, without discarding any potentially matching pair, under exceptionally weak assumptions about the data.
2. An unsupervised probabilistic model for predicting whether two object or relation names co-refer.
3. An empirical demonstration that RESOLVER can resolve objects with 78% precision and 68% recall, and relations with 90% precision and 35% recall.

The next section discusses previous work. Section 3 introduces our probabilistic model for SR. Section 4 describes our clustering algorithm. Section 5 describes extensions to our basic SR system. Section 6 presents our experiments, and section 7 discusses our conclusions and areas for future work.

2 Previous Work

The DIRT algorithm (Lin and Pantel, 2001) addresses a piece of the unsupervised SR problem. DIRT is a heuristic method for finding synonymous relations, or "inference rules." DIRT uses a dependency parser and mutual information statistics over a corpus to identify relations that have similar sets of arguments. In contrast, our algorithm provides a formal probabilistic model that applies equally well to relations and objects, and we provide an evaluation of the algorithm in terms of precision and recall.

There are many unsupervised approaches for object resolution in databases, but unlike our algorithm these approaches depend on a known, fixed schema. Ravikumar and Cohen (Ravikumar and Cohen, 2004) present an unsupervised approach to object resolution using Expectation-Maximization on a hierarchical graphical model. Several other recent approaches leverage domain-specific information and heuristics for object resolution. For example, many (Dong et al., 2005; Bhattacharya and Getoor, 2005; Bhattacharya and Getoor, 2006) rely on evidence from observing which strings appear as arguments to the same relation simultaneously (*e.g.*, co-authors of the same publication). While this is useful information when resolving authors in the citation domain, it is extremely rare to find relations with similar properties in extracted assertions. None of these approaches applies to the problem of resolving relations. See (Winkler, 1999) for a survey of this area.

Several supervised learning techniques make entity resolution decisions (Kehler, 1997; McCallum and Wellner, 2004; Singla and Domingos, 2006), but of course these systems depend on the availability of training data, and often on a significant number of labeled examples per relation of interest. These approaches also depend on complex probabilistic models and learning algorithms, and they have order $O(n^3)$ time complexity, or worse. They currently do not scale to the amounts of data extracted from the Web. Previous systems were tested on at most a few thousand examples, compared with millions or hundreds of millions of extractions from WIE systems such as TEXTRUNNER.

Coreference resolution systems (e.g., (Lappin and Leass, 1994; Ng and Cardie, 2002)), like SR systems, try to merge references to the same object (typically pronouns, but potentially other types of noun phrases). This problem differs from the SR problem in several ways: first, it deals with unstructered text input, possibly with syntactic annotation, rather than relational input. Second, it deals only with resolving objects. Finally, it requires local decisions about strings; that is, the same word may appear twice in a text and refer to two different things, so each occurrence of a word must be treated separately.

The PASCAL Recognising Textual Entailment Challenge proposes the task of recognizing when two sentences entail one another, and many authors have submitted responses to this challenge (Dagan et al., 2006). Synonym resolution is a subtask of this problem. Our task differs significantly from the textual entailment task in that it has no labeled training data, and its input is in the form of relational extractions rather than raw text.

Two probabilistic models for information extraction have a connection with ours. Our probabilistic model is partly inspired by the ball-and-urns abstraction of information extraction presented by Downey et al. (2005) Our task and probability model are different from theirs, but we make many of the same modeling assumptions. Second, we follow Snow et al.'s work (2006) on taxonomy induction in incorporating transitive closure constraints in our probability calculations, as explained below.

3 Probabilistic Model

Our probabilistic model provides a formal, rigorous method for resolving synonyms in the absence of training data. It has two sources of evidence: the similarity of the strings themselves (i.e., edit distance) and the similarity of the assertions they appear in. This second source of evidence is sometimes referred to as "distributional similarity" (Hindle, 1990).

Section 3.2 presents a simple model for predicting whether a pair of strings co-refer based on string similarity. Section 3.3 then presents a model called the Extracted Shared Property (ESP) Model for predicting whether a pair of strings co-refer based on their distributional similarity. Finally, a method is presented for combining these models to come up with an overall prediction for coreference decisions between two clusters of strings.

