# Entity Resolution Algorithms: Part I

The data world is having a come-to-Jesus moment. Brave ambitions to derive beautiful insights from previously unmined data sources are forcing us to face the unwieldy results of haphazardly-collected data. Data preparation, including entity resolution, absorbs a significant portion of analysis efforts. Reconciliation issues are characterized as below:

* Data is fractured across databases, requiring complex joins,
* Platforms built on different architectures,
* Varying data collection techniques invite errors and
* Original purpose of the data leaves missing, changed, multiple interpretation of values.

|  |  |  |
| --- | --- | --- |
| *Table 1 ER vs. Classification [[1]](#footnote-1)* | | |
|  | Entity Resolution | Classification |
| End Entity | Define multi-faceted entities using varying datasets, metadata | Entities are reconciled to each other in an existing static context |
| Single Truth | Relationships are among groups of entities | Pairwise relationships |
| Inputs | Capable of assessing entity across n-number of inputs | Typically matches between two entities at a time |
| Input Sequence | Sequence-neutral; ER models refine entities with new data inputs | Results are sensitive to sequence of when inputs were provided |

Frequently performed manually, entity resolution has quietly evolved into a formal discipline, complete with easily used interfaces[[2]](#footnote-2) and foundational algorithms. This post will focus on the latter of these, summarizing the most popular entity resolution algorithms and their practical implementations.

Clarifications[[3]](#footnote-3):

ER is not Classification. Simply identifying whether or not two records match or do not match is classification while entity resolution develops a dynamic entity using metadata. See *Table 1* for a comparison.

ER is not Clustering. The goal of clustering is to identify similar groupings of entities; the goal of entity resolution is to reconcile different iterations of the same entity down to their common iteration. Differences between the two are itemized in Table 1.

# Underlying Concepts

## Triangular inequality

In this post we will speak in terms of matching between two strings, however, the concept of triangle inequality, which addresses inequality among three entities, is an important consideration as this principle determines whether the algorithm is a formal metric or not.

Mathematically, triangular inequality says that the sum of the length of any two vertices of a triangle is greater than or equal to the length of the third. Alternatively, it may be defined using points. Say we determine that A=B. Separately we determine that B=C. Does it naturally follow that A=C? Not necessarily. Some algorithms preserve Triangular inequality while others defy it.

|  |  |  |
| --- | --- | --- |
| *Table 2 ER Objectives* | | |
| Objective | Description | Examples |
| String Matching | Quantify permutations needed to convert one string to another | Edit Distance, Alignment, Phonetic |
| Distance Metrics | Apply physical distance measures to abstract concept of data objects | Similarity, Text Analytics |
| Relational Matching | Conjunctive view reliant on one data object’s relationship to other objects | Set Based, Aggregate |

Dissimilarity Measure

For every yin there’s a yang. For every similarity, there’s a dissimilarity. When researching, beware that you may stumble on the yang when you were looking for the yin. The dissimilarity measure is, in fact, the discriminating factor separating some of the major algorithms.

# Let’s Get Started

While these two posts is intended as a general overview of algorithms, note that before setting out on your ER task you should first determine which of the main ER objectives is your goal (see *Table 2* for a summary).

For each of the major ER objectives (from *Table 2*) we will:

1. identify popular algorithms which address the objective,
2. describe variations, unique situations for each algorithm, and
3. show practical packages implementing the algorithm in two programming languages (R, Python), covered in Part II of this post

## The Data.

We begin with large datasets and apply blocking techniques[[4]](#footnote-4) to reduce the size of probable matches. For two[[5]](#footnote-5) given records within these datasets we begin with the compilation of a set of comparison vectors of similarity scores for component attributes. A good rule of thumb is that String Matching algorithms cover central concepts which are further developed by algorithms in the later groups.

## The Algorithms: String Matching.

String matching algorithms are concerned with whether or not two strings say the same thing. Outlined in *Table 4*, there are four essential approaches. They may, however, be further subdivided into exact element-by-element character or phonetic comparison.

|  |  |  |
| --- | --- | --- |
| *Table 3 Nature of Similarity Scores in String Matching* | |  |
| Boolean | 0 or 1; Match or non-match |  |
| Edit Distance | Quantified permutations to convert one textual string into another | Levenstein, Jaro-Winkler |
| Jaccard Coefficient | Ratio of existence or absence of one entity’s individual attributes in another | Jaccard |
| Phonetic Similarity | Pronunciation of letters are phonetically related on a 0 to 1 similarity scale  Aka, fuzzy matching | Soundex, Translation |

**Boolean Matching**, is easily understood as a Yes or No, 0 or 1, match or non-match between two strings. It is the most simplistic of the group and is the core logic on which the subsequent algorithms operate.

