Entity Resolution: A Primer

Mike Purewal Fall 2021

Outline

1. Overview & Landscape

2. Entity Resolution Pipeline

3. Scaling & Blocking In-Depth

Problem Statement

- Entity Resolution (ER) detects different entity profiles that correspond to the same real-world object. This includes:
  - De-duplication: grouping records that correspond to the same object
  - Record linking: matching records across data sources
  - Reference matching: matching noisy records to a reference source

Example Challenges for Matching

- Name/attribute ambiguity: Thomas Cruise, Michael Jordan
- Data entry error: age=320, date of birth=Juky 20, 1990
- Missing values: (this example is missing)
- Changing attributes: business address or a desk location
- Data formatting: Dec 11, 2021; 11-12-2021
- Abbreviation/truncation: Jon Doe Jr., Jon Doe Junior, Jon Doe

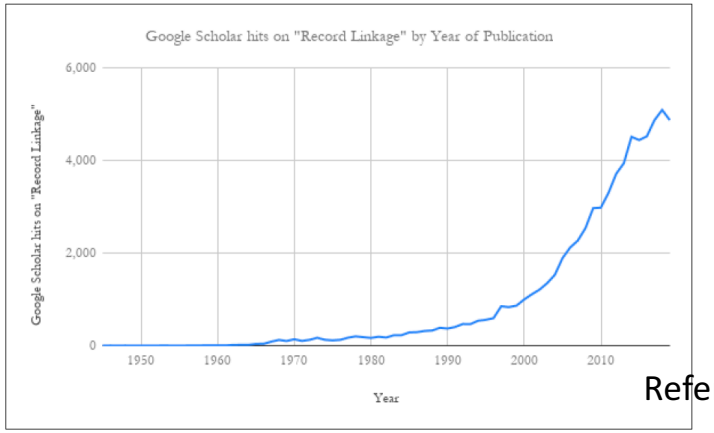
Alternative Names

- Entity Resolution goes by many names (ironically).
- There is a lack of standardization of the subject's taxonomy. Primarily because of its cross-disciplinary nature.
- Alternative names: Duplicate detection, record linkage, fuzzy match, hardening soft information, reference matching.

Business & Research Use Cases

ER begins with post-WWII health research and continues with genealogical researchers exploring linking records from historical files and the linking of census records. Since then, use cases across disciplines have exploded.

- **Preventative Crime**: Financial Fraud, Watchlist Screening, Identity
- **Marketing**: List de-duplication, omni-channel, next best action
- **Risk Analysis**: Credit Risk, Brand Protection
- **Privacy Compliance**: Right to be Forgotten Monitoring, Central Disclosure and Consent Tracking
- **Public Health & Safety**: School Safety



Reference:

Industry Players

Many companies perform their own Entity Resolution schemes in-house. However, there are a few companies that need to do this as an explicit function at scale.

- Senzing
- Enigma
- Reonomy
- Palantir
- Dedupe.io
- ThinkData
- Quantexa

Python Packages

- jellyfish
- Dedupe
- Postal (libpostal)
- GraphX: Apache Spark's API for graphs and graph-parallel computation.
- GraphFrames:package for Apache Spark which provides DataFrame-based Graphs.

