Entity Matching in the Wild: a Consistent and Versatile Framework to Unify Data in Industrial Applications

Yan Yan
Amperity, Inc.
yanyan@amperity.com

Stephen Meyles
Amperity, Inc.
stephen@amperity.com

Aria Haghighi
Amperity, Inc.
aria@amperity.com

Dan Suciu
University of Washington
suciu@cs.washington.edu

ABSTRACT

Entity matching – the task of clustering duplicated database records to underlying entities – has become an increasingly critical component in modern data integration management. Amperity provides a platform for businesses to manage customer data that utilizes a machine-learning approach to entity matching, resolving billions of customer records on a daily basis. We face several challenges in deploying entity matching to industrial applications at scale, and they are less prominent in the literature. These challenges include: (1) Providing not just a single entity clustering, but supporting clusterings at multiple confidence levels to enable downstream applications with varying precision/recall trade-off needs. (2) Many customer record attributes may be systematically missing from different sources of data, creating many pairs of records in a cluster that appear to not match due to incomplete, rather than conflicting information. Allowing these records to connect transitively without introducing conflicts is invaluable to businesses because they can acquire a more comprehensive profile of their customers without incorrect entity merges. (3) How to cluster records over time and assign persistent cluster IDs that can be used for downstream use cases such as A/B tests or predictive model training; this is made more challenging by the fact that we receive new customer data every day and clusters naturally evolving over time still require persistent IDs that refer to the same entity. In this work, we describe Amperity's entity matching framework, Fusion, and how its design provides solutions to these challenges. In particular, we describe our pairwise matching model based on ordinal regression that permits a well-defined way to produce entity clusterings at different confidence levels, a novel clustering algorithm that separates conflicting record pairs in clusters while allowing for pairs that may appear dissimilar due to missing data, and a persistent ID generation algorithm which balances stability of the identifier with ever-evolving entities.

ACM Reference Format:
Yan Yan, Stephen Meyles, Aria Haghighi, and Dan Suciu. 2020. Entity Matching in the Wild: a Consistent and Versatile Framework to Unify Data in Industrial Applications. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3318464.3386143

1 INTRODUCTION

Entity matching is a fundamental operation that occurs in virtually all modern data management tasks: in information extraction [6], in knowledge base construction [24], in data integration [7], in big data analysis [11], and in machine learning [23]. As businesses and organizations seek to better leverage a growing amount of data, the importance of entity matching, and data integration systems more broadly, have grown substantially to become mission-critical components. In particular, such systems have become a key part of how consumer businesses use their customer data to drive decisions in products, customer service, and marketing.

Entity matching is a straightforward problem to define: given a large collection of records, cluster these records so that the records in each cluster all refer to the same underlying entity. While this definition captures the core substance of the problem, deploying systems that use entity matching typically face challenges not encapsulated by this definition. In this work, we'll examine some of the real-world challenges faced by entity matching systems in the wild. Specifically, challenges commonly encountered by customer

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3386143

PK	DS	Fname	Lname	Suff	Birthdate	Email	Address	City	State	Zip
r_1	A	John	Smith			jsmith12@gmail.com	123 W Elm Street	Seattle	WA	98109
r_2	A	John	Smith				123 West Elm Street	Seattle	WA	98109
r_3	B	J	Smith	Jr		jsmith12@gmail.com		Seattle	WA	98109
r_4	C	John	Smith		1980-01-12	jsmith12@hotmail.com		Seattle	WA	98109
r_5	C	John	Smith		1980-01-12					
r_6	C	John	Smith	Sr	1955-12-22		123 West Elm Street	Seattle	WA	98109

Table 1: A toy example of a group of inter-related records. The records are collected from different data sources and thus have different levels of completeness. The DS (data source) attribute refers to the source table of the record, and the PK (primary key) is the record identifier within the dataset; the tuple (DS, PK) uniquely identifies each customer record in the database.

data platforms (CDP)[1], such as Amperity, which aim to produce unified customer profile records from customer data. In the CDP setting, duplicate customer records typically come from different business data feeds that store customer information. For example, a typical retail business will have web traffic on their eCommerce site associated with a customer, in-store transactions with a loyalty card, customer service interactions, and many other sources.