3.1 Terminology and Notation

We use the following notation to describe the probabilistic models. The input is a data set D containing extracted *assertions* of the form $a = (r, o_1, \dots, o_n)$, where r is a relation string and each o_i is an object string representing the arguments to the relation. In our data, all of the extracted assertions are binary, so $n = 2$. The subset of all assertions in D containing a string s is called D_s.

For strings s_i and s_j, let $R_{i,j}$ be the random variable for the event that s_i and s_j refer to the same entity. Let $R^t_{i,j}$ denote the event that $R_{i,j}$ is true, and $R^f_{i,j}$ denote the event that it is false.

A pair of strings (r, s_2) is called a *property* of a string s_1 if there is an assertion $(r, s_1, s_2) \in D$ or $(r, s_2, s_1) \in D$. A pair of strings (s_1, s_2) is an *instance* of a string r if there is an assertion $(r, s_1, s_2) \in D$. Equivalently, the property $p = (r, s_2)$ *applies* to s_1, and the relation r *applies* to the instance $i = (s_1, s_2)$. Finally, two strings x and y *share* a property (or instance) if both x and y are extracted with the same property (or instance).

3.2 String Similarity Model

Many objects appear with multiple names that are substrings, acronyms, abbreviations, or other simple variations of one another. Thus string similarity can be an important source of evidence for whether two strings co-refer. Our probabilistic String Similarity Model (SSM) assumes a similarity function $\text{sim}(s_1, s_2)$: $STRING \times STRING \rightarrow [0, 1]$. The model sets the probability of s_1 co-referring with s_2 to a smoothed version of the similarity:

$$P(R^t_{i,j}|\text{sim}(s_1, s_2)) = \frac{\alpha * \text{sim}(s_1, s_2) + 1}{\alpha + \beta}$$

The particular choice of α and β make little difference to our results, so long as they are chosen such that the resulting probability can never be one or zero. In our experiments $\alpha = 20$ and $\beta = 5$, and we use the well-known Monge-Elkan string similarity function for objects and the Levenshtein string edit-distance function for relations (Cohen et al., 2003).

3.3 The Extracted Shared Property Model

The Extracted Shared Property (ESP) Model outputs the probability that s_1 and s_2 co-refer based on how many properties (or instances) they share. As an example, consider the strings "Mars" and "Red Planet", which appear in our data 659 and 26 times respectively. Out of these extracted assertions, they share four properties. For example, $(lacks, Mars, ozone\ layer)$ and $(lacks, Red\ Planet, ozone\ layer)$ both appear as assertions in our data. The ESP model determines the probability that "Mars" and "Red Planet" refer to the same entity after observing k, the number of properties that apply to both, n_1, the total number of extracted properties for "Mars", and n_2, the total number of extracted properties for "Red Planet."

ESP models the extraction of assertions as a generative process, much like the URNS model (Downey et al., 2005). For each string s_i, a certain number, P_i, of properties of the string are written on balls and placed in an urn. Extracting n_i assertions that contain s_i amounts to selecting a subset of size n_i from these labeled balls.[3] Properties in the urn are called *potential properties* to distinguish them from extracted properties.

To model coreference decisions, ESP uses a pair of urns, containing P_i and P_j balls respectively, for the two strings s_i and s_j. Some subset of the P_i balls have the exact same labels as an equal-sized subset of the P_j balls. Let the size of this subset be $S_{i,j}$. The ESP model assumes that coreferential strings share as many potential properties as possible, though only a few of the potential properties will be extracted for both. For non-coreferential strings, the number of shared potential properties is a strict subset of the potential properties of each string. Thus if $R_{i,j}$ is true then $S_{i,j} = \min(P_i, P_j)$, and if $R_{i,j}$ is false then $S_{i,j} < \min(P_i, P_j)$.

The ESP model makes several simplifying assumptions in order to make probability predictions. As is suggested by the ball-and-urn abstraction, it assumes that each ball for a string is equally likely to be selected from its urn. Because of data sparsity, almost all properties are very rare, so it would be difficult to get a better estimate for the prior probability of selecting a particular potential property. Second, it assumes that without knowing the value of k, every value of $S_{i,j}$ is equally likely, since we have no better information. Finally, it assumes that all subsets of potential properties are equally likely to be shared by two non-coreferential objects, regardless of the particular labels on the balls, given the size of the shared subset.