Entity Resolution Pipeline

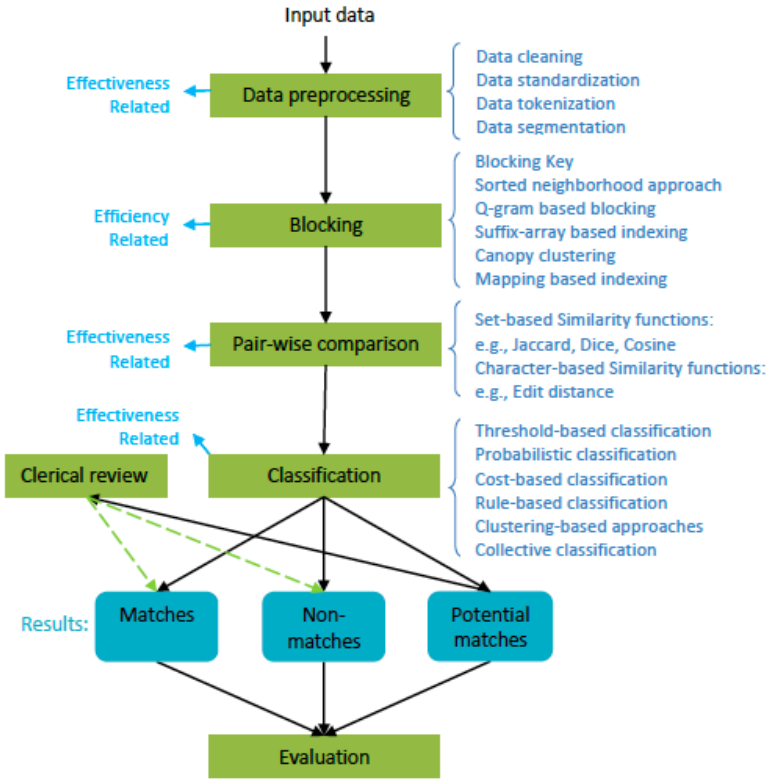


Figure 1: The general process of ER based on [15]

Description

A. Data preprocessing: this step is mostly generic for unstructured/non-numeric data - cleaning, standardizing, tokenizing and segmenting.

B. Blocking: If ER was not a computationally expensive process, there would be no need for blocking. Mostly an art at this point of the subject's life, this finds ways to speed up the calculation (efficiency) without losing performance (effectiveness).

C. Pair-wise comparison: More generically, this can be thought of as an evaluation of whether two records are identical. This is mostly what one thinks of when they first begin to link or de-duplicate their data set.

D. Classification: In the case of more complicated frameworks, the labeled pair is evaluated in a larger context to determine if it is a Match/Non-Match/Potential Match.

E. Evaluation: Assessing the effectiveness and efficiency of the pipeline choices.

Compute

- Quadratic complexity in pairwise matching
- Example sizing problem: 100 records from 100 sources.
  - How many pairwise comparisons overall? (10^2*10^2)^2=10^8;
  - Assume records from different sources are known to be non-matches.
  - How many pairs of records from same source? (10^2)^2*10^2 = 10^6== 1% of overall dataset.
  - 99% can be excluded from pairwise matching approach.
- *Effectiveness vs Efficiency*

Modeling Problems

- Identifying Matches ~O(R) vs. Non-matches~O(R^2); Imbalanced classification problem.
- *Blocking and Filtering* - techniques to address efficiency.
- Labels, ground truth, golden source: not easy to source.

Big Data Problems

- ***Volume***: Generally, more of everything. Larger/more datasets requires more efficient compute (parallelization) and algorithms (blocking, filtering).
- ***Variety:*** Heterogeneity, more tracking creates "data lakes".
- Multi-relational: structured entities - Microsoft Powerpoint, Microsoft Word \subset of Microsoft Office
- Multiple applications: different accuracy requirements

Schema Normalization

- Schema matching. Examples:
  - Phone Number, Cell number, Contact number, Home number
- Compound attributes:
  - Address
- Libpostal
- Schema.org

Data Normalization

- Implemented using dictionaries, rules and Python libraries
  - Casing
  - Whitespace
  - Typographical errors / variations
  - Abbreviations and nicknames

Names

- Types of name variations:
  - Spelling: typographical, doesn't usually impact phonetical. Meyer, Meir
  - Phonetic: structure fundamentally changes
  - Compound names
  - Alternative names
  - Initials
- Sources of variations
  - Optical Character Recognition (OCR)
  - Keyboard based data entry
  - Phone / verbal communication entry
  - Limitations like field length
  - Self-reporting of names
- Phonetic encoding
  - Soundex
  - Phonex
  - Phonix
  - NYSIS
- Matching can use distances of original name, matching of phonetic encoding, or combination. Authors suggest direct matching is best, with cavaets.

Overview

- Given an assumption, substantially reduce O(n^2) pairwise matching.
- Reduction example:
  - Specific
    - 1e6 records --> 1e12 comparisons
    - Define 1e2 blocks, each containing 1e4 records
    - Each block has 1e8 comparisons, all comparisons-->1e10
    - Reduction 100x: Reduction ratio is 99%
  - Generic
    - 1e*a* records-->1e(2*a*) comparisons
    - Define 1e*b* blocks, each containing 1e(*a*-*b*) records
    - Each block has 1e(2*a*-2*b*), all comparisons-->1e(2*a*-*b*)
    - Reduction 1e*b* x: Reduction ratio is 1-(1e-*b)*

Blocking Objective

- Trades slightly lower *effectiveness* for significantly higher *efficiency*. The more comparisons in a block, the more duplicates will be detected at the cost of more compute. Blocking is about finding the balance.
  - Pair Completeness: *Recall* [Effectiveness]
  - Pair Quality: *Precision* [Efficiency]
  - Reduction Ratio: reduction in comparisons [Efficiency]

Character-Base similarity functions

- Edit distance quantifies dissimilarity of two strings.