Amperity [2] is a Seattle-based software startup, launched in 2017, with the mission of building an intelligent customer data platform to help businesses unify their customer data and drive better actions and insights. Faced with many application challenges for these businesses, we have developed Fusion, a unique entity matching system that clusters customer records into their underlying entities. The Fusion system has been utilized by over 25 businesses to process over two billion customer records, representing more than a billion customer relationships; these customer entities are in turn merged with hundreds of billions of behavioral, contextual, and transactional data to form unified customer profiles. A report we recently published [14] has shown that before using Fusion businesses misidentified on average 23% of their customer base. This misidentification is a result of under-clustering or over-clustering customer identities and has the downstream effect of misattributing lifetime spend amongst customers. Furthermore, the customers that are easiest to under- or over-cluster are usually the most valuable customers, interacting with a business through many channels, and account for over half of the overall revenue. By correcting this misidentification, a business can develop more accurate business KPIs, including total customer counts and customer lifetime values, and their customers will receive fewer wrong offers, messages and targeting.

In this paper, we describe Amperity's Fusion system by first providing an overview of the system in Section 2. The focus of this work is on how this system addresses the following application challenges. We use a small toy example described in Table 1 throughout the paper to explain the problem in a consistent way.

1.1 Multi-Confidence Clusterings

Although entity matching is a well-defined problem, there is a large degree of uncertainty about whether records refer to the same entity. For example in Table 1, most annotators would agree that r_1 and r_2 represent the same person. However, that question becomes much more difficult when evaluating the record pair r_2 and r_5, with the decision potentially changing depending on the commonness of the name in a given geographical area. This uncertainty should affect how we consider using the output of entity matching for downstream applications.

If our goal is to use this information for direct customer interaction (e.g, an email survey asking "How was your purchase last week?"), we typically require high confidence in our matches since being wrong will create a negative customer experience. In contrast, if the email content is less specific, the cost of a precision error can be lower and we may be willing to trade lower precision for increased recall..

Given the scale of the data (in the billions of records for Amperity), it is expensive and prohibitive to run entity matching end-to-end multiple times for each application and its preferred precision/recall trade-off. In Section 3, we propose the use of ordinal regression to model the pairwise similarity which assigns a discrete ordinal match level between two records; this in turn permits us to produce multiple clusters at different match levels while paying the full cost of entity matching only once. The match levels used by our ordinal regression approach are also understandable to annotators and have a strong agreement among them, suggesting an operator can make informed choices about what level to use for a given application. In Section 6, we empirically compare

[1]https://en.wikipedia.org/wiki/Customer_data_platform
[2]https://amperity.com

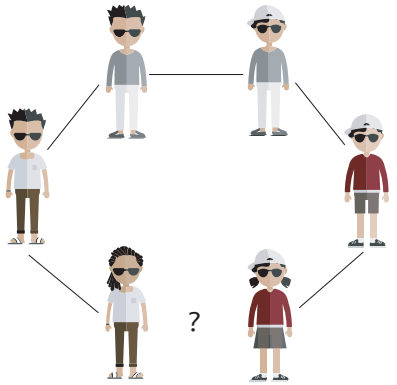

Figure 1: Illustration of multiple different entities merged into one connected component through transitive closure. Although each edge connects similar records, the transitive closure contains highly dissimilar entities.

our ordinal regression approach to a binary logistic regression approach and show significantly reduced classification errors.

1.2 Resolving Conflicts in Clustering

The goal of entity matching systems is not just to detect pairs of records associated with the same entity, but rather to produce coherent clusters. A common approach in entity matching systems [20], is to create entity clusters by forming connected components (e.g, transitive closure) over detected pairs of similar records. The transitive closure can lead to the formation of long chains of records, where individual pairs are classified as matches, but multiple records in the chain can be quite different from each other; see Figure 1 for an illustrated example.

In the entity matching literature, this problem has been deemed as the "bad-triplet" problem [1]. A bad-triplet relationship can be defined as a graph between three records where two edges are positive (meaning a pairwise classifier predicts that the record pair represents the same entity) and the third is negative (an edge where the connected records do not appear pairwise similar).

Most existing approaches resolve the bad-triplet problem by performing some form of graph pruning over the transitive closure to split the clusters that contain negative edges. Crucially, many of these approaches assume that the bad-triplet problem occurs exclusively because of conflicting information, whereas in many practical settings, they arise due to information systematically missing from a source of data; for instance, birthdates are typically not in transaction data feeds. In this paper, we use *disagreeable-triplet* to denote the first scenario and use *agreeable-triplet* to denote

