

# Entity Resolution, Information Integration, CSCI 6967-01

## 1 Entity Resolution

Entity resolution refers to the problem of checking whether two references refer to the same entity or not. This is a very hard problem and used in many different contexts, also called data cleaning, deduplication, etc. We will only consider resolution of references that look similar but may have errors or missing information. There are two associated problems:

- references that look different may refer to the same entity
- references that look the same may refer to different entities

So, to check whether two references refer to the same entity or not, we are going to look at some basic pieces of information:

- syntactic similarity of their attributes (i.e. names are the same, emails are different)
- constraints associated with the underlying data model (users with the same email are the same people, two authors of a paper have to be different)
- relationships between entities through shared properties (person x and x' wrote many papers with persons y and z, so x and x' might be same person)

So, we need to create a method that can incorporate these types of information.

### 1.1 Matching entities based on attributes

When matching string literals, we need to find a measure that tells us how similar they are. Some possibilities are based on the number of characters that match and their locations for single words (jaro, jaro-winkler) and measures based on the overall frequency of words in general for phrases (tf-idf that takes into account the number of times a word appears in the given string [tf] and the number of times it appears in the database [1/idf]).

These measures give us a similarity measure between two strings of a specific type. In many cases, this is not sufficient to conclude two entities are the same. However, we generally use rules about the domain to conclude two entities are the same based on the similarity between their specific attributes:

- Similarity attributes: the more similar names, addresses are the more we believe that the entities are the same
- Boolean attributes: if certain attributes are the very similar or the same, then we conclude that the entities are the same (furthermore this may imply certain other attributes should be the same)

This approach is not sufficient to conclude that two entities with the same name are in fact different. This can only be done using relationships between entities.

## 1.2 Matching entities based on relationships

The relationships serve as the enrichment of the attribute based matching. If two entities are the same, then one expects that they share lots of common relationships. For example, people write multiple papers with the same coauthors, people who work in the same place usually collaborate with each other, etc.

We would like to use this information to conclude matches between entities. There are possible options to include the relationship information into the matches: (1) treat relationship similarity as enforcing information for the matches between items, (2) treat relationship as the main way to partition entities.

One of the main problems that must be solved in this case is how to model the similarity of relationships and whether this similarity will involve attribute level similarity.

**Clustering approach** Treat identical entities as a cluster. The relationships define adjacency graphs for each entity reference. Hence, the other entities (clusters) are considered adjacent to a cluster (or neighbors) with respect to these relationships. We can now check whether two entity clusters are similar by looking at whether their neighbors are very similar. Some possible ways to define whether two sets are similar:

- number of common elements between them,
- number of common elements divided by the total number of elements in both clusters (so that large sets are not favored by default),
- for each common element  $z$ , add up  $1/\log(\text{frequency}(z))$  where frequency tells us how many other elements link to  $z$  (elements with few links are more significant),
- compute uniqueness of each element to be  $1/\log(\text{frequency}(z))$  and then find the ratio of  $A/B$  where  $A$  is the sum of uniqueness for common elements between the two sets and  $B$  is sum of uniqueness for all elements in both sets.

Now, at each step we merge the two clusters that are the most similar to each other based on one of the above measures. We stop if the similarity of the two most similar classes is below a certain threshold. The problems with this approach are: we do not consider the attribute similarity after an initial step. Secondly, the algorithm requires an initial step to find first set of matches using only attribute based similarity. Errors in this step will propagate to all the other steps. Finally, we need to determine the correct thresholds for each application.

**Brute force approach** Consider all pairs of entities in the database. Anytime a match between two attributes or entities are found, then propagate them to the related entities through the relationships. The problem with this approach is that it requires too much space. The propagation algorithm is not based on any specific metric but system determined scaling factors.

**Relationship paths approach** Find the connection strength between a pair of entities from the graph of all relationships which also includes the possible similarity between entities.

First we assign types to edges based on the relationship that causes the entities to be linked. Then, we assign weights to the edges. These weights could be given by a domain specialized or learnt from the system. Intuitively, rare edge types should be given more weight than the others. The idea is to disregard associations between entities that may be accidental.

We consider all paths of length at most  $L$  between two entities  $u, v$  and for each path, find the strength of the connection by multiplying the weight of all the edges in that path. The strength is now the sum of the strength of all the paths.

Now, suppose we introduce edges to the relationship graph between entities  $e_1$  and  $e_2$  if these entities are similar due to their attributes. We also introduce a weight for these edges that is equivalent to the similarity between the nodes. We can incorporate these edges to the set of possible paths in the usual way. If we do not want the similarity measures to have a very dominant role in determining the path strength, then we can scale them down with a factor  $\lambda$  that is system determined.

Compute a new graph where each node is an entity and the edges between nodes have weight equal to connection strength. Then, find a partition (cut) such that the edges that cross the partition have the smallest total weight. Then, the partitions are not going to separate two records that correspond to the same entities. The problem with this approach is that we need to know the correct number of clusters. Furthermore, the connection strength is computed once. Once we find two entities are determined to be the same, we do not propagate this information to increase the connection strength of others.

Note that the correct approach may integrate all these methods to a certain degree. However, as in all clustering based approaches, there are always some parameters that need to feed into these methods (such as number of entities or the thresholds).

## 2 Computing ER efficiently

Regardless of the method used for computing ER, there are two basic operations involved:

- figuring out when two entities are the same ( $r_1 \approx r_2$ ) called the comparison operator,
- combining the representations of the matched entities to find a new representation for the entity  $\langle r_1, r_2 \rangle$  called the merge operator.