- Levenshtein distance: accounts typographical errors
  - Counts the minimum number of operations (insertion, deletion, substitution) required to transform one string into the other.
  - Lower bound is 0 (exact match); Upper bound is length of longer string.
  - Demerau-Levenshtein distances counts transposition as single edit (fish, ifsh).

- Jaro-Winkler distance: alignment based, account for name variations
  - Measures common characters with considerations for transposition.
  - Winkler modification weights earlier characters more significantly.
  - Normalized from 0 to 1. 0 is lowest distance, or most similar.

Implementation

- Jellyfish package in Python (see Jupyter Notebook example)

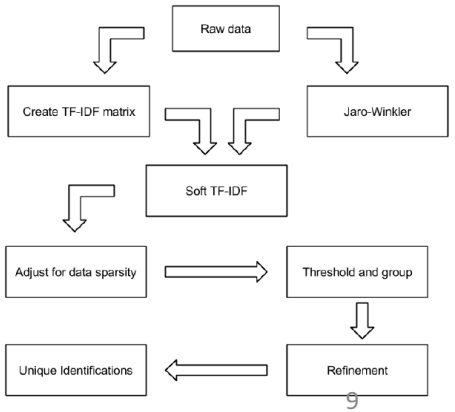
- Calculating edit distance is a computationally expensive operation.

Token-based similarity functions: Set similarity

- **Overlap**:
  - *Number of common attributes*;
  - |x \intersect y|
- **Cosine**:
  - *Overlap normalized by product of of vectors sizes;*
  - Overlap / sqrt(|x| \dot |y|)
- **Dice (Sorensen-Dice)**:
  - *Overlap normalized by average size*
  - 2*Overlap / |x|+|y|
- **Jaccard**:
  - *Overlap normalized by **unique** set of words*
  - Overlap / (|x|+|y| - Overlap)

Hybrid Approach

- Soft TF-IDF: Combines TF-IDF with character similarity measure (eg Jaro-Winkler)
  - TF-IDF = tf * idf
  - TF: Term frequency weights all features equally, Order of words doesn't matter
  - IDF: Inverse document frequency, higher weight on rare features – usually a good discriminator



Component-wise similarities

- Given a vector of comonent-wise similarities for a pair of records, output the record match.
  - r1 = ['Joe', 'Blogs', 'NYU']; r2= ['Joseph', 'Blogs', 'New York University']
  - Compute similarity score on vector of attributes.
  - For example, Jaro-Winkler: 0.83, 1.0, .72

- Approach 1: Formulate rules of what results in match.
  - First name >0.4, Last name >0.9, University >0.2
  - Maintenance is hard, rules can be complex

- Approach 2: Weighted sum > threshold results in match.
  - score = w1*similarity_fn(r11,r21) + w2*similarity_fn(r12,r22)+....
  - Assigning weights and threshold.

Basic Model Driven Approach: Fellegi and Sunter

Definitions

- **m**: Probability that fields link conditional on entities matching.
  - Accounts for *Data Quality*
  - Two records matching the same student should have the same *gender* field with 99% probability.

- **u**: Probability that entities don't match conditional on two fields linking. Often simplified as: chance that 2 fields will randomly match
  - Accounts for *Data Value*
  - If NYU ID fields link, 0.1% chance the entities will not match.
  - If gender fields link, 50% chance the entities will not match.

- Reference link

Table 1. Pair of records compared during a probabilistic record linkage process. In this case, overall match weight, based on three matched fields and two non-matched fields, is 17.20.

Field 1: First Name	Field 2: Last Name	Field 3: Date of Birth	Field 4: Address	Field 5: Gender
Jana	Asher	10/17/1970	603 Brook Court	F
Jane	Asher	10/17/1970	1111 Jackson Ave	F
$m_1 = 0.95$	$m_2 = 0.99$	$m_3 = 0.97$	$m_4 = 0.95$	$m_5 = 0.99$
$u_1 = 0.001$	$u_2 = 0.00004$	$u_3 = 0.001$	$u_4 = 0.01$	$u_5 = 0.48$
$\log_2((1-m_1)/(1-u_1)) = -4.32$	$\log_2(m_2/u_2) = 14.60$	$\log_2(m_3/u_3) = 9.92$	$\log_2((1-m_4)/(1-u_4)) = -4.31$	$\log_2(m_5/u_5) = 1.31$

m is the probability the fields agree given they represent the same entity; **u** is the probability the fields agree given they do not represent the same entity.

