**1.Similarity algorithms**: **1.1.** **Edit Distance(Levenshtein)**Min. operations (insert, delete, substitute) to transform x to y; Compute: For strings x (length n) and y (length m), initialize a (n+1)\*(m+1) matrix; d(i,0)=i, d(0,j)=j. Similarity Measure: s(x,y) = 1 - d(x,y) / max(length(x), length(y)). **1.2. Needleman-Wunsch**: Alignment: set of correspondences between characters of x and y, allowing for gaps. Diff from Edit Distance: computes similarity score instead of distance values; generalizes edit costs into a score matrix, allowing for more fine-grained score modeling; generalizes insertion and deletion into gaps and theirs costs from 1 to cg. Compute: Initialize (n+1)\*(m+1) matrix for strings x (length n) and y (length m), set s(i,0)=-i\*cg, s(0,j)=-j\*cg for gap cost cg, compute s(n,m) for similarity. **1.3. Affine Gap Measure**: An extension of Needleman-Wunch that handles longer gap. Compute: Initialize matrices M, , ,:opening gap costs; *:* extension gap costs ; Init: M(0,0)=0, (0,0)=-, (0,0)=-, (i,0)=--\*(i-1), (0,j)= --\*(j-1); others: -inf. **1.4. Smith-Waterman Measure(local alignment)**: Extends Needleman-Wunsch; focuses on local alignment(most similar substrings within x and y), key changes: 1, allowing restarts at any position, 2, ignoring non-optimal suffixes for higher local similarity scores. Compute: Initialize a (n+1)x(m+1) matrix for strings x (length n) and y (length m), set s(i,0)=0 and s(0,j)=0, aim for the highest score in any cell for local similarity. **1.5. Jaro Measure:** Ideal for short strings like names. Common characters and must be identical and within positions of each other. Transpositions occur if the i-th common character in x doesn't match the i-th in y. Formula: , where is number of common characters, is transpositions. **1.6 Jaro-Winkler Measure:** Enhances Jaro for strings with common prefixes. Formula: , where PL is the length of the longest common prefix and PW is a prefix weight. Helps when x and y have a low Jaro score but share a prefix. **1.7 Overlap Measure:** Counts common tokens in sets and , generated from strings x and y. Overlap . Eg. For x = "dave", y = "dav" with 2-grams, , so . **1.8 Jaccard Measure**: Computes similarity by ratio of common tokens to total unique tokens in and , sets of tokens from x and y. Formula: . **1.9 TF/IDF Measure** Main idea: two strings are similar if they share distinguishing terms. 1,Document: A collection of terms. Each string or text is treated as a 'document'. 2, Term Frequency (TF): tf(t, d)= How many t in d. It measures how often a term appears in a document. 3, Inverse Document Frequency (IDF): , where N is the total number of documents, and is the number of documents containing term t. It reflects the importance of a term; rarer terms have higher IDF. 4, Feature Vectors(TF/IDF score): ; 5, Physical Meaning of TF/IDF: Combines TF and IDF to assess the importance of a word in a document within a collection. High TF/IDF score indicates a term is frequent in a particular document but rare in the overall document set, signifying its distinguishing importance. 5, TF/IDF Similarity Score: s(p,q), where p and q are strings. The score is higher if p and q share terms that are frequent in each individual string (high TF) but rare across the entire document collection (high IDF), indicating significant shared content that is unique to these documents. **1.10** Generalized Jaccard Measure: Extends traditional Jaccard to account for non-identical but similar tokens.(eg. Misspelled words) **1.11 Soft TF/IDF** the generalized Jaccard using TF/IDF as the similarity measure. **1.12 Monge-Elkan Measure**: For x = A1 … An and y = B1 … Bm, the score is , where s' is a secondary similarity measure like Jaro-Winkler. Note: . This measure is effective for comparing strings with rearranged but similar components. **1.13 Soundex Measure** Converts surnames into a 4-letter code based on their phonetic properties; surnames are considered similar if they share the same code. Effective primarily for Caucasian names, it may not perform as well for names with different phonetic structures, such as those of East Asian origin. **5. Minhashing**: **5.1 Shingling** Convert documents to sets. e.g. D1="abcab", the set of 2-shingles S(D1) is {ab, bc, ca}. This technique captures word order, differing from simply using sets of words. Jaccard similarity is a natural measure for shingle similarity. For effective comparison, k(k-shingles) should be sufficiently large(k=5 for short doc, 10 for long). **5.2 Minhashing**  1. Key Ideas: convert large sets to short signatures, while preserving similarity; 2, compute: a, Given a matrix with documents as columns and shingles as rows. b, Rearrange the rows according to the permutation c, Select row index of the first 1 in each column d, Record these row indices for each document column. This forms the Minhash signature for each document. 