DataMade

Modern Approaches to Schema Matching

Date: January 30, 2017
By: Forest Gregg and Peter Xu, DataMade
2543 North Spaulding Ave, #2, Chicago, IL 60647
(312) 725-0195 | info@datamade.us

Introduction

With large collections of data, an analyst often struggles to discover all the data that is relevant to her question. For her to do her work, the analyst needs to know which datasets could be about her subject and what the fields in those datasets might mean. *Schema matching* attempts to carry out these two task in an automated or semi-automated way.

More precisely, schema matching addresses two questions.

- Which datasets are likely to contain information about the same type of entity?
- Which fields in separate datasets are likely to contain comparable information about an entity?

Either problem is difficult by itself, but schema matching is made even trickier because these questions are entangled. For example, in order to guess that two columns, from separate tables, are both referring to a date of birth, we might need know that the rows in each table are referring to people. Conversely, if we knew that two tables each had a field that represented a date of birth, we could likely guess that the rows are about people.

The Schema Matching Literature

In the early 2000s, schema matching was an active topic of academic research. The research was largely motivated by the promise and problems of the "Semantic Web," which envisioned a near-term future where many facts about the world were represented in semantically rich machine readable formats. For example, some information about the Empire State Building could be represented as:

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Place",
  "geo": {
    "@type": "GeoCoordinates",
    "latitude": "40.75",
    "longitude": "73.98"
  },
```