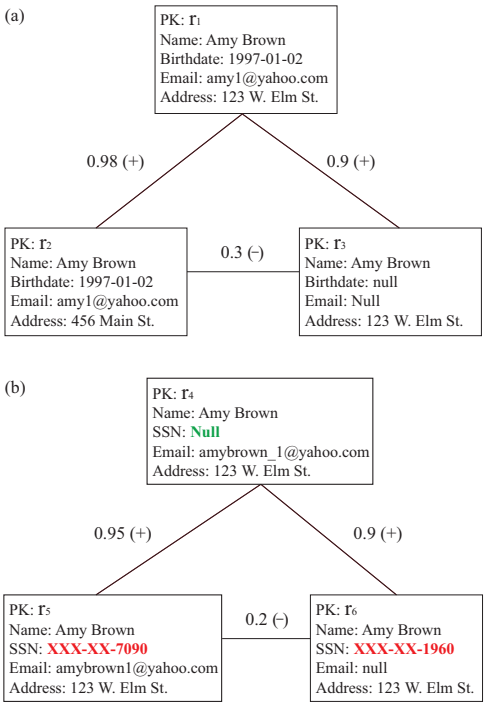

Figure 2: Example of two types of bad-triplets, where + and − represent the match and non-match relationships of record pairs. (a) is an agreeable triplet because there is no conflicting information among the records. The negative edge between r_2 and r_3 is caused by the lack of information; (b) is a disagreeable triplet because there is a hard conflict in the cluster - the conflicting SSNs between r_5 and r_6. Because r_4 has the missing SSN, it transitively connects r_5 and r_6 without surfacing the conflict. The conflicting values are highlighted in red while the critical missing information is highlighted in green. The difference between SSNs is used as an example of a hard conflict. In practice, the hard conflicts are configured according to business requirements.

the second one. Figure 2 (a) shows an agreeable-triplet example where the edge is negative primarily because of a *lack* of information rather than *conflicting* information as in Figure 2 (b), which presents a disagreeable-triplet example. These two types of negative edges should be treated differently, with the former being admissible in a cluster and the latter typically pruned. In Section 4 we describe our hierarchical clustering approach that does exactly this. In Section 6, we also compare our approach empirically to correlation clustering [1] and a few commonly used hierarchical clustering approaches [12, 13] to demonstrate improved clustering results.

1.3 Cluster Identifiers Over Time

Most industrial applications of entity matching require a persistent "primary key" or identifier for an entity cluster in order to track and aggregate information about this entity over time. For instance, if we perform an A/B test over customers, we typically want to split customers into multiple groups and track response variables at the level of individual customers using a stable customer identifier. This requirement is trivial to satisfy if records are static and entity matching is performed just once since each unique cluster can be assigned a unique key based on the records in it. However, customer data is seldom static, and the underlying data being matched grows and changes over time. As such, we must also re-run or update clustering in order to have customer entities reflect changes in source data. Entity clusters may change for several reasons such as adding or removing records, merging records with other clusters, or splitting records into different clusters. For instance, each new transaction by a customer will add a new element to the entity cluster. With this context, assigning a persistent identifier to clusters is challenging since clusters can and do change over time. In Section 5, we describe Amperity's persistent ID assignment algorithm that maintains persistent and stable cluster identifiers in the face of changes to clustering.

The rest of the paper is organized as follows. Section 2 provides a system overview of Fusion. Section 3 reviews a threshold-based ordinal regression algorithm and explains why it is important to our applications. Section 4 introduces a hierarchical clustering-based algorithm for cluster-level conflict resolution. Section 5 presents a special algorithm that generates persistent cluster identifiers that follow the entities even if entity clusters change over time. Section 6 shares experimental results, showing improved performance using ordinal regression to solve the entity-matching problem, the efficacy of using the proposed clustering algorithm to resolve clustering conflicts, and the extent to which cluster ID reassignments can be avoided using our persistent ID assignment algorithm. Section 7 concludes our discussion.

2 FUSION SYSTEM OVERVIEW

In this section, we first provide an overview of Amperity's entity matching framework, Fusion, and discuss how Fusion solves many real-world challenges that are not tackled by existing solutions. To make our discussion more concrete, we present a toy example (see Table 1) and use it throughout the rest of the paper.

The Fusion architecture runs in the cloud atop infrastructure as a service (IAAS), such as Amazon AWS or Microsoft Azure. Customers drop periodic (generally daily) data updates which trigger Fusion pipeline. Data is ingested and aggregated with existing records in a versioned and immutable manner. Storing the data in this manner enables longitudinal analysis of the data over time including the calculation of persistent identifiers (see Section 5). The newly ingested version of the data initially lacks resolved entities. The entity matching pipeline groups the raw data, and creates a complete view for each entity. The data is further augmented with a persistent entity identifier using our persistent-ID assignment algorithm. This process also generates change data capture (CDC) outputs as well as alerts in the event of issues. The output of Fusion is delivered to downstream "identity-oriented" systems enabling market segmentation or customer relationship management (CRM). The entire entity matching pipeline is implemented on Apache Spark [27] for high throughput and scalability.