Note that the comparison operation  $r_1 \approx r_2$  is the most complex operator. The main idea is to reduce the number of times this operation is called.

The ER operation finds all possible matches between entities  $r_1 \approx r_2$ , adds the new matched entities  $\langle r_1, r_2 \rangle$  to the set of all entities and then removes all entities implied by the rest. The following algorithm performs the same operation but with the smallest number of similarity operations:

Algorithm G-SWOOSH:

```

1: Input: a set of  $I$  records
2: Output: a set of  $I' = ER(I)$  records
3:  $I' \leftarrow \emptyset$ 
4: while  $I \neq \emptyset$  do
5:    $r \leftarrow$  a record from  $I$ 
6:   remove  $r$  from  $I$ 
7:   for records  $r'$  in  $I' \cup \{r\}$  do
8:     if  $r \approx r'$  (or  $r' \approx r$ ) then
9:        $merged \leftarrow \langle r, r' \rangle$  (or  $\langle r, r' \rangle$ )
10:    if  $merged \notin I \cup I' \cup \{r\}$  then
11:      add merged to  $I$ 
12:   add  $r$  to  $I'$ 
13: remove dominated (redundant) records from  $I'$ 
14: return  $I'$ 

```

However, it is possible to reduce the complexity of this algorithm by reducing the number of comparisons ( $r' \approx r$ ) if the matching and comparison algorithms satisfy a number of conditions.

- Idempotence:  $\forall r, r \approx r$  and  $\langle r, r \rangle = r$ .
- Commutativity:  $\forall r_1, r_2, r_1 \approx r_2$  iff  $r_2 \approx r_1$  and if  $r_1 \approx r_2$  then  $\langle r_1, r_2 \rangle = \langle r_2, r_1 \rangle$ .
- Associativity:  $\forall r_1, r_2, r_3$  such that  $\langle r_1, \langle r_2, r_3 \rangle \rangle$  and  $\langle \langle r_1, r_2 \rangle, r_3 \rangle$  exist,  $\langle r_1, \langle r_2, r_3 \rangle \rangle = \langle \langle r_1, r_2 \rangle, r_3 \rangle$ .
- Representativity: If  $r_3 = \langle r_1, r_2 \rangle$  then for any  $r_4$  such that  $r_1 \approx r_4$ , we also have  $r_3 \approx r_4$ .

Given two records  $r_1$  and  $r_2$ , we say that  $r_1$  is merge dominated by  $r_2$ , denoted by  $r_1 \leq r_2$  if  $r_1 \approx r_2$  and  $r_2 = \langle r_1, r_2 \rangle$ .

We can see that this ordering specifies the following monotonicity properties:

- for any records  $r_1, r_2$  such that  $r_1 \approx r_2$ , it holds that  $r_1 \leq \langle r_1, r_2 \rangle$  and  $r_2 \leq \langle r_1, r_2 \rangle$ .
- if  $r_1 \leq r_2$  and  $r_1 \approx r$  then  $r_2 \approx r$ .
- if  $r_1 \leq r_2$  and  $r_1 \approx r$  then  $\langle r_1, r \rangle \leq \langle r_2, r \rangle$ .
- if  $r_1 \leq s$   $r_2 \leq s$  and  $r_1 \approx r_2$ , then  $\langle r_1, r_2 \rangle \leq s$ .

Given this ordering, we can order two database of records such that  $I_1 \leq I_2$  iff for all  $r_1 \in I_1$  there exists an  $r_2 \in I_2$  such that  $r_1 \leq r_2$ .

We define a derivation step for entity resolution for a database of records  $I$  that computes a new database  $I'$  to be one of the following two steps:

- Merge step: Given two records  $r_1, r_2 \in I$  such that  $r_1 \approx r_2$  and  $r_3 = \langle r_1, r_2 \rangle \notin I$ , then  $I' = I \cup \{r_3\}$ .
- Purge step: Given two  $r_1, r_2 \in I$  such that  $r_1 \leq r_2$ , then  $I' = I - \{r_1\}$ .

We denote this derivation as  $I \rightarrow I'$ .

Then, we can define entity resolution to be the set  $I'$  that is obtained by applying all possible derivation steps starting with database  $I$ . Now the algorithm G-SWOOSH above computes this closure in the general case. However, if the above conditions for the merge and comparison operators hold, we can define the following algorithm to find the entity resolution more efficiently. This algorithm is given below.

Algorithm R-SWOOSH:

```
1: Input: a set of  $I$  records
2: Output: a set of  $I' = ER(I)$  records
3:  $I' \leftarrow \emptyset$ 
4: while  $I \neq \emptyset$  do
5:    $r \leftarrow$  a record from  $I$ 
6:   remove  $r$  from  $I$ 
7:    $buddy \leftarrow null$ 
8:   for records  $r'$  in  $I'$  do
9:     if  $r \approx r'$  then
10:       $buddy \leftarrow r'$ 
11:     exitfor
12:   if  $buddy = null$  then
13:     add  $r$  to  $I'$ 
14:   else
15:      $r'' \leftarrow \langle r, buddy \rangle$ 
16:     remove  $buddy$  from  $I'$ 
17:     add  $r''$  to  $I$ 
18: return  $I'$ 
```

The main feature of this algorithm is the fact that we need to find the first match for each record  $r$  we are investigating. As soon as we match  $r$  with  $r'$ , we remove both  $r, r'$  and replace them with  $\langle r, r' \rangle$  which reduces the number of comparisons the algorithm will make considerably.