```
  "name": "Empire State Building"
}
</script>
```

Where all the meaning of `@context`, `@type`, `geo`, and `name` are defined more or less precisely at `http://schema.org/Place`. If a sizable fraction of human knowledge was represented in this way, then machines could begin to deduce new and interesting things through syllogistic reasoning. For example, if hotels published semantic information about their rooms and restaurants published their menus semantically, then computers could help find the best priced room that's close to many restaurants, without either hotels, restaurants, even being aware of each other.

The Semantic Web quickly ran into two problems. First, it's ends up being very, very difficult to adequately represent any interesting domain of human action in a way that is both unambiguous and that supports many different uses and cases. Publishers of semantic data tended to invent their own, idiosyncratic way of representing the data that worked for them.

Research in schema matching was focused on harmonizing these idiosyncratic representations. Work progressed for a few years until the second main problem of the Semantic Web became apparent. Most publishers of information never found a compelling reason to publish data in any of these formats. Active research fell off sharply by 2005.

Because of this history, the academic literature has two features that limit its use for a next generation data warehouse like CARRA. First, these literature generally assumes that data is represented using controlled, semantically rich, and hierarchical data formats. In much of the data that CARRA will work with, the data is represented in flat tables, column names were chosen idiosyncratically, and there is no codebook let alone a machine readable description of what each column means.

Second, the methods described in this literature are simplistic. The field only had a few years to develop and the active period occurred about five years before machine learning techniques were commonly being used throughout computer science.

In this memo, we review that literature and highlight opportunities to advance upon the ideas. We also will evaluate a number of open source toolkits. The COMA matcher [3] will be referred to often, as it is includes most of of the standard schema matching techniques.

Comparing Columns

Schema matching starts with trying to identify columns that contain the same type of information. Most existing schema matchers do this by computing a number of different distance measures for each possible pair of columns and then applying some rule to aggregate these into a single score for each column pair.

Distance Measures

The distance measures used in schema matching typically either compare the names of columns or the contents of columns.

String Distance

Column names can be compared by calculating the distance between the strings of the column names. The simplest measure is an exact match, which is 0 if the strings are exactly the same and 1 if not.

Beyond that, there are two broad ways of comparing the distance between strings. The first is to calculate how many insertions, deletions, or substitutions are required to turn one string into another. For example, `SCHEMA` and `SCEMA` has an distance of one, because you can transform one string into another with either a deletion or an insertion.

Alternately, a string can be treated as a collection of characters and we can calculate the overlap between the collections. A simple measure of this type is the Jaccard distance

$$\mathrm{J}(A,B) = 1 - \frac{\text{\# Unique Letters Common to A and B}}{(\text{\# Unique Letters in A}) + (\text{\# Unique Letters in B}) - (\text{\# Unique Letters Common to A and B})}$$

so that `SCHEMA` and `SCEMA` have a Jaccard distance of 0.167.

Both approaches have many, many variations, and existing toolkits usually provide at least four or five to choose from.

Semantic Distance

If the computers already knew the meaning of column names, then the problem of matching columns would be trivial to solve. If columns names are synonyms, there should be little distance between them. If both column names are examples of common class, for example `HOURLY WAGE` and `ANNUAL SALARY`, we would want to score those as similar, although maybe not as similar as synonyms.

In most schema matching toolkits, the semantic distance between column names is calculated with the aid of dictionaries that list both synonyms and the class that the word is an example of. Typically, the semantic distance between two columns names is calculated in the following way.

1. If the words are exactly the same, then the distance is 0.
2. If the words are synonyms then, the the distance is d_1.
3. If the words are examples of the same class then the distance is d_2.
4. If one word is an example of the second word then the distance is d_3. For example, `HOURLY WAGE` and `COMPENSATION`.

The setting of the distances d_1, d_2, and d_3 is typically arbitrary, with d_3 greater than d_2 and d_2 greater than d_1.

The main drawback to the dictionary based approach is that it requires a domain specific taxonomic dictionary, which is usually laborious to construct. Available, general purpose taxonomic dictionaries, like WordNet,[9] do not usually help much in schema matching.

A naive implementation of the dictionary approach will also not be robust to misspellings or column name abbreviations. This can be addressed with algorithms for approximate string matching used in spelling

correction algorithms.

Recent developments in natural language processing suggests a promising alternative for calculating the semantic distances. The word2vec model has shown itself to be surprisingly good model for measuring the "semantic" distance between words, and could likely be usefully applied for schema matching. This technique requires a large corpus of tables, but does not require a handmade taxonomic dictionary.[10]

Data Types

If we have access to the database system, we can extract metadata on the types of data that can occur in the columns of table. For example, we can see whether a column contains integers, floating point numbers, text, dates, or timestamps. When we compare table columns, we can compare data types and score columns with the same data types as similar.

Unfortunately, data often does not have the most specific data type possible in a database. Timestamps can be represented in an integer field, integers can be represented in floating point number fields, and everything can be represented as text. The distance between timestamp columns should be closer than between text columns. A robust system will attempt to infer the most specific data type possible for a column.

Inferring Semantic Meaning

It is often possible to infer the type of information contained in the column by examining the data in the column. One method, described in [8], is to develop a library of common formats (e.g. different ways to write dates/times, email addresses, zip codes, etc.) described by regular expressions. If columns match the same formats, then can score them as similar.