Calculation

- Calculate weights:
  - Linked field i: log_2(m_i / u_i)
  - Non-linked field j: log_2 ((1-m_j) / (1-u_j))

- Decision rule:
  - Sum weights across all fields. "Naive Bayes" type assumption that assumes fields are independent, which isn't always the case. For example, name and gender are typically correlated.
  - Rank order records from highest to lowest
  - Qualitatively identify either:
    - One threshold for two classes: match/non-match entities
    - Two thresholds for three classes: match/non-match/uncertain entities

DRAFT: Work in Progress

Supervised Learning Techniques

Requires Labels

- Decision Trees
- Support Vector Machines
- Ensembles
- Conditional Random Fields

Unsupervised Learning Techniques

- Clustering: k-means
- Clustering; Agglomerative clustering
- Generative Models
- Active Learning

Training Set Generation

- Most pairs are 'easy' non-matches.
- Some are hard to judge by humans. Ambiguity or missing information. Bayes limit.
- Potential label source: Golden source might be embedded within the data: eg SSN links are known to be entity matches, but exist only on a subset of records. Preserve these as train/test set.
- Other labeling tactics:
  - Crowdsourcing
  - Snorkel
  - Data Augmentation
  - Synthetic Data Generation

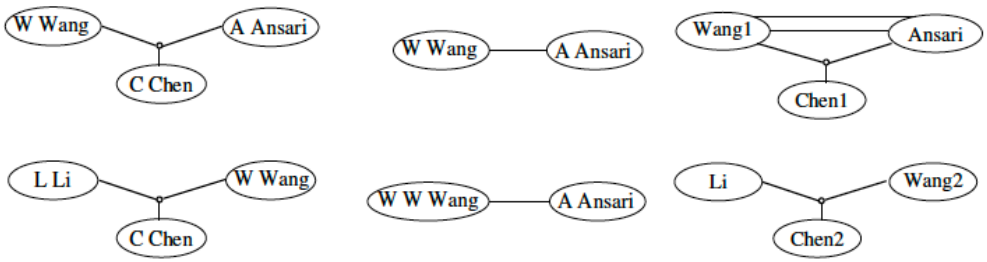
DRAFT: Work in Progress

Adding Constraints

- Transitivity - for deduplication: If M1=M2, M2=M3, then M1=M3
  - Add matches based on transitive closure; can result in large chains
  - Example: Jonathan->John->Jon->Joe->Joseph
- Exclusivity - for record linkage: If M1=M2, then M1<>M3 and M2<>M3
  - Weighted k-partite matching
- Functional Dependency - for data cleaning: If M1=M2, then M3=M4
- Soft constraints: If ___, then likely match.

Collective Entity Resolution: Problem Definition

- Pairwise ER techniques assume that the entity for a reference depends on the *attribute similarities* of related references and thus independent of other matches.
- Collective ER assumes the entity for a references depends on the *entities to which they correspond.*
- In other words, matching decisions depend on other matching decisions. They are no longer independent.
- Necessary trade-off between computational complexity and increased accuracy.
- The identity of the entity depends on the relationships and the relationships depend on the identity.



Collective Entity Resolution: Solutions

- Attribute based:
  - Pairwise techniques based on attributes only.
  - Transitive closure may be taken over the pairwise decision.
  - In the case of common names or attributes, this approach will almost always have trouble disambiguating.
  - sim_A(r_i, r_j)
- Naive relational:
  - Simple method that accounts for relationships is to include it as an attribute in the matching score. The importance of this feature can be weighted by alpha.
  - sim_H(r_i, r_j) is the similarity measure of hyper-edges associated with those entities. The hyper-edge similarity is measured by accounting for common relationships. For example, h_1 and h_2 are more similar if they share the same entities associated with them.
  - sim_NR = (1- \alpha) sim_A(r_i, r_j) + \alpha sim_H(r_i, r_j)
  - Misleads in domains where names are frequent and hyper-edges are dense.
- Collective relational:
  - Considers the labels of related *clusters* that represent the entities.
  - Similarity is dynamic, as the labels evolve as entities are added to neighboring clusters.
  - sim(c_i, c_j) = (1- \alpha) sim_A(c_i, c_j) + \alpha sim_R(c_i, c_j)
- Reference