Given these assumptions, we can derive an expression for $P(R^t_{i,j})$. First, note that there are $\binom{P_i}{n_i}\binom{P_j}{n_j}$ total ways of extracting n_i and n_j assertions for s_i and s_j. Given a particular value of $S_{i,j}$, the number of ways in which n_i and n_j assertions can be extracted such that they share exactly k is given by

$$\mathrm{Count}(k, n_i, n_j | P_i, P_j, S_{i,j}) =$$

$$\binom{S_{i,j}}{k} \sum_{r,s \geq 0} \binom{S_{i,j}-k}{r+s}\binom{r+s}{r}\binom{P_i - S_{i,j}}{n_i-(k+r)}\binom{P_j - S_{i,j}}{n_j-(k+s)}$$

By our assumptions,

$$P(k|n_i, n_j, P_i, P_j, S_{i,j}) =$$

$$\frac{\mathrm{Count}(k, n_i, n_j | P_i, P_j, S_{i,j})}{\binom{P_i}{n_i}\binom{P_j}{n_j}} \qquad (1)$$

Let $P_{\min} = \min(P_i, P_j)$. The result below follows from Bayes' Rule and our assumptions above:

Proposition 1 *If two strings s_i and s_j have P_i and P_j potential properties (or instances), and they appear in extracted assertions D_i and D_j such that $|D_i| = n_i$ and $|D_j| = n_j$, and they share k extracted properties (or instances), the probability that s_i and s_j co-refer is:*

$$P(R^t_{i,j}|D_i, D_j, P_i, P_j) =$$

$$\frac{P(k|n_i, n_j, P_i, P_j, S_{i,j} = P_{\min})}{\sum_{k \leq S_{i,j} \leq P_{\min}} P(k|n_i, n_j, P_i, P_j, S_{i,j})} \qquad (2)$$

Substituting equation 1 into equation 2 gives us a complete expression for the probability we are looking for.

Note that the probability for $R_{i,j}$ depends on just two hidden parameters, P_i and P_j. Since we have no labeled data to estimate these parameters from, we tie these parameters to the number of times the respective strings s_i and s_j are extracted. Thus we set $P_i = N \times n_i$, and we set $N = 50$ in our experiments.

3.4 Combining the Evidence

For each potential coreference relationship $R_{i,j}$, there are now two pieces of probabilistic evidence. Let $E^e_{i,j}$ be the evidence for ESP, and let $E^s_{i,j}$ be the evidence for SSM. Our method for combining the two uses the Naïve Bayes assumption that each piece of evidence is conditionally independent, given the coreference relation:

$$P(E^s_{i,j}, E^e_{i,j}|R_{i,j}) = P(E^s_{i,j}|R_{i,j})P(E^e_{i,j}|R_{i,j})$$

Given this simplifying assumption, we can combine the evidence to find the probability of a coference relationship by applying Bayes' Rule to both sides (we omit the i, j indices for brevity):

$$P(R^t|E^s, E^e) =$$

$$\frac{P(R^t|E^s)P(R^t|E^e)(1 - P(R^t))}{\sum_{i \in \{t,f\}} P(R^i|E^s)P(R^i|E^e)(1 - P(R^i))}$$

[3]Unlike the URNS model, balls are drawn without replacement because each extracted property is distinct in our data.

3.5 Comparing Clusters of Strings

Our algorithm merges clusters of strings with one another, using one of the above models. However, these models give probabilities for coreference decisions between two individual strings, not two clusters of strings.

We follow the work of Snow et al. (2006) in incorporating transitive closure constraints in probabilistic modeling, and make the same independence assumptions. The benefit of this approach is that the calculation for merging two clusters depends only on coreference decisions between individual strings, which can be calculated independently.