3, Property: . 4, one-Pass Implementation Trick: Pick K = 100 hash functions h. **5.3 Locality-Sensitive Hashing**: Focus on pairs of signatures likely to be from similar documents. **Concepts:** a, Column: Represents a document in the Minhash signature matrix, with each row being a hash value from a different hash function. b, Band: A group of consecutive rows in the Minhash signature matrix. c, Rows Per Band: The number of rows included in each band. Determines the granularity of comparison between document signatures. d, Buckets: In LSH, each band of a document's signature is hashed into a bucket. Documents whose bands hash to the same bucket are considered as candidate similar pairs. **Formula:** bands and rows per band, Probability that at least one band is identical: **7. Data Integration** def: Data integration is the combination data from multiple sources into a unified view. **a) Why Hard** (1)Systems-level reasons: Managing different platforms; SQL across multiple systems is not so simple; Distributed query processing (2) Logical reasons: Schema (and data) heterogeneity; (3) Social reasons: Locating and capturing relevant data in the enterprise; Convincing people to share (data fiefdoms); Security, privacy and performance implications. **b) Architecture** (1)Data warehousing: integrate by bringing the data into a single physical warehouse; (2)Virtual data integration: leave the data at the sources and access it at query time. **c) Process** (1)Schema alignment: mapping of structure; (2)Record linkage: matching based on content; (3)Data fusion: reconciliation of mismatching content. **d) Big Data Integration** Why need? (1)Building web-scale knowledge bases; (2)Reasoning over linked data. Why hard? (1)Deep Web Quality. (a)Poor quality data can have big impact. (b)Data inconsistency. Why such inconsistency? (1)Semantic ambiguity; (2)Unit errors; (3)Pure errors. **8. Schema Alignment** **8.1 Schema Heterogeneity** Discrepancies: (1)Table and attribute names;(2)Tabular organization;(3)Coverage and granularity level;(4)Data-level variations. **8.2 Mediated Schema** enables domain specific modelling. e.g. remove an redundant attribute. **8.3 Schema Matching** **a)Matcher**: schemas →similarity matrix. It includes: (1)name matchers: Use string matching techniques(like edit distance, Jaccard, Soundex), pre-process names(split them using certain delimiters, like saleLocID, expand known abbreviations or acronyms, like loc) ; (2)instance-based matchers: consists of: (1)recognizers: Use dictionaries, regexes, or rules to recognize data values of certain kinds of attributes;(2)overlap matchers: Typically applies to attributes whose values are drawn from some finite domain, like country names. Jaccard measure is often used. (3)Classifiers. Builds classifiers on one schema and uses them to classify the elements of the other schema. steps: 1) For each element, train a classifier to recognize the instances of it. 2)A training set of positive and negative examples: all data instances of a particular element as positive examples and all data instances of the other elements as negative examples. 3)Use the classifiers for computing the similarity score between an element from the source and another element from the target. 4)Aggregate the scores produced by the classifer on data instances of the element from target. **b) Combiner** combined(i,j)=avg/max/min{matcherScore(m,i,j)}, where matcherScore(m, i, j) is score between and as produced by the m-th matcher. avg: when we do not have any reason to trust one matcher over the others; max: when we trust a strong signal from matchers, i.e., if a matcher outputs a high value, we are relatively confident that the two elements match; min: when we want to be more conservative. **c) Constraint Enforcer** Domain integrity constraints used to prune the result. (Domain knowledge is needed) **d) Match Selector** similarity matrix->matches. Simplest way: (1)Thresholding. (2)Stable Marriage. Unstable: Suppose is currently matched with , and is matched with . If both and would rather be matched with each other than with their current partners, then it is unstable. Algorithm: Repeat: (1) Let (i, j) be the highest value in the sim such that and are not in match. (2) Add ≈ to match. **e) Multi-Strategy Learning** (1)Training Phase: Multiple Learners: a)Use different machine learning algorithms (e.g., Naive Bayes, Decision Tree) as learners (L1, L2, ..., Lk). b) Each learner develops classifiers for elements of the mediated schema G. c) Training Data: Derive training examples from semantic matches between training sources (S1, S2, ..., Sm) and the mediated schema G; Positive examples: matched attribute instances; Negative