In the heavily-studied **Edit Distance**, similarity is quantified by physically measuring the permutations needed to convert one string into other. The core implementation of edit distance is **Levenstein**, which penalizes for insertions, deletions and substitutions. Over time Levenstein has been modified with additional costs for gaps (Sellers), transpositions (Smith-Waterman), and affine gaps, i.e., weighted costs per each of the actions or the location of where the permutation must be made (Gotoh).

**Jaro Distance** is a hybrid version of edit distance whose more popular counterpart, Jaro-Winkler, is considered a hybrid algorithm. Practically, Jaro slides along two strings, comparing nGrams along the way to quantify the number of characters appearing in the same position and the number of transpositions required for coincident characters which must be reordered in one string to match the other. Its best application is with short strings, also it disobeys triangle inequality.

The **Jaccard Coefficient** is an element-by-element measure of intersection. Stated otherwise, it is the ratio of the intersecting set to the union set. The Jaccard Coefficient satisfies triangle inequality. One frequently-confused issue: the similarity version of Jaccard and Tanimoto Coefficients are identical, but their dissimilarity coefficients diverge due to triangle inequality. While this justifies the need for two separate algorithms, they are frequently credited as the Jaccard-Tanimoto Coefficient as both mathematicians independently published this ratio unbeknownst of each other.

The fourth, **Phonetic Similarity**, is a throwback to that 80’s-era infomercial, “1-800-ABC-DEFG Hooked on Phonics worked for me!” Phonetic algorithms result in Soundex encodings, which sidestep misspellings and variations, by indexing a table of language-specific homophones for a string’s soundex encoding rather than searching the string itself. Two critical inputs to phonetic similarity are (1) discerning which language the string is written in and (2) knowing the context of the letters you are matching. The crucial former prerequisite is accomplished by matching pronunciation rules of letter sequences using their location in the string (“sch” in German vs. “sz” in Polish at beginning of a string). The latter is accomplished by parsing the string into a sequence of phonetic tokens according to pronunciation rules in that language. The International Phonetic Alphabet (IPA) is popularly used to identify tokens with corresponding sounds, though frequently criticized for being too fine of match.

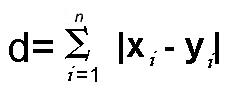
## The Algorithms: Distance Metrics.

|  |  |  |
| --- | --- | --- |
| *Table 4 Nature of Similarity Scores in String Matching* | |  |
| Euclidean | ‘As-the-crow-flies’ distance | L2-Norm, Ruler, Spearman[[6]](#footnote-6) |
| Manhattan | Distance if following a grid-like path, turning corners | L1-Norm, Taxicab, City-Block, Footruler[[7]](#footnote-7), Rectilinear |
| Minkowski[[8]](#footnote-8) |  | Lp |
| Chebyshev | Distance along axis on which the objects show greatest absolute difference | Lmax-Norm, Chessboard |
| Text Analytics | Pearson Coefficient, Jaccard Similarity Coefficient |  |
| Vector Similarity | Cosine Similarity, TFIDF |  |

While string matching compares strings element-wise, distance metrics incorporate a spatial element, measuring the literal distance between two entities using algorithms seen in *Table 4*. The first three are inter-related, easy visualizes by plotting the entities to be reconciled on a preference space with x, y axes. A discerning eye anticipates the obvious limitation of these, that only a certain number of attributes is practical.

**Minkowski Distance** is the generalized distance between two points in a plane. Specialized forms include Euclidean, Manhattan and the less-common Chebyshev[[9]](#footnote-9).

http://www.improvedoutcomes.com/docs/WebSiteDocs/image/diagram_euclidean_distance_metric.gifMathematically, **Euclidean Distance** is Minkowski Distance squared. Practically, it is the equivalent of the bishop in chess in that it moves diagonally, or as-the-crow-flies. The Euclidian Squared Distance Metric is a variation with quicker processing time since it does not take the square root.