Let a *clustering* be a set of coreference relationships between pairs of strings such that the coreference relationships obey the transitive closure property. We let the probability of a set of assertions D given a clustering C be:

$$P(D|C) = \prod_{R^t_{i,j} \in C} P(D_i \cup D_j | R^t_{i,j}) \times \prod_{R^f_{i,j} \in C} P(D_i \cup D_j | R^f_{i,j})$$

The metric used to determine if two clusters should be merged is the likelihood ratio, or the probability for the set of assertions given the merged clusters over the probability given the original clustering. Let C' be a clustering that differs from C only in that two clusters in C have been merged in C', and let ΔC be the set of coreference relationships in C' that are true, but the corresponding ones in C are false. This metric is given by:

$$P(D|C')/P(D|C) = \frac{\prod_{R^t_{i,j} \in \Delta C} P(R^t_{i,j}|D_i \cup D_j)(1 - P(R^t_{i,j}))}{\prod_{R^t_{i,j} \in \Delta C}(1 - P(R^t_{i,j}|D_i \cup D_j))P(R^t_{i,j})}$$

The probability $P(R^t_{i,j}|D_i \cup D_j)$ may be supplied by the SSM, ESP, or combination model. In our experiments, we let the prior for the SSM model be 0.5. For the ESP and combined models, we set the prior to $P(R^t_{i,j}) = \frac{1}{\min(P_1, P_2)}$.

4 RESOLVER's Clustering Algorithm

Our clustering algorithm iteratively merges clusters of co-referential names, making each iteration in time $O(N\ log\ N)$ in the number of extracted assertions. The algorithm requires only basic assumptions about which strings to compare. Previous work on speeding up clustering algorithms for SR has either required far stronger assumptions, or else it has focused on heuristic methods that remain, in the worst case, $O(N^2)$ in the number of distinct objects.

```
$S$ := set of all strings
For each property or instance $p$,
    $S_p := \{s \in S | s$ has property $p\}$
1. $Scores := \{\}$
2. Build index mapping properties (and instances)
    to strings with those properties (instances)
3. For each property or instance $p$:
    If $|S_p| <$ Max:
        For each pair $\{s1, s2\} \subset S_p$:
            Add mergeScore($s1, s2$) to $Scores$
4. Repeat until no merges can be performed:
    Sort $Scores$
    $UsedClusters := \{\}$
    While score of top clusters $c_1, c_2$
      is above Threshold:
        Skip if either is in $UsedClusters$
        Merge $c_1$ and $c_2$
        Add $c_1, c_2$ to $UsedClusters$
        Merge properties containing $c_1, c_2$
    Recalculate merge scores as in Steps 1-3
```

Figure 1: RESOLVER's Clustering Algorithm

Our algorithm, a greedy agglomerative clustering method, is outlined in Figure 1. The first novel part of the algorithm, step 3, compares pairs of strings that share the same property or instance, so long as no more than Max strings share that same property or instance. After the scores for all comparisons are made, each string is assigned its own cluster. Then the scores are sorted and the best cluster pairs are merged until no pair of clusters has a score above threshold. The second novel aspect of this algorithm is that as it merges clusters in Step 4, it merges properties containing those clusters in a process we call *mutual recursion*, which is discussed below.

This algorithm compares every pair of clusters that have the potential to be merged, assuming two properties of the data. First, it assumes that pairs of clusters with no shared properties are not worth comparing. Since the number of shared properties is a key source of evidence for our approach, these clusters almost certainly will not be merged, even if they are compared, so the assumption is quite reasonable. Second, the approach assumes that clusters sharing only properties that apply to very many strings (more than Max) need not be compared. Since properties shared by many strings provide little evidence that the strings are coreferential, this assumption is reasonable for SR. We use $Max = 50$ in our experiments. Less than 0.1% of the properties are thrown out using this cutoff.

4.1 Algorithm Analysis

Let D be the set of extracted assertions. The following analysis shows that one iteration of merges takes time $O(N\ log\ N)$, where $N = |D|$. Let NC be the number of comparisons between strings in step 3. To simplify the analysis, we consider only those properties that contain a relation string and an argument 1 string. Let A be the set of all such properties. NC is linear in N:[4]

$$\begin{aligned} NC &= \sum_{p \in A} \frac{|S_p| \times (|S_p| - 1)}{2} \\ &\leq \frac{(Max - 1)}{2} \times \sum_{p \in A} |S_p| \\ &= \frac{(Max - 1)}{2} \times N \end{aligned}$$

Note that this bound is quite loose because most properties apply to only a few strings. Step 4 requires time $O(N\ log\ N)$ to sort the comparison scores and perform one iteration of merges. If the largest cluster has size K, in the worst case the algorithm will take K iterations. In our experiments, the algorithm never took more than 9 iterations.