Confusion Matrix

	Match	**Non-Match**
Linkage	True Match (True Positive)	False Match (False Positive)
Non-Linkage	Missed Match (False Negative)	True Non-Match (True Negative)

Precision

Recall

Classification Performance Metrics

- **Accuracy**: TP + TN / (TP+TN+FP+FN)
  - Note that for imbalanced datasets, a model that 'predicts' all negatives can have good accuracy. Why?
- **Precision**: TP / (TP + FP)
  - Of the *linked records*, how many were true matches?
- **Recall**: TP / (TP + FN)
  - Of the *matched records*, how many were missed?

Measuring Performance

- **F1 score**: 2 / recall^-1 + precision^-1
  - Harmonic mean. Creates a scalar that skews towards worst performer.
- **Adjusting misclassification cost**: Precision or Recall?
  - Maximize recall subject to precision > threshold

Spectrum of Linkage Outcomes: Matching Uncertainty

<table>
<tr><th>Fields Linked</th><th>Model Output</th><th colspan="3">True Match Status</th></tr>
<tr><td>Agree on all</td><td rowspan="2">Links</td><td rowspan="3">Matches</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>Agree on most</td></tr>
<tr><td>Agree on some,
Disagree on some</td><td>Uncertain</td><td rowspan="3">Non-Matches</td><td>Uncertain</td></tr>
<tr><td>Disagree on most</td><td rowspan="2">Non-Links</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>Disagree on all</td></tr>
</table>

Entity Resolution Pipeline

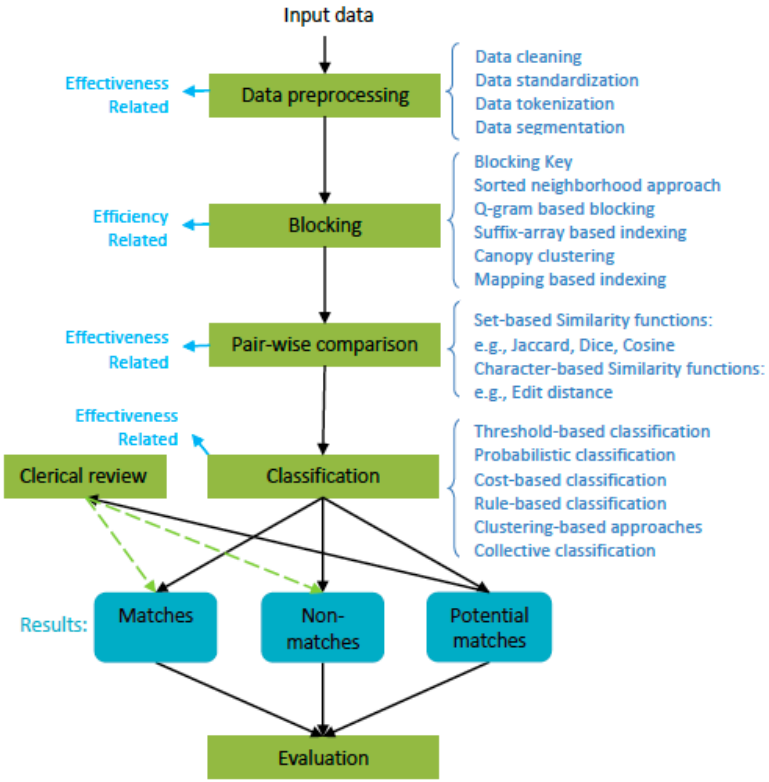


Figure 1: The general process of ER based on [15]

Scale: Parallelization

- *[Common in ER]* **Intra-step parallelism**: aka data parallelism; independent operations are carried out simultaneously on elements of one data set. Each step in ER are performed in parallel.
- *[Not common in ER]* **Inter-step parallelism**: aka task parallelism; relationships between adjacent steps are independent – the output of the previous step effects the next.
- ER typically parallelized the pairwise comparison approach only.
- Historically, parallelization was achieved via parallelized DMBS, followed by Hadoop MapReduce and now Apache Spark.