examples: unmatched attribute instances. d) Meta-Learner: Train a meta-learner to assign weights (we,Li) to each learner for each element of G; These weights indicate the reliability of each learner for each element. (2)Matching Phase: a)Applying Learners: Apply learners to attributes of a new schema S (e1’, e2’, ..., en’); Each learner computes similarity scores between G's elements and S's attributes. b) Combining Predictions: Use meta-learner's weights to combine learners' predictions. Final similarity score represents the overall match between element e of G and attribute e’ of S. **8.4 Big Data Integration: Schema Alignment** a)Volume: Huge data/single source, numerous sources. b)Velocity: Changes in sources, increasing sources. c)Variety: Diverse/heterogeneous sources, changing schemas. **a) Dataspace Approach**: Standard integration infeasible for Big Data due to volume/variety (high modeling cost), velocity (maintenance cost); pay-as-you-go approach - start simple, add complexity iteratively; Components - Bootstrapping (auto schema matching), Guidance (semantic coherence). **b)Bootstrapping DI Systems**: Model uncertainty (source attributes), automatic P-mediated schema, uncertainty incorporation. **c) Probabilistic Mediated Schemas**: Auto-create by inspecting sources, attribute clustering, volume/variety causes clustering uncertainty. **d) Probabilistic Mappings**: Map P-mediated schema ↔ source schema, query answering by-table/by-tuple semantics. **e) WebTables**: Google's web crawl data, keyword search to rank tables, challenges - ambiguous web features, query relevance. WebTables Keyword Search: FeatureRank (table features), SchemaRank (schema coherency, pmi), example query “presidents of the US”. **f)Annotating Web Tables**: Goal - identify entities/column types/relationships, use catalog (type hierarchy, entities, relationships). **g)Finding Related Tables**: Augment table info from corpus, entity complement (top-k tables for union, schema consistency). **Select Feature for Splitting:** Use a criterion like Information Gain or Gini Impurity to choose which feature to split on at each node. **Information Gain:** Prefer the feature that most reduces uncertainty (entropy). **Gini Impurity:** Prefer the feature that leads to the most homogeneous child nodes.**1. Calculate Entropy (for Information Gain):** is the proportion of the sample belonging to class i in set S. **2. Calculate Information Gain:** is the subset of S for each value v of attribute A. **3. Calculate Gini Impurity (alternative to Entropy):** **4. Perform Split Based on Best Feature:** Split the dataset into subsets based on the feature value. Each subset forms a node in the tree. **5. Recursive Splitting:** Repeat steps 1-5 for each child node. Stop if all instances in a node belong to the same class or another stopping criterion is met (e.g., maximum depth). **6. NN** **a) compute** In a feedforward neural network, forward propagation involves calculating neuron inputs with and outputs using activation functions like , , and ; activations are . Backpropagation computes gradients: derivatives of Sigmoid are , Tanh are , and ReLU are if else . Error at the output layer is , propagating back via , and gradients are , . a) Naive Bayes: Probability of class given features is . Assumes feature independence, so for features. Uses Laplace smoothing to handle zero probabilities: Adjusted where is count of feature in class , is total count of features in , (typically 1) is the smoothing parameter, and is number of distinct feature values. **9.Record Linkage** Def: same schema. Find same tuples. **9.1 Difference from String Matching**: Treating tuples as concatenated strings for matching is simplistic; loses sophistication and domain-specific knowledge. **9.2 Rule Based:** **a)Linearly Weighted Combination Rules** (1)Match if ; Not matched if ; Human review otherwise. (2)Similarity score: , = weight of i-th attribute. (3)Pros: Simple, learnable weights. (4)Cons: Linear increase with simi(x, y); No diminishing returns. **b)Logistic Regression Rules** (1)Address diminishing-returns: , . (2) not constrained to [0, 1]. (2) Useful when many signals contribute to match; Not all signals need to fire. (3)Pros: Ensures diminishing returns, fits multiple signal scenarios. (4)Cons: Not detailed. **c) More Complex Rules** (1)Encode complex knowledge: Match conditions based on specific attribute thresholds. (2) Steps: Check name similarity, then phone or address match. (3)Pros: Encodes complex knowledge, fast, easy to start and understand. (4)Cons: Labor-intensive, challenging to set weights and write rules. **9.3 Learning-based** **a)preprocessing**: (1)Supervised Learning: Learn model M from training data , label is 'yes'/'no' for match.