4.2 Relation to other speed-up techniques

The merge/purge algorithm (Hernandez and Stolfo, 1995) assumes the existence of a particular attribute such that when the data set is sorted on this attribute, matching pairs will all appear within a narrow window of one another. This algorithm is $O(M\ log\ M)$ where M is the number of distinct strings. However, there is no attribute or set of attributes that comes close to satisfying this assumption in the context of domain-independent information extraction.

There are several techniques that often provide speed-ups in practice, but in the worst case they make $O(M^2)$ comparisons at each merge iteration, where M is the number of distinct strings. This can cause problems on very large data sets. Notably, McCallum et al. (2000) use a cheap comparison metric to place objects into overlapping "canopies," and then use a more expensive metric to cluster objects appearing in the same canopy. The RESOLVER clustering algorithm is in fact an adaptation of the canopy method; it adds the restriction that strings are not compared when they share only high-frequency properties. The canopy method works well on high-dimensional data with many clusters, which is the case with our problem, but its time complexity is worse than ours.

For information extraction data, a complexity of $O(M^2)$ in the number of distinct strings turns out to be considerably worse than our algorithm's complexity of $O(N\ log\ N)$ in the number of extracted assertions. This is because the data obeys a Zipf law relationship between the frequency of a string and its rank, so the number of distinct strings grows linearly or almost linearly with the number of assertions.[5]

4.3 Mutual Recursion

Mutual recursion refers to the novel property of our algorithm that as it clusters relation strings together into sets of synonyms, it collapses properties together for object strings and potentially finds more shared properties between coreferential object strings. Likewise, as it clusters objects together into sets of coreferential names, it collapses instances of relations together and potentially finds more shared instances between coreferential relations. Thus the clustering decisions for relations and objects mutually depend on one another.

For example, the strings "Kennedy" and "President Kennedy" appear in 430 and 97 assertions in our data, respectively, but none of their extracted properties match exactly. Many properties,

[4]If the Max parameter is allowed to vary with $log|D|$, rather than remaining constant, the same analysis leads to a slightly looser bound that is still better than $O(N^2)$.

[5]The exact relationship depends on the shape parameter z of the Zipf curve. If $z < 1$, as it is for our data set, the number of total extractions grows linearly with the number of distinct strings extracted. If $z = 1$, then n extractions will contain $\Omega(\frac{n}{\ln n})$ distinct strings.

however, *almost* match. For example, the assertions $\langle$challenged, Kennedy, Premier Krushchev$\rangle$ and $\langle$stood up to, President Kennedy, Kruschev$\rangle$ both appear in our data. Because "challenged" and "stood up to" are similar, and "Krushchev" and "Premier Krushchev" are similar, our algorithm is able to merge these pairs into two clusters, thereby creating a new shared property between "Kennedy" and "President Kennedy." Eventually it can merge these two strings as well.

5 Extensions to RESOLVER

While the basic RESOLVER system can cluster synonyms accurately and quickly, there is one type of error that it frequently makes. In some cases, it has difficulty distinguishing between similar pairs of objects and *identical* pairs. For example, "Virginia" and "West Virginia" share several extractions because they have the same type, and they have high string similarity. As a result, RESOLVER clusters these two together. The next two sections describe two extensions to RESOLVER that address the problem of similarity vs. identity.

5.1 Function Filtering

RESOLVER can use functions and one-to-one relations to help distinguish between similar and identical pairs. For example, West Virginia and Virginia have different capitals: Richmond and Charleston, respectively. If both of these facts are extracted, and if RESOLVER knows that the "capital of" relation is functional, it should prevent Virginia and West Virginia from merging.

The Function Filter prevents merges between strings that have different values for the same function. More precisely, it decides that two strings y_1 and y_2 *match* if their string similarity is above a high threshold. It prevents a merge between strings x_1 and x_2 if there exist a function f and extractions $f(x_1, y_1)$ and $f(x_2, y_2)$, and there are no such extractions such that y_1 and y_2 match (and *vice versa* for one-to-one relations). Experiments described in section 6 show that the Function Filter can improve the precision of RESOLVER without significantly affecting its recall.