Scale: Efficiency Classification

- **Blocking**: see Blocking section

- **Data partitioning**: significantly impacts performance as data migration is expensive.
  - Size-based: partition based on even data size
  - Pair-based: evenly dividing on candidate pairs.
  - Block-based: each block to a separate node.

- **Load balancing**: Goal is to evenly assign workload.
  - Prevention-based: control block size to be less than some threshold value.
  - Remedying-based: oversized blocks are segmented further.

- **Redundancy handling**:
  - Measures to remove redundant pairs resulting from overlapping blocks.
  - Transitive closure: identify matches without having to compare.
  - Pruning

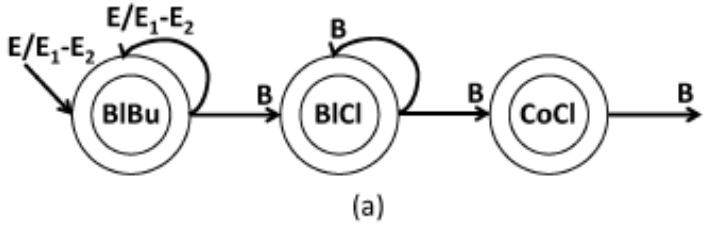
Overview

- Given an assumption, substantially reduce O(n^2) pairwise matching.
- Reduction example:
  - Specific
    - 1e6 records --> 1e12 comparisons
    - Define 1e2 blocks, each containing 1e4 records
    - Each block has 1e8 comparisons, all comparisons-->1e10
    - Reduction 100x: Reduction ratio is 99%
  - Generic
    - 1e*a* records-->1e(2*a*) comparisons
    - Define 1e*b* blocks, each containing 1e(*a*-*b*) records
    - Each block has 1e(2*a*-2*b*), all comparisons-->1e(2*a*-*b*)
    - Reduction 1e*b* x: Reduction ratio is 1-(1e-*b)*

Blocking Objective

- Trades slightly lower *effectiveness* for significantly higher *efficiency*. The more comparisons in a block, the more duplicates will be detected at the cost of more compute. Blocking is about finding the balance.
  - Pair Completeness: *Recall* [Effectiveness]
  - Pair Quality: *Precision* [Efficiency]
  - Reduction Ratio: reduction in comparisons [Efficiency]

Blocking Workflow Stages

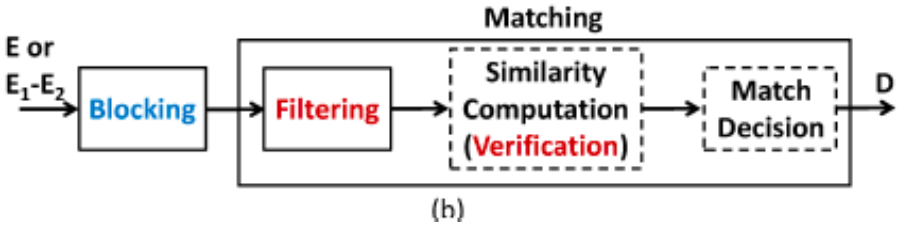


- Block Building (BlBu): Applies a blocking scheme to build the blocks. May be iterative.

Optional: Block Processing

- Block Cleaning (BlCl): Discarding unnecessary blocks using a static or dynamic method
- Comparison Cleaning (CoCl): Discarding individual comparisons using learning or non-learning approaches.

Example Entity Resolution Workflow



- Filtering: identifying *potential* matches pruning true negatives and allowing false positives. An example of basic filtering is looking at field length.
- Verification: pairwise comparison.
- Match decision: classification.

Block Building: Blocking Key

- Blocks are defined by the predicate or key.
  - Examples are first letter of first name, first letter of last name, plus zip code.
  - These predicates are crudes models with high recall (approaching 1), but low precision.
  - Since recall may not be exactly 1 in practice, and iterate key variations.

Domain	Blocking Predicate
Census Data	Same 1st Three Chars in *Last Name*
Product Normalization	Common token in *Manufacturer*
Citations	*Publication Year* same or off-by-one

Block Building: Taxonomy

- **Key selection**:
  - non-learning: expert based
  - learning based: training set
- **Schema awareness**:
  - schema aware: less noisy fields
  - schema agnostic: uses all fields
- **Key type**:
  - hash/equality based methods: map pair of entities with **same** key to the same block
  - sort/similarity based methods: map pair of entity with **similar** key to the same block
- **Redundancy awareness**
  - Redundancy free: every entity to a single block; disjoint blocks.
  - Redundancy positive: every entity to multiple blocks; overlapping blocks
  - Redundancy neutral: degree of redundancy is arbitrary
- **Constraint awareness**
  - Lazy: no constraint
  - Proactive: enforce constraints on blocks (eg size).
- **Matching awareness**
  - Static: independent of subsequent matching results; immutable block.
  - Dynamic: updates as matching detected

Block Building: Taxonomy

Table 1. Taxonomy of the Block Building methods discussed in Sections 3.2 and 3.3.