At a very high-level, Fusion follows the standard workflow as other common entity matching systems with the following major components: 1) preprocessing; 2) blocking; 3) pairwise comparison/classification; 4) computing connected components; and 5) clustering (see Figure 3 for the architecture overview of the Fusion pipeline).

Given a collection of raw datasets, we first preprocess all records in each table to normalize the data to an approximately common representation, with invalid values removed. For example, in the toy example in Table 1, records r_1 and r_2 have null birthdates because the birthdate information is systematically unavailable in their data source (see the data source ("DS") column in Table 1).

The blocking step groups records by multiple *blocking indices* in order to determine which record pairs should be considered by the matching classifier as potential matched record pairs. The blocking index can be constructed by a single attribute, such as last name or email alias, or a combination of multiple attributes, such as the name initials. Many blocking methods have been developed to scale up entity matching systems. In Fusion, we applied an algorithm proposed in [4] to learn a set of DNF (disjunctive normal form) based blocking indices. Record pairs that share any of the blocking indices are evaluated in the next pairwise comparison step. In our toy example, all pairs of records share at least one of the blocking indices, composed by the initial of the first name and the full last name, so we would consider all $\binom{6}{2}$=15 candidate record pairs. In practice, blocking reduces the $O(n^2)$ space of all possible record pairs to a much smaller space of candidate record pairs.

The next pairwise comparison step uses a learned ordinal regression model to assign a discrete ordered label to each candidate pair (see details in Section 3). The possible ordinal labels ordered from least-likely to most-likely to indicate

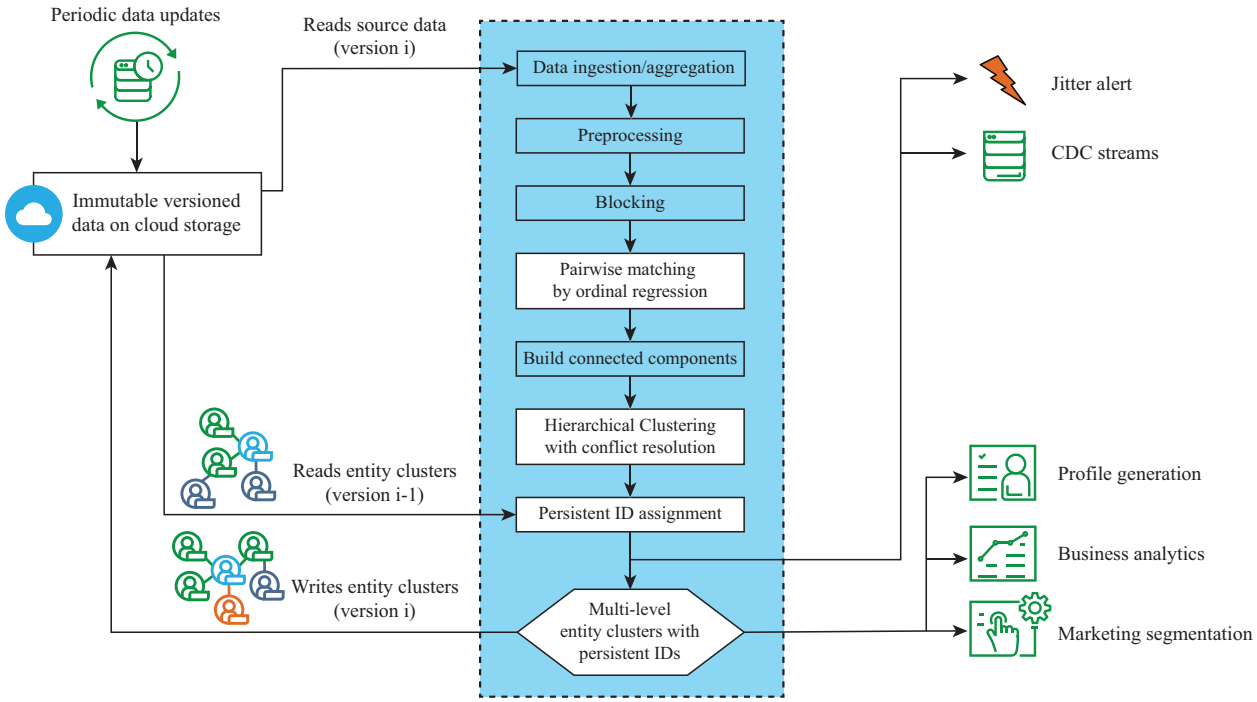

Figure 3: The architecture overview of Fusion together with its upstream and downstream data integration flow.

a match are: `hard-conflict`, `non-conflict`, `weak-match`, `moderate-match`, and `strong-match`. These match levels are defined by an annotation guideline that is comprehensible to a system operator who can select a threshold most appropriate to their downstream application. We validated our annotation guideline by measuring internal annotations agreeing over 95% of the time. Beyond ease-of-use, we also show in Section 6.1 that this formulation has superior performance in predicting ordinal match levels versus a logistic regression approach.