This idea could be made more robust by using machine learning technique for text classification, instead of hand coded, brittle rules.

Distributional Distance

If two columns contain the same type of information about the same type of entities, they will often have similar distributions of data. For example, the distribution of birth dates of unemployment recipients will be more similar to the distribution of birth dates of high school freshman than the distribution of application dates for business licenses, because there will be very few weekends in the application dates.

We will often usefully compare distributions by comparing summary statistics. If we have continuous data, these will likely include

- Range of Values
- Average or Median
- Standard Deviation or Interquartile Range
- Number of Missing Values

If we have categorical data then we can compare

- All Possible Values
- Proportion of Categorical Values

Alternately, we can compare the distributions directly using distribution distance measures like the Kullback-Leibler divergence, the earthmover's distance, or the Chi Squared distance. This can either be done with the full data, or more practically, a representative sample.

In order to fully benefit from comparing distributions, some care will be necessary to handle differences in units and coding. If the data is continuous, float and integer data may need to be standardized. If the data is categorical, the proportions of the categories should be compared but the labels of the categories should likely be ignored.

Combining Distances

Using these distance measures, we will have an array of distances for every column pair. In order to find the best matches, we need to combine these multiple measures into a single score.

At the end of the column comparison phase, we will compared every pair of columns using a number of distance measures.

Say we are comparing two tables. The first has the columns `automobile`, `color`, `years`, and `owner`. The second table has the columns `car`, `color`, `age`, and `owned by`. We have compared all pair combinations using an edit distance and a dictionary-based semantic distance

	edit distance	semantic distance
automobile-car	10	0.1
automobile-color	8	1
automobile-age	8	1
automobile-owned by	9	1
color-car	3	1
color-color	0	0
color-age	5	1
color-owned by	8	1
years-car	3	1
years-color	5	1
years-age	4	0.6
years-owned by	7	1
owner-car	4	1
owner-color	4	1
owner-age	4	1
owner-owned by	4	0.5

For each column pair, we now need to aggregate these distances to get a final score. Existing toolkits allow the user to choose how they want to aggregate. For example, COMA lets the user decide whether they want the maximum score, the minimum score, the average, or median score. All of these approaches discard a lot of information and also require that each distance measure be normalized to have the same scale,

typically between 0 and 1.

Different distance measures provide different amounts of information about whether a pair of columns contain the same type of information. Ideally, we would weight the contribution of different distances measures based upon how much information it contributes to the problem of column matching. While it's possible to choose weights manually, it is a difficult task to do well especially as the number of distance measures increases. Many existing toolkits do allow for the user to set weights.

It would be a much more efficient use of information, and also simpler, to use the distance measures as features in a statistical model that predicted whether a pair of columns were about the same thing or not. Here, the algorithm sets the weights automatically, based upon training data. Some existing schema matching systems use this machine learning approach like LSD [4], but it is rare.

In addition to automatically learning weights, a machine learning approach can make the construction of distance functions much simpler. If the distance scores were simply features, then we could remove many of the arbitrary decisions that we must make within the existing approach. For example, instead of choosing how to score the distance between a timezone data type and a integer data type, we could just have a set of indicator variables that indicate what combination of data types a pair of columns have. Using logistic regression, or a similar technique, we would then learn what combinations suggested a good match or a bad match.

In order to use the machine learning, we must have training data, in the form of pairs of columns that have been labeled as either matching or not. These labels can be solicited from users through Active Learning. In Active Learning, the predictive model is initialized, and the system identifies column pairs where there is the greatest uncertainty of whether the pair is a match or not. This pair is presented to the user for labeling. This new labeled example is used to update the predictive model and the system now finds a new pair that it is most uncertain about. In our experience, a very good model can be learned with 50 user solicited labels.

Final Scores

At the end of this stage, we should have final score for every column. If we were using a machine learning approach and the same tables as we used in the example above, it might look like this.

	edit distance	semantic distance	**probability of a match**
automobile-car	10	0.1	0.8
automobile-color	8	1	0.1
automobile-age	8	1	0.1
automobile-owned by	9	1	0.1
color-car	3	1	0.3
color-color	0	0	1.0
color-age	5	1	0.2
color-owned by	8	1	0.1
years-car	3	1	0.3
years-color	5	1	0.2
years-age	4	0.6	0.7
years-owned by	7	1	0.1
owner-car	4	1	0.2
owner-color	4	1	0.2
owner-age	4	1	0.3
owner-owned by	4	0.5	0.6

Aligning Columns

Once we have the final column-pair scores, we can now decide which columns actually contain comparable information. There are number of ways to do this, and which one is best will depend on what assumptions you can make about the tables.

Simple Threshold

The most conservative approach is to consider all column pairs that have a score higher than some threshold. So if we set a threshold of 0.3 we would consider these possible column matches:

	edit distance	semantic distance	**probability of a match**
automobile-car	10	0.1	0.8
color-car	3	1	0.3
color-color	0	0	1.0
years-car	3	1	0.3
years-age	4	0.6	0.7
owner-age	4	1	0.3
owner-owned by	4	0.5	0.6

This equivalent to assuming that a table can have columns with somewhat redundant data. In this case, we would need to consider the possibility that the `automobile`, `color`, and `years` columns from the first table have redundant data. This is commonly happens with names and addresses, where the parts of an name or address are split across multiple column, i.e. `address_1`, `address_2`, `zipcode`.

Assuming Non-redundancy