While the Function Filter currently uses functions and one-to-one relations as negative evidence, it is also possible to use them as positive evidence. For example, the relation "married" is not strictly one-to-one, but for most people the set of spouses is very small. If a pair of strings are extracted with the same spouse—*e.g.*, "FDR" and "President Roosevelt" share the property ("married", "Eleanor Roosevelt")—this is far stronger evidence that the two strings are identical than if they shared some random property.

Unfortunately, various techniques that attempted to model this insight, including a TF-IDF weighting of properties, yielded essentially no improvement of RESOLVER. One major reason is that there are relatively few examples of shared functional or one-to-one properties because of sparsity. This idea deserves more investigation, however, and is an area for future work.

5.2 Using Web Hitcounts

While names for two similar objects may often appear together in the same sentence, it is relatively rare for two different names of the same object to appear in the same sentence. RESOLVER exploits this fact by querying the Web to determine how often a pair of strings appears together in a large corpus. When the hitcount is high, RESOLVER can prevent the merge.

Specifically, the Coordination-Phrase Filter searches for hitcounts of the phrase "s_1 and s_2", where s_1 and s_2 are a candidate pair for merging. It then computes a variant of pointwise mutual information, given by

$$\text{coordination score}(s_1, s_2) = \frac{\text{hits}(s_1 \text{ and } s_2)^2}{\text{hits}(s_1) \times \text{hits}(s_2)}$$

The filter prevents any merge for which the coordination score is above a threshold, which is determined on a development set. The results of Coordination-Phrase filtering are discussed in the next section.

6 Experiments

Our experiments demonstrate that the ESP model is significantly better at resolving synonyms than a widely-used distributional similarity metric, the cosine similarity metric (CSM) (Salton and McGill, 1983), and that RESOLVER is significantly better at resolving synonyms than either of its components, SSM or ESP.

We test these models on a data set of 2.1 million assertions extracted from a Web crawl.[6] All models ran over all assertions, but compared only those objects or relations that appeared at least 25 times in the data, to give the ESP and CSM models sufficient data for estimating similarity. However, the models do use strings that appear less than 25 times as features. In all, the data contains 9,797 distinct object strings and 10,151 distinct relation strings that appear at least 25 times.

We judged the precision of each model by manually labeling all of the clusters that each model outputs. Judging recall would require inspecting not just the clusters that the system outputs, but the entire data set, to find all of the true clusters. Because of the size of the data set, we instead estimated recall over a smaller subset of the data. We took the top 200 most frequent object strings and top 200 most frequent relation strings in the data. For each one of these high-frequency strings, we manually searched through all strings with frequency over 25 that shared at least one property, as well as all strings that contained one of the keywords in the high-frequency strings or obvious variations of them. We manually clustered the resulting matches. The top 200 object strings formed 51 clusters of size greater than one, with an average cluster size of 2.9. For relations, the top 200 strings and their matches formed 110 clusters with size greater than one, with an average cluster size of 4.9. We measured the recall of our models by comparing the set of all clusters containing at least one of the high-frequency words against these gold standard clusters.

For our precision and recall measures, we only compare clusters of size two or more, in order to focus on the interesting cases. Using the term *hypothesis cluster* for clusters created by one of the models, we define the precision of a model to be the number of elements in all hypothesis clusters which are correct divided by the total number of elements in hypothesis clusters. An element s is marked correct if a plurality of the elements in s's cluster refer to the same entity as s; we break ties arbitrarily, as they do not affect results. We define recall as the sum over gold standard clusters of the most number of elements found in a single hypothesis cluster, divided by the total number of elements in gold standard clusters.

[6]The data is made available at http://www.cs.washington.edu/homes/ayates/.

For the ESP and SSM models in our experiment, we prevented mutual recursion by clustering relations and objects separately. Only the full RESOLVER system uses mutual recursion. For the CSM model, we create for each distinct string a row vector, with each column representing a property. If that property applies to the string, we set the value of that column to the inverse frequency of the property and zero otherwise. CSM finds the cosine of the angle between the vectors for each pair of strings, and merges the best pairs that score above threshold.