The clustering step first groups record pairs into connected components. A special hierarchical clustering algorithm is applied to separate these components into sub-components (or final clusters in the pipeline) to resolve any detected hard conflicts (formally defined in Section 4). Hence, at the end of the entity matching process, the system has partitioned the input records into a hierarchy of disjoint clusters, with the clustering on each level associated with an ordinal match threshold. Depending on the business use case, different ordinal thresholds can be selected and therefore different versions of entity clusters can be created. Figure 4 illustrates three possible clustering outcomes of the toy example based on different thresholds. More specifically, when `weak-match` (represented by green dotted lines) is configured as the threshold, any record pair with a match-level equal to or stronger than `weak-match` is grouped into the same cluster, unless a `hard-conflict` (represented by black dashed lines) is identified, in which case the clustering algorithm will partition record pairs with `hard-conflict` into separate clusters. When a higher (or more conservative) threshold is selected, more entity clusters are generated, while a lower (or more liberal) threshold usually results in fewer entity clusters; compare the difference between Figure 4 (b1) and (b3).

Although the traditional entity matching task ends with entity clusters, Fusion has an additional step of assigning each entity a persistent entity identifier. The entity identifier is persistent in the sense that it remains in the system as long as the entity still exists, even if the actual cluster composition changes over time. An existing identifier is removed when all the records associated with the entity are deleted completely. Similarly, a new identifier is created when records that represent a new entity appear. We call the process of generating and maintaining the entity identifier "Persistent ID Assignment" (see Section 5). Underlying the Fusion framework is an immutable information model which maintains source data records in full and computes entity resolution results as the derivative of those source data records, as well as prior generations of entity resolution; this is distinct from many industrial systems which collapse (merge) source data in a lossy manner. By storing generations of entity matching results, Fusion is able to maintain a persistent ID for each entity over time and also enables lateral analysis for generations of clustering results.