Method	Key type	Redundancy awareness	Constraint awareness	Matching awareness
Standard Blocking (SB) [50]	hash-based	redundancy-free	lazy	static
Suffix Arrays Blocking (SA) [3]	hash-based	redundancy-positive	proactive	static
Extended Suffix Arrays Blocking [25, 116]	hash-based	redundancy-positive	proactive	static
Improved Suffix Arrays Blocking [34]	hash-based	redundancy-positive	proactive	static
Q-Grams Blocking [25, 116]	hash-based	redundancy-positive	lazy	static
Extended Q-Grams Blocking [11, 25, 116]	hash-based	redundancy-positive	lazy	static
MFIBlocks [79]	hash-based	redundancy-positive	proactive	static
Sorted Neighborhood (SN) [64, 65, 136]	sort-based	redundancy-neutral	proactive	static
Extended Sorted Neighborhood [25]	sort-based	redundancy-neutral	lazy	static
Incrementally Adaptive SN [189]	sort-based	redundancy-neutral	proactive	static
Accumulative Adaptive SN [189]	sort-based	redundancy-neutral	proactive	static
Duplicate Count Strategy (DCS) [42]	sort-based	redundancy-neutral	proactive	dynamic
DCS++ [42]	sort-based	redundancy-neutral	proactive	dynamic
Sorted Blocks [41]	hybrid	redundancy-neutral	lazy	static
Sorted Blocks New Partition [41]	hybrid	redundancy-neutral	proactive	static
Sorted Blocks Sliding Window [41]	hybrid	redundancy-neutral	proactive	static
(a) Non-learning, schema-aware methods.				
ApproxRBSetCover [16]	hash-based	redundancy-positive	lazy	static
ApproxDNF [16]	hash-based	redundancy-positive	lazy	static
Blocking Scheme Learner (BSL) [104]	hash-based	redundancy-positive	lazy	static
Conjunction Learner [21] (semi-supervised)	hash-based	redundancy-positive	lazy	static
BGP [49]	hash-based	redundancy-positive	lazy	static
CBlock [150]	hash-based	redundancy-positive	proactive	static
DNF Learner [55]	hash-based	redundancy-positive	lazy	dynamic
FisherDisjunctive [76] (unsupervised)	hash-based	redundancy-positive	lazy	static
(b) Learning-based (supervised), schema-aware methods.				
Token Blocking (TB) [120]	hash-based	redundancy-positive	lazy	static
Attribute Clustering Blocking [124]	hash-based	redundancy-positive	lazy	static
RDFKeyLearner [158]	hash-based	redundancy-positive	lazy	static
Prefix-Infix(-Suffix) Blocking [123]	hash-based	redundancy-positive	lazy	static
TYPiMatch [96]	hash-based	redundancy-positive	lazy	static
Semantic Graph Blocking [113]	-	redundancy-neutral	proactive	static
(c) Non-learning, schema-agnostic methods.				
Hetero [77]	hash-based	redundancy-positive	lazy	static
Extended DNF BSL [78]	hash-based	redundancy-positive	lazy	static
(d) Learning-based (unsupervised), schema-agnostic methods.				

References: Papers

- Collective Entity Resolution in Relational Data. Bhattacharya, Getoor
- A Survey of Blocking and Filtering Techniques for Entity Resolution. PAPADAKIS, SKOUTAS, THANOS, PALPANAS
- A Comparison of Personal Name Matching: Techniques and Practical Issues. Christen
- An Introduction to Probabilistic Record Linkage with a Focus on Linkage Processing for WTC Registries. Asher, Resnick, Brite, Brackbill and Cone.
- Unsupervised record matching with noisy and incomplete data. Gennip, Hunter, Ma, Moyer, de Vera, Bertozzi
- Soft TF-IDF: https://arxiv.org/pdf/1704.02955.pdf

References: Talks

- Entity Resolution: Tutorial. Getoor, Machanavajjhala