Each model requires a threshold parameter to determine which scores are suitable for merging. For these experiments we arbitrarily chose a threshold of 3 for the ESP model (that is, the data needs to be 3 times more likely given the merged cluster than the unmerged clusters in order to perform the merge) and chose thresholds for the other models by hand so that the difference between them and ESP would be roughly even between precision and recall, although for relations it was harder to improve the recall. It is an important item for future work to be able to estimate these thresholds and perhaps other parameters of our models from unlabeled data, but the chosen parameters worked well enough for the experiments. Table 1 shows the precision and recall of our models.

6.1 Discussion

ESP significantly outperforms CSM on both object and relation clustering. CSM had particular trouble with lower-frequency strings, judging far too many of them to be co-referential on too little evidence. If the threshold for clustering using CSM is increased, however, the recall begins to approach zero.

ESP and CSM make predictions based on a very noisy signal. "Canada," for example, shares more properties with "United States" in our data than "U.S." does, even though "Canada" appears less often than "U.S." The results show that both models perform below the SSM model on its own for object merging, and both perform slightly better than SSM on relations because of SSM's poor recall.

We found a significant improvement in both pre-

	Objects			Relations		
Model	Prec.	Rec.	F1	Prec.	Rec.	F1
CSM	0.51	0.36	0.42	0.62	0.29	0.40
ESP	**0.56**	0.41	0.47	**0.79**	0.33	0.47
SSM	**0.62**	0.53	0.57	**0.85**	0.25	0.39
RESOLVER	**0.71**	0.66	0.68	**0.90**	**0.35**	0.50

Table 1: **Comparison of the cosine similarity metric (CSM), RESOLVER components (SSM and ESP), and the RESOLVER system.** Bold indicates the score is significantly different from the score in the row above at $p < 0.05$ using the chi-squared test with one degree of freedom. Using the same test, RESOLVER is also significantly different from ESP and CSM in recall on objects, and from CSM and SSM in recall on relations. RESOLVER's F1 on objects is a 19% increase over SSM's F1. RESOLVER's F1 on relations is a 28% increase over SSM's F1.

cision and recall when using a combined model over using SSM alone. RESOLVER's F1 is 19% higher than SSM's on objects, and 28% higher on relations.

In a separate experiment we found that mutual recursion provides mixed results. A combination of SSM and ESP without mutual recursion had a precision of 0.76 and recall of 0.59 on objects, and a precision of 0.91 and recall of 0.35 on relations. Mutual recursion increased recall and decreased precision for both objects and relations. None of the differences were statistically significant, however.

There is clearly room for improvement on the SR task. Except for the problem of confusing similar and identical pairs (see section 5), error analysis shows that most of RESOLVER's mistakes are because of two kinds of errors:

1. *Extraction errors*. For example, "US News" gets extracted separately from "World Report", and then RESOLVER clusters them together because they share almost all of the same properties.

2. *Multiple word senses*. For example, there are two President Bushes; also, there are many terms like "President" and "Army" that can refer to many different entities.

6.2 Experiments with Extensions

The extensions to RESOLVER attempt to address the confusion between similar and identical pairs. Experiments with the extensions, using the same datasets and metrics as above, demonstrate that the Function Filter (FF) and the Coordination-Phrase Filter (CPF) boost RESOLVER's performance.

FF requires as input the set of functional and one-to-one relations in the data. Table 2 contains a sam-

is capital of	is capital city of
named after	was named after
headquartered in	is headquartered in

Table 2: **A sample of the set of functions used by the Function Filter.**

Model	Prec.	Rec.	F1
RESOLVER	0.71	0.66	0.68
RESOLVER+FF	0.74	0.66	0.70
RESOLVER+CPF	**0.78**	0.68	0.73
RESOLVER+FF+CPF	**0.78**	0.68	0.73

Table 3: **Comparison of object merging results for the RESOLVER system, RESOLVER plus Function Filtering (RESOLVER+FF), RESOLVER plus Coordination-Phrase Filtering (RESOLVER+CPF), and RESOLVER plus both types of filtering (RESOLVER+FF+CPF).** Bold indicates the score is significantly different from RESOLVER's score at $p < 0.05$ using the chi-squared test with one degree of freedom. RESOLVER+CPF's F1 on objects is a 28% increase over SSM's F1, and a 7% increase over RESOLVER's F1.

pling of the manually-selected functions used in our experiment. Automatically discovering such functions from extractions has been addressed in Ana-Maria Popescu's dissertation (Popescu, 2007), and we did not attempt to duplicate this effort in RESOLVER.