(2)Features: Define features , each quantifying one aspect of domain relevance.(3)Transform Training Data: Convert to , , is transformed label. **b) Learning a Linearly Weighted Rule**: Got a Feature Vector: . For strings, use Jaro-Winkler and edit distance; for numbers, use exact match. **c)Learning a Decision Tree** (1)Feature Conditions: e.g., leads to 'yes'/'no' decisions based on subsequent feature tests.(2)Tree Structure: Decision nodes based on feature scores; leaves represent match decision. **d)Pros and Cons** (1)Pros: Can examine many features automatically, construct complex rules.(2)Cons: Requires many training examples, hard to obtain; clustering can help address this. **9.4 Matching by Clustering** **a) AHC Process**: (1)Start with each tuple as a single cluster, iteratively merge most similar clusters, stop at desired cluster number or similarity threshold. (2)Similarity Score Computation:a)Single link: ; b)Complete link: ; c)Average link: ; d)Canonical Tuple: Represent each cluster by a canonical tuple, e.g., combining similar names or phone numbers. **9.5 Probabilistic Approaches to Matching** **a) Modeling**: Use probability distribution for matching domain. **b) Reasoning**: Make decisions based on the distribution. **c) Benefits**: Incorporates various domain knowledge, leverages AI/DB techniques. **d) Disadvantages**: Computationally expensive, hard to debug. **e) Methods**: Generative models (full distributions, data generation), discriminative models (conditional probabilities for matching). **f) Bayesian Networks/Generative Models**: Encode full probability distributions, guide Bayesian network construction. **g) Naive Bayes for Data Matching**: Compute using Bayes' Rule; assume feature independence given M. **h) CPTs Learning**: Convert training examples into feature vectors; learn Conditional Probability Tables (CPTs). **i) Naive Bayes Assumption**: Independency of features given M. **j) EM Algorithm**: Estimate unknown quantities and missing values iteratively; apply to match new tuple pairs. **k) Generative Model for Entity Mentions**: Select entities, representatives, mentions; model over attributes; probability distribution over documents. **l) Matching Mentions**: Determine if mentions correspond to the same entity; simplify model for matching. **m) Estimating Model Parameters**: Labelled entity-mention pairs, transformation probabilities over attribute values. **9.6 Scaling Up Data Matching** **a) Goals:** (1)Minimize number of tuple pairs to match. (2)Minimize time for each pair matching. **b)Techniques** (1)Hashing: hash tuples into buckets, match only tuples within each bucket. (2)Sorting: Sort tuples by a key, match within a window size . (3)Indexing: index tuples such that given any tuple a, can use the index to quickly locate a relatively small set of tuples that are likely to match a (4)Canopies: Group tuples into overlapping clusters, then match within clusters. (5)Representatives: Use group representatives for matching new tuples. (6)Combining Techniques: E.g., hash, then sort within buckets, then match. **c)Minimizing Match Time:** (1)Tailor to application; use short circuiting in rule-based approaches.(2)No universally established technique. **e)Parallel Processing:** (1)hash tuples into buckets, then match each bucket in parallel. (2)match tuples against a taxonomy of entities (e.g., a product or Wikipedia-like concept taxonomy) in parallel. two tuples are declared matched if they match into the same taxonomic node **10 Indexing** Target: reduce the number of record pairs that are to be compared in detail as much as possible. **a)eg** The Sorted Neighbourhood Approach is an indexing method used for data matching, where records are first sorted based on a 'sorting key'. A sliding window of fixed size then moves over these sorted records, generating candidate pairs for matching within each window position. This method limits comparisons to adjacent records in the sorted list, thereby reducing the computational load and improving efficiency in large datasets. The approach is particularly effective in narrowing down potential matches by exploiting the proximity of similar records post-sorting. **11. Evaluation** Precision-recall Graph: Ideally, high recall and high precision→ curve as high up in the upper right corner as possible. Accuracy suffers from class imbalance problem and hence is not suitable for assessing matching quality. F\_score: . **measuring matching complexity** In evaluating data matching techniques, efficiency and effectiveness are measured by run-time and the number of candidate record pairs generated, necessitating a platform-independent comparison method. Key metrics include the Reduction Ratio (), assessing the proportion of candidate pairs generated compared to all possible pairs, and Pairs Completeness (), reflecting