If we assume that tables do not contain redundant columns, then if we select one column pair, we will not consider any other pairs that have either of those two columns.

There are few ways to do this, but in practice, a greedy approach works well. First, choose the highest scoring column pair, then discard all remaining pairs that have a column in common with the chosen pair, then choose the highest scoring column pair. Repeat until the set of column pairs is empty.

	edit distance	semantic distance	**probability of a match**
automobile-car	10	0.1	0.8
color-color	0	0	1.0
years-age	4	0.6	0.7
owner-owned by	4	0.5	0.6

This approach also assumes that the columns in one table are a subset of the columns in the other. In other words, every column in the narrower table has matching column in the larger table.

We can relax that assumption by using the greedy approach, but only considering column pairs that have scores above some threshold.

Table Matching

While we do want to know what columns we can compare, this usually comes after identifying what tables are likely to contain data about the same type of entities.

Once we compared all the columns in the table, we can combine the column pair distances into table level distances.

In existing systems, the typical way of doing this is to first align the columns, using one of the approaches described in the previous section. Then take the average score of those columns. This approach depends on the column alignment strategy, the width of the columns, and the threshold.

Using the simple threshold strategy for column alignment, the average score between our example tables is 0.57 if the column alignment score threshold is 0.3, but the table comparison score jumps to 0.78 if the column alignment score threshold is raised to 0.5.

A good measure of table similarity should only depend upon column similarity, not column alignment. There are a few approaches here that could be useful. In particular, we should investigate comparing the observed distribution of scores in a final column match table with the distribution that would be expected if we compared a random sample of columns. This should give us a measure that should not depend on column alignment or table widths.

Computational Complexity

The approaches in the existing toolkits cannot scale to large collections of data. Everything builds on comparing columns, and the number of comparisons is the Cartesian product of the width of columns in each dataset.

Let's say that all datasets in a collection have at least 5 columns. For each pair of datasets there are 25 ways to combine the 5 columns from each dataset. If there are 20 datasets, then there are 190 combinations of datasets. So for each distance measure, we will need to make at least $4,750$ comparisons. If there are 100 datasets, we will need to make at least $123,750$ comparisons. If there are 1000 datasets, we will need to make at least $12,487,500$ comparisons.

In order to reach scale, we will need to find a way to cut down on the number of comparisons we have to make. The most promising way to do this is with blocking. In blocking, we find a feature that columns which are likely to be comparable are likely to share, and then we only compare columns that share that feature. For example, we could only compare columns that had the same data type. Choosing good blocking rules involves a trade off between using enough blocking rules that you compare all the columns that should be compared while comparing the fewest combination of columns overall. The machine learning approach used in record linkage for blocking can be applied here.[11]

Further resources

A much more comprehensive further resource is `ontologymatching.org`, the home page of an organization which hosts academic conferences on the topic. There, one can find much more comprehensive (though often extremely long) surveys, other recent publications, and information on conferences. A subsidiary, the Ontology Alignment Evaluation Initiative, provides evaluations of many existing matchers, which may also be of interest.

Finally, listed below are a collection of open-source schema matchers whose licensing, documentation, and source code is all easily accessible online. A brief summary of their basic features is given in the table, though of course each also has many other varying focuses and strengths.

	COMA 3.0	OntoBuilder	AgreementMaker	LogMap	S-Match
Custom matchers	Yes	Yes	No	No	No
Instance data	Yes	Yes	No	No	No
Machine learning features	No	No	No	Yes	No
Semantic learning features	Yes	No	Yes	Yes	Yes

These matchers can be found at the following URLs:

- COMA: `http://dbs.uni-leipzig.de/de/Research/coma.html`
- OntoBuilder: `https://bitbucket.org/tomers77/ontobuilder/wiki/OntoM`
- AgreementMaker: `https://agreementmaker.github.io/`
- LogMap: `https://www.cs.ox.ac.uk/isg/tools/LogMap/`
- S-Match: `http://semanticmatching.org/s-match.html`

All matchers listed are cross-platform, written in Java, with both command-line and graphical interfaces. They all accept inputs via the standard XML-based format OWL specified by W3C, though other formats are supported by some.

References

[1] Arnold, Patrick and Rahm, Erhard. "Enriching Ontology Mappings with Semantic Relations." University of Leipzig. http://dbs.uni-leipzig.de/file/document_0.pdf

[2] Aumuller, David et. al. "Schema and Ontology Matching with COMA++." *SIGMOD 2005* (June 2005). http://dbs.uni-leipzig.de/file/comaplusplus.pdf

[3] Do, Hong-Hai and Rahm, Erhard. "COMA - A system for flexible combination of schema matching approaches." *Proceedings of the 28th VLDB Conference*, Hong Kong (2002). dbs.uni-leipzig.de/file/COMA.pdf

[4] Doan, AnHai et. al. "Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach." *ACM SIGMOD*, Santa Barbara (2001). http://homes.cs.washington.edu/~pedrod/papers/sigmod01.pdf

[5] Engmann, Daniel and Massmann, Sabine. "Instance Matching with COMA++." University of Leipzig. http://dbs.uni-leipzig.de/file/BTW-Workshop_2007_EngmannMassmann.pdf

[6] Gal, Avigdor and Modica, Giovanni. "Onto-Builder: Fully Automatic Extraction and Consolidation of Ontologies from Web Sources." http://ceur-ws.org/Vol-82/SI_demo_04.pdf

[7] Raunich, Salvatore and Rahm, Erhard. "Towards a Benchmark for Ontology Merging." University of Leipzig. http://dbs.uni-leipzig.de/file/E2IN2012-raunich-rahm.pdf

[8] Zapilko, Benjamin et. al. "Utilizing Regular Expressions for Instance-Based Schema Matching." Leibniz Institute for the Social Sciences. http://ceur-ws.org/Vol-946/om2012_poster4.pdf

[9] Princeton University "About WordNet." WordNet. Princeton University. http://wordnet.princeton.edu

[10] Řehůřek, Radim and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora." *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA. http://is.muni.cz/publication/884893/en

[11] Bilenko, Mikhail. *Learnable Similarity Functions and their Application to Record Linkage and Clustering*. University of Texas. http://www.cs.utexas.edu/%7Eml/papers/marlin-dissertation-06.pdf