**Manhattan Distance** is mathematically the Minkowski Distance raised to 1; it is the same as Euclidean, except for the requirement of absolute value since it is not squared. Practically it is the equivalent of a knight, which makes L-shape moves. Its name is coined after the great borough of New York City, where pedestrians and cars must obey the laws of street corners.

A great reference blog for the Minkowski Distance and its variations is http://bit.ly/1O0w3Xv.

## The Algorithms: Text Analytics.

In contrast to the Minkowski distances, which scale similarity on a scale of 0 to 1, **Pearson's Coefficient** scales from –1 to 1, in other words fitting similarity along a line, making it a better choice for non-normalized data and when attributes' scales are undefined. Mathematically, it is the ratio between two points' covariance and standard deviation.

The **Jaccard Similarity Coefficient** is mathematically the size, i.e., the existence of defined attributes using a binary 0/1, of the intersection of two points divided by the size of the union of the points. Bonus points if you noticed the repetition of Jaccard Coefficient in both String and Distance Metrics!

## The Algorithms: Vector Similarity.

First, a quick introduction to Vector Similarity. We construct a VSM (Vector Space Model) as a series of vectors quantifying frequency of a selected attribute inside a document. These vectors are subsequently assembled into a matrix, allowing easy algebraic manipulation. Two vector similarity functions are of particular note:

The widely-known bag of words model is enhanced to a 'bag of terms' with **TF-IDF**. Weighted TF-IDF incorporates local and global parameters, applying a logarithmic scale to account for a term's relative importance versus frequency of appearance. This allows the algorithm emphasize less-frequent terms' importance. TF-IDF normalizes any bias introduced into the vectors by keyword spanning, most commonly with the L2 (Euclidean) Norm. The equation is the row-wise multiple of two matrices:

TF (Term Frequency, the local parameter): matrix of vectors of selected terms' frequencies of appearance in each document

IDF (Inverse Document Frequency, the global parameter): diagonal matrix version of vector containing, for each term, the log of the number of documents divided by the number of documents in which the selected term appears

   \mathrm{tf}(t,d) = \sum\limits_{x\in d} \mathrm{fr}(x, t)    **x**    \displaystyle \mathrm{idf}(t) = \log{\frac{\left|D\right|}{1+\left|\{d : t \in d\}\right|}}  

**Cosine Similarity** is most useful when it is known that two points have a high proportion of non-shared attributes. Mathematically, the attributes are presented in a vector, allowing the algorithm to find the dot product of the two points. It measures the angle of the vector rather than the magnitude. Theoretically this results in the angle between the two points' attributes; a 90° angle is perfect dissimilarity.

## The Algorithms: Relational Matching.

Relational Matching algorithms retain many commonalities with Jaccard and Euclidean, but are mathematically differentiated since as a group do not satisfy triangular inequality. Practically speaking, while the aforementioned algorithms measure similarity between two documents, relational algorithms broaden the playing field, incorporating a third document's attributes into the mix.

While the **Tanimoto (Jaccard) Similarity Coefficient** is the same as the Jaccard Similarity Coefficient, the dissimilarity coefficient is where these two algorithms diverge. This is to say that Tanimoto is a proper similarity metric but its distance metric is not mathematically legal since it allows the two points to share commonality with a third point, causing it to disprove triangular inequality. In application, Tanimoto is preferred over Jaccard in cases when we want to allow the two points, themselves very different, to share commonalities with a third point. Mathematically, Tanimoto is the number of intersecting elements divided by the number of elements in either point.

**Dice's Coefficient** is mathematically the number of intersecting attributes divided into the total population of attributes, thus, as with Tanimoto, it shares a definition in its similarity metric version but Dice's dissimilarity coefficient is not the same as it does not satisfy triangle equality. Compared to Markowski's, Dice's coefficient is sensitive to hetergeneity in data sets and less sensitive to outliers.

A simplistic similarity measure is **Common Neighbors**, which predicts the likeness between two documents in terms of the number of common attributes each of those two documents independently shares with other documents.

**Adamic/Adar Weighted** modifies Common Neighbors to weight attributes that are shared infrequently relatively higher than those which are more common across all documents. Mathematically, this is accomplished by weighting a shared attribute's vector value with 1/ log of the number of times the attribute is shared across all documents.

|  |  |
| --- | --- |
| Table Hybrid Algorithms | |
| Jaro-Winkler | Jaro Distance modified to favor common prefixes |
| Monge-Elkan | Atomic Strings matching with Gotoh |
| Soft-TFIDF | A forgiving version of Cosine & Monge-Elkan |

## The Algorithms: Hybrid.

All seminal concepts experience merging with each other, thus, this section explains some well executed hybrid metrics derived from the ones above.