the true match status of candidate pairs and akin to recall. Pairs Quality () parallels precision, evaluating the success in generating true matches. Notation used encompasses and for matched and non-matched record pairs respectively, and and for true matched and non-matched candidate pairs. The approach reveals trade-offs: a high reduction ratio indicates many pairs removed but may lead to lower pairs completeness, and there's an inverse relationship between pairs completeness and pairs quality, highlighting the balance between removing pairs in indexing and missing true matches. **12. Data Fusion** **Def**: Data Fusion aims to detect and solve data conflicts from multiple  
sources. **Naïve Voting** is a data fusion method that facilitates conflict resolution by supporting differing opinions among multiple data sources. It operates effectively when these sources are independent and possess similar levels of accuracy. The method involves aggregating votes or inputs from various sources to determine the most probable value. However, when dealing with sources of varying accuracies, Naïve Voting needs to be adjusted to give more weight to inputs from more knowledgeable or reliable sources. Additionally, the method must account for sources that may copy from others, by assigning lower weights to their votes. A notable challenge with Naïve Voting is addressing the 'wisdom of minority' problem, where the method might overlook valuable but less frequent inputs from minority sources, especially in cases where sources significantly disagree on values. **12 Data Privacy** **12.1 Four Types of Attributes in Data**: (1)Identifier. Can be used to uniquely identify a person (2)Quasi-identifier. Cannot uniquely identify a person by the attribute May be used to re-identify a person when linked with some external dataset. (3) Sensitive attribute: Attributes that a person may want to conceal. (4)Non-sensitive attribute. Privacy of a person will not be violated when the attribute is disclosed. **12.2Privacy by Design** Minimize: The amount of personal data should be restricted to the minimal amount possible (data minimization); Hide: Personal data and their interrelations should be hidden from plain view. **12.3 Methods** **a) Anonymization** The subject identity of the data records are removed, concealed or hidden. e.g. Data swapping: Data values of selected records are swapped to hide the true owner of the records. A high rate of swapping destroys relationships involving the swapped and unswapped variables. Data synthesis: The values of sensitive variables are replaced with synthetic values generated by simulation. **12.4 Re-identification Attacks** : Even when identifying details are removed, individuals can still be reidentified. e.g. Correlation attacks: Linking a data set to other sources to create more ne grained and unique database entries. **12.5 K-Anonymity&L-diversity** K-Anonymity: No individual's record in the dataset released is distinguishable from at least K-1 other records. L-diversity: For each group of records sharing a combination of quasi-identifier (key attributes) attribute values, there are at least L “well-represented” values for each confidential attribute. **scalability** **4.1Scalability in Matching:** Avoid s(x, y) on all pairs; use blocking for promising pairs. Find candidate method for subset Z ⊆ Y; match if s(x, y) ≥ t. **4.2 Inverted Index Over Strings:** Convert strings in Y to documents (tokens), build inverted index. Quickly find documents containing term t. Example: x = {lake, mendota}, find ID lists for tokens in Z = {4, 6}. **4.3 Limitations of Inverted Index:** Long inverted lists for common terms. Costly to build/manipulate. Large enumeration of pairs sharing terms. **4.4 Size Filtering:** Retrieve strings in Y meeting size constraints of x. B-tree index for size satisfying strings. Jaccard measure used for constraints; 1/t ≥ y/x ≥ t. Example: x = {lake, mendota}, t = 0.8, 2.5 ≥ y ≥ 1.6. **4.5 Prefix Filtering:** Shared terms in large subsets of x and y.Overlap measure O(x, y) = x ∩ y. Subset x' from prefix of x for overlap. Example: x1 = {lake, mendota}, x' = {lake}, candidates {y4, y6}. **2.Data Preprocessing**: **2.1 Major Tasks** (1)Data Cleaning Correct bad data, filter some incorrect data out of the data set and reduce the unnecessary detail of data. (2)Data Transformation. The data is consolidated so that the mining process result could be applied or may be more efficient. (3) Data Integration. Merging of data from multiple data stores. (4)Data Normalization To express data in the same measurement units, scale or range. (5) Missing Data Imputation: To fill the variables that contain missing values with some intuitive data. (6)Noise Identification: To detect random errors or variances in a measured variable.