Table 3 contains the results of our experiments. With coordination-phrase filtering, RESOLVER's F1 is 28% higher than SSM's on objects, and 6% higher than RESOLVER's F1 without filtering. While function filtering is a promising idea, FF provides a smaller benefit than CPF on this dataset, and the merges that it prevents are, with a few exceptions, a subset of the merges prevented by CPF. This is in part due to the limited number of functions available in the data. In addition to outperforming FF on this dataset, CPF has the added advantage that it does not require additional input, like a set of functions.

7 Conclusion and Future Work

We have shown that the unsupervised and scalable RESOLVER system is able to find clusters of co-referential object names in extracted relations with a precision of 78% and a recall of 68% with the aid of coordination-phrase filtering, and can find clusters of co-referential relation names with precision of 90% and recall of 35%. We have demonstrated significant improvements over using simple similarity metrics for this task by employing a novel probabilistic model of coreference.

In future work, we plan to use RESOLVER on a much larger data set of over a hundred million assertions, further testing its scalability and its ability to improve in accuracy given additional data. We also plan to add techniques for handling multiple word senses. Finally, to make the probabilistic model more accurate and easier to use, we plan to investigate methods for automatically estimating its parameters from unlabeled data.

Acknowledgements

This research was supported in part by NSF grants IIS-0535284 and IIS-0312988, DARPA contract NBCHD030010, ONR grant N00014-05-1-0185 as well as gifts from Google, and carried out at the University of Washington's Turing Center. We thank Doug Downey, Michele Banko, Stef Schoenmackers, Dan Weld, Fei Wu, and the anonymous reviewers for their helpful comments on previous drafts.

References

M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. 2007. Open information extraction from the web. In *IJCAI*.

I. Bhattacharya and L. Getoor. 2005. Relational Clustering for Multi-type Entity Resolution. In *11th ACM SIGKDD Workshop on Multi Relational Data Mining*.

I. Bhattacharya and L. Getoor. 2006. Query-time entity resolution. In *KDD*.

W.W. Cohen, P. Ravikumar, and S.E. Fienberg. 2003. A comparison of string distance metrics for name-matching tasks. In *IIWeb*.

I. Dagan, O. Glickman, and B. Magnini. 2006. The PASCAL Recognising Textual Entailment Challenge. *Lecture Notes in Computer Science*, 3944:177–190.

X. Dong, A.Y. Halevy, and J. Madhavan. 2005. Reference reconciliation in complex information spaces. In *SIGMOD*.

D. Downey, O. Etzioni, and S. Soderland. 2005. A Probabilistic Model of Redundancy in Information Extraction. In *IJCAI*.

M. A. Hernandez and S. J. Stolfo. 1995. The merge/purge problem for large databases. In *SIGMOD*.

D. Hindle. 1990. Noun classification from predicage-argument structures. In *ACL*.

A. Kehler. 1997. Probabilistic coreference in information extraction. In *EMNLP*.

S. Lappin and H. J. Leass. 1994. An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4):535–561.

D. Lin and P. Pantel. 2001. DIRT – Discovery of Inference Rules from Text. In *KDD*.

A. McCallum and B. Wellner. 2004. Conditional models of identity uncertainty with application to noun coreference. In *NIPS*.

A. McCallum, K. Nigam, and L. Ungar. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*.

V. Ng and C. Cardie. 2002. Improving machine learning approaches to coreference resolution. In *ACL*.

Ana-Maria Popescu. 2007. *Information Extraction from Unstructured Web Text*. University of Washington.

P. Ravikumar and W. W. Cohen. 2004. A hierarchical graphical model for record linkage. In *UAI*.

G. Salton and M.J. McGill. 1983. *Introduction to Modern Information Retrieval*. McGraw-Hill.

P. Singla and P. Domingos. 2006. Entity Resolution with Markov Logic. In *ICDM*.

R. Snow, D. Jurafsky, and A. Y. Ng. 2006. Semantic taxonomy induction from heterogenous evidence. In *COLING/ACL*.

W.E. Winkler. 1999. The state of record linkage and current research problems. Technical report, U.S. Bureau of the Census, Washington, D.C.