**Jaro-Winkler** is a hybrid algorithm with its roots in Jaro Distance, edit distance, but incorporates Cosine Similarity’s approach towards strings with high degree of dissimilarity and TF-IDF’s concept of applying a weight to certain elements. It improves on the basic Jaro Distance by accommodating for strings with a common prefix, effectively biasing its matching to favor similarity between two otherwise-dissimilar strings who share a common prefix.

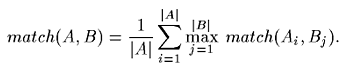
 **Monge-Elkan** is sometimes considered synonymously with Smith-Waterman Edit Distance, but the two are differentiated as Monge-Elkan uses the Gotoh Distance. The confusion is understandable, as Gotoh amends Smith-Waterman distance by accommodating affine gaps. Practically, it applies the combined power of Levenstein and Jaro Similarity Measures to n-Gram subsets of strings (called *atomic strings*). Mathematically, Monge-Elkan uses Gotoh edit distance to evaluate atomic strings against each other. Before deciding on Monge-Elkan you should understand how sensitive your matching is to the symmetry of your strings, i.e., if one string is longer than the other. It has quadratic time complexity due to its recursive calculations.

Figure http://bit.ly/1W3zC7v

**Soft-TFIDF** adds a forgiveness factor to Cosine Similarity and Monge-Elkan, which are intolerant of spelling errors as they roll along atomic strings in the order of appearance by incorporating TF-IDF’s concept of matrix of terms (i.e., letters) to develop an internal frequency per Atomic string. Soft-TFIDF calculates an inner score comparator, thus allowing partial matches.

## Implementation

There it is. Hopefully Part I of this two-part post solidifies the purpose behind the most popular entity resolution algorithms, giving you an idea of which ones work best for your immediate needs. The next step is, of course, implementation. Before embarking on Part II of this post, consciously decide whether you will implement a Learning- or Non-Learning-Based Matching approach, detailed in *Table 5*. The latter approach, as characteristic of machine learning, minimizes human interaction.

Stay tuned for Part II of this post, which will cover implementation in both Python and R. The programming deities have generously built packages inside both of these languages to ease implementation and we will guide you through them.

|  |  |  |
| --- | --- | --- |
| *Table 7 Learning vs.Non-Learning Metrics* | | |
| Approach | Non-Learning-Based Matching (Probabilistic) | Learning Based Matching |
| General | Edit Distance | 2 phases: model generation, model application |
|  | Single or Conjunctive Match |  |
| Requirements | Similarity Threshold as parameter. Apply threshold-based selection of the matching entity pairs | Model generation needs training dataset w/ manually labeled boolean correspondences |
| Examples | PPJoin+ Cosine  PPJoin+ Jaccard  Fellagi Sunter Trigram  Fellagi Sunter TokenSet  Fellagi Sunter Winkler | FEBRL SVM (Freely Extensible Biomedical Record Linkage)  MARLIN (Multiple Adaptive Record Linkage w/ Induction) has 2 string similarity measures (edit distance, cosine) & several learners (SVM, decision trees) |

1. http://jeffjonas.typepad.com/jeff\_jonas/2007/09/entity-resoluti.html [↑](#footnote-ref-1)
2. D-Dupe is a Python implementation of active learning. [↑](#footnote-ref-2)
3. http://linqs.cs.umd.edu/projects//Tutorials/ER-AAAI12/ER\_Tutorial\_part2.pdf [↑](#footnote-ref-3)
4. Blocking is not the subject of this paper, however, please refer to <insert blog- thought I saw one by Ben?> [↑](#footnote-ref-4)
5. When triangle inequality differentiates one algorithm from another we will insert a third record [↑](#footnote-ref-5)
6. For two vectors of ranked ordinal variables, aka, Spearman Distance [↑](#footnote-ref-6)
7. For two vectors of ranked ordinal variables, aka, Footruler Distance [↑](#footnote-ref-7)
8. http://bit.ly/21jyR9G [↑](#footnote-ref-8)
9. Not covered in this blog [↑](#footnote-ref-9)