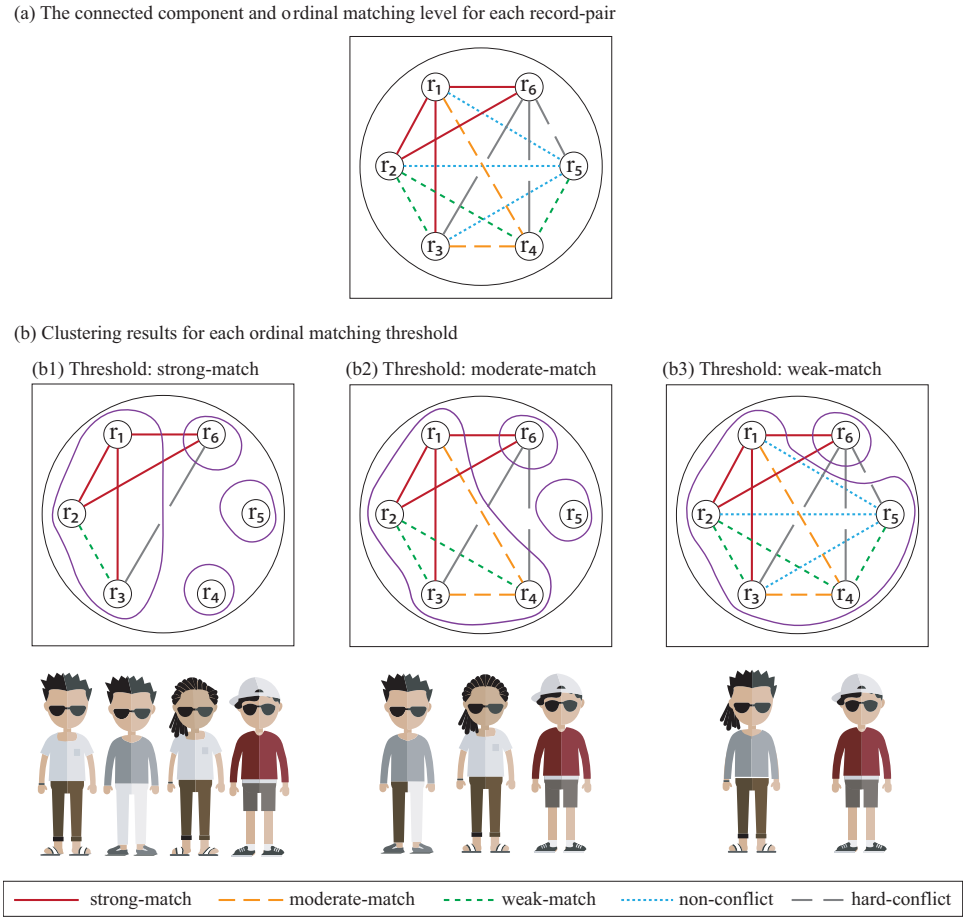

Figure 4: (a) Illustration of the connected component and record pairs with predicted ordinal match levels for the toy example in Table 1; (b) three different clustering outcomes based on three different ordinal match thresholds (b1-b3), which together form a hierarchy of clusterings.

3 FRAMING ENTITY-MATCHING AS AN ORDINAL REGRESSION PROBLEM

Entity-matching systems that leverage machine learning most commonly treat pairwise record matching as a supervised binary classification problem [8, 10], although there are also ternary formulations that add a "potential" match category [26]. As discussed in Section 1.1, many downstream applications benefit from being able to produce clusters with different ordinal match levels.

One way to approach this problem is to use a probabilistic binary classifier, such as logistic regression, and then tune the threshold to achieve the desired precision/recall trade-off. However, this approach has many problems in practice, the foremost being that an operator of the entity matching system needs to be able to tune thresholds for desired outcomes. This can be very challenging for a probabilistic model since the semantics of a pairwise match probability depend heavily on the underlying training data.

In Fusion, we instead use a threshold-based ordinal regression formulation [21] with several pre-defined match levels such as `hard-conflict`, `non-conflict`, `weak-match`, `moderate-match`, and `strong-match` (see Section 3.1). In this approach, the thresholds between match levels are learned, rather than manually tuned or linearly spaced. We also revised the loss function described in [21] to allow for cost sensitivity between match levels to reflect the varying business cost for precision or recall errors.

3.1 Threshold-Based Ordinal Regression

Threshold-based ordinal regression can be viewed as a generalization of binary regression where only two ordinal labels are available. Just like binary regression, we are given a training set $(\mathbf{x}^{(i)}, y^{(i)})_{i=1,\cdots,N}$, where each $\mathbf{x}^{(i)}$ is a feature

vector describing a record pair and $y^{(i)}$ is the target match level for the record pair. The goal is to learn a mapping $z(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ that minimizes the loss function, $\text{loss}(z(\mathbf{x}), y)$, on the target ordinal level y. Different than binary regression, not only the feature weight vector $\mathbf{w}$ is learned, the thresholds associated with boundaries of the ordinal labels are also learned. Here, we use θ_l to represent each threshold, where $\theta_1 < \theta_2 < \cdots < \theta_T$ with $\theta_0 = -\text{inf}$, $\theta_{T+1} = +\text{inf}$. The predicted level is given by which two thresholds the score falls in between, i.e., a score $z(\mathbf{x})$ satisfying $\theta_k < z(\mathbf{x}) < \theta_{k+1}$ predicts the label k.

The cost function for a single sample is given by summing up the loss across all ordinal levels [21]:

$$C(\mathbf{x}, y) = \sum_{l=1}^{T} \text{loss}\left(s(l, y)(\theta_l - z(\mathbf{x}))\right) , \tag{1}$$

where

$$s(i, j) = \begin{cases} -1 & if\ i < j\ , \\ 0 & if\ i = j\ , \\ 1 & if\ i > j\ . \end{cases} \tag{2}$$

Common choices of the loss function include hinge loss, least squared loss, logistic loss, etc., and the different outcomes of them are studied in [21]. In practice, we notice that the choice of the loss function does not have a significant impact on the performance of the classifier, and logistic loss is chosen due to its slightly better performance. Logistic loss is defined as follows:

$$\text{loss}(z) = \log(1 + e^{-z})\ . \tag{3}$$

The loss function when augmented with L2 regularization becomes:

$$\begin{aligned} J(\mathbf{w}, \theta) &= \frac{1}{2}\sum_{i=1}^{N} C(\mathbf{x}_i, y_i) + \frac{\lambda}{2}\|\mathbf{w}\|^2 \qquad (4) \\ &= \frac{1}{2}\sum_{i=1}^{N}\sum_{l=1}^{T} \text{loss}\left(s(l^{(i)}, y^{(i)})(\theta_l - \mathbf{w}^T\mathbf{x}^{(i)})\right) + \frac{\lambda}{2}\|\mathbf{w}\|^2, \end{aligned}$$

where $l^{(i)}$ is the predicted ordinal level for the i-th example, and λ is the regularization parameter.

We note that compared to multi-class classification where all mistakes are treated equal and there is no incentive to prefer one incorrect label over another, in ordinal regression, there is a distinct ordering to the labels, and therefore the solutions are encouraged to minimize the number of crossed thresholds. This is a desirable behavior for our multi-level entity matching problem.

3.2 Cost Sensitivity for Prediction Errors

In the standard threshold-based ordinal regression, although the algorithm penalizes more when a mistake is made across multiple thresholds, predicting a record pair `non-conflict` when it is `hard-conflict` has the same cost as predicting `moderate-match` when it is a `weak-match`. However, in real applications, these types of mistakes often have very different business costs. In Fusion, we introduce an additional weight matrix Γ, with each entry $\gamma_{p,q}$ associated with an error cost for a given (*predicted-level*, *true-level*) combination, (p, q). When Γ is a matrix of ones, it reduces to Eq. (1). These error costs weight each term in the summation over all possibly predicted labels l in Eq. (1):

$$C(\mathbf{x}, y) = \sum_{l=1}^{T} \gamma_{l,y} \cdot \text{loss}\left(s(l, y)(\theta_l - z(\mathbf{x}))\right)\ . \tag{5}$$

When the cost is high associated with the error to a certain ordinal level, we can adjust the weight matrix to further reduce this type of error. For example, if a business needs the precision of `strong-match` to be as high as possible (e.g., when identifying VIP customers), the weight associated with that level should be raised. As we show in Section 6.1, by raising the relevant weight $\gamma_{4,5}$ from 1.0 to 5.0, the precision of `strong-match` is increased from 99.3% to 100%, where 4 and 5 are the level indices associated with `weak-match` and `strong-match`. Γ is an asymmetric matrix. As the examples suggested, when $p < q$, the precision of the level q can be improved by increasing the weight $\gamma_{p,q}$; the recall of the level p can be improved by increasing the weight $\gamma_{q,p}$. In practice, it is usually sufficient to only adjust the weights of adjacent levels such as $\gamma_{4,5}$ and $\gamma_{3,2}$.

4 RESOLVING CLUSTERING CONFLICTS

In this section, we provide a formal definition of cluster-level conflicts (Section 4.1), review a few commonly used clustering algorithms (Section 4.2), and address their limitations in solving these conflicts. Finally, we introduce our hierarchical-based clustering algorithm (Section 4.5).

4.1 Problem Statement

The goal of resolving conflicts in the cluster level is to further partition a connected component into a set of clusters so that there are no hard conflicts in each individual cluster; this is done to ensure a cluster represents a single coherent entity. To provide a definition of cluster-level hard-conflicts, we need to define cluster-level attributes first.

A cluster C in the database D is composed by records $r_1, \cdots, r_n$. Each record has M attributes, or columns. We use $\mathbf{a} = \text{tuple}(a_1, a_2, \cdots, a_M)$ to denote the set of ordered attributes; we use $r_i[a_j]$ to refer to the j-th attribute in the i-th record. The cluster-level attribute for the attribute field j can be defined as follows

$$A_j = \cup r_i[a_j],\ \forall r_i \in C\ . \tag{6}$$

For example, in Figure 2, the BIRTHDATE attribute for Component (a) is {1997-01-02}, and the SSN attribute for Component (b) is {XXX-XX-7090, XXX-XX-1960}. Invalid values including null values are removed.

We use a similarity measure, $S_j(a, a')$, to denote the similarity between two attribute values, a and a', in the j-th attribute field. Commons similarity measures include Levenshtein distance, Jaccard distance, Hamming distance, and many others. This measure can be sensitive to the underlying attribute type. For instance, for the EMAIL attribute, it makes more sense to use Levenshtein distance instead of Jaccard distance to establish similarity. We note that $S_j(a, a')$ composes one type of features in the pairwise classifier, associated with the direct comparison between attribute values from a single field. Other features such as the closeness between an email alias and a name, the popularity of a first name, or the population of a zip code are not covered by it.

We use $\mathrm{S}_{\min}(A_j)$ to denote the weakest similarity between any given two values a_p and a_q in A_j,

$$\mathrm{S}_{\min}(A_j) = \min_{a_p, a_q \in A_j} S_j(a_p, a_q) \ . \tag{7}$$

A hard conflict can be defined as, in any given cluster, the condition where the weakest similarity (or largest dissimilarity) in the attribute field j is less than a threshold t_j, i.e.,

$$\mathrm{S}_{\min}(A_j) > t_j \ . \tag{8}$$

Depending on the use case, different attribute fields can be assigned with different tolerances over dissimilarity, represented by t_j. For example, it is common that a business keeps multiple emails or addresses per customer because large dissimilarity in those attributes does not indicate different entities. On the other hand, one person should have only one birthdate and one social security number. Further tolerances are needed when you consider imprecisely entered information. Allowing for this imprecision, the system should permit one person to have a pair of birthdates like "1960-11-10" and "1960-10-11", but recognize that "1960-11-10" and "1980-02-11" likely come from different individuals. If we normalize the similarity measure to a range between 0.0 and 1.0, an end user can assign a threshold as low as 0.0 to an attribute like EMAIL, and as high as 1.0 to some other attribute like SSN (see the example in Figure 2).

The task of conflict resolution is to partition a dataset into a set of clusters, such that no cluster C has any attribute field A_j whose most dissimilar value pair exceeds the dissimilarity tolerance for that attribute. More formally, this condition can be expressed as:

$$\forall C, A_j, \quad \mathrm{S}_{\min}(A_j) > t_j \ . \tag{9}$$

4.2 Solution Overview

Once we have identified matched record pairs using the ordinal regression matching model in Section 3, we must produce clusters from these record pairs.

Clustering can be as simple as constructing connected components by grouping matched record pairs into entity clusters such that there is no match status for record pairs spanning different entity clusters. However, we can have a situation where two record pairs, (r_i, r_j) and (r_j, r_k) have been classified as matches, but the record pair (r_i, r_k) has been classified as non-match. Through transitivity, the three records r_i, r_j, and r_k are grouped into one connected component, implying the match status of the record pair (r_i, r_k), which contradicts the classification result. Without proper treatment of the transitivity, a connected component can grow into a long chain or lead to an even more degenerate phenomenon, black hole clusters [15, 19].

In FUSION, we first combine matched record pairs into connected components, and then further partition each connected component into smaller clusters to resolve any hard conflicts. The similar procedure has been adopted by many entity matching systems [9, 15, 25], and different choices on the clustering algorithms have been made. For internal completeness, we first review two classical clustering methods that are often used in entity matching systems: correlation clustering [1, 2] and hierarchical clustering [3] as baselines.

However, a non-match record pair can be caused by two different reasons - 1) missing information; or 2) conflicting information (e.g., different SSNs). Some approaches can guarantee a conflict-free cluster, however, they also tend to falsely force the connected component to split when there is no real conflict, which compromises the desirable outcome of transitive closure. In this work, we propose a variant of hierarchical clustering-based algorithm to partition connected components if and only if a *hard conflict* appears. The algorithm works for both binary classification and ordinal regression. We begin by assuming the similarity measure is represented by the estimated match probability returned by a binary classifier, and then we extend the algorithm to an ordinal regression context.

4.3 Correlation Clustering

In correlation clustering [2], the goal is to find a clustering solution that minimizes disagreements or maximizes agreements when contradictory pieces of input information are given. Both objectives are equivalent at the optimum, but they differ as far as the approximation is concerned. In this paper, we focus on the minimization version for discussion.

For each pair of records in the same cluster, there is a penalty on the dissimilarity; for each pair of records in different clusters, there is a penalty on the similarity. The objective is to minimize the sum of the penalties:

$$\mathbf{CC}(L) = \sum_{C \in L, r_i, r_j \in C} \big(1 - M(r_i, r_j)\big) \quad (10)$$

$$+ \sum_{C, C' \in L, C \neq C', r_i \in C, r_j \in C'} M(r_i, r_j)\,, \quad (11)$$

where i and j are any two records in D; $M(i, j)$ represents the pairwise match function between records i and j; L denotes the clustering that partitions D ; C and C' are any two clusters in L; .

A special case for correlation clustering is when the similarity function between two records is changed to binary, i.e., $M(i, j) = 0$ (non-match) or 1 (match). It has been proved that even for this special case, optimal correlation clustering is NP-complete. The best result for the minimization version is the 3-approximation randomized greedy algorithm, known as CC-Pivot Algorithm [1, 2], and it is used as one of our baseline algorithms.

With the goal of minimizing disagreements that occur in a clustering, correlation clustering does not guarantee a conflict-free result; this can be shown by the examples in Figure 2. To simplify the discussion, we treat it as a binary classification problem (i.e., a record pair is a match if and only if the score is higher than 0.5). For the connected component in (a), there is no hard conflict, and the low score between r_2 and r_3 is the result of lacking information. CC-PIVOT can successfully keep r_1, r_2, and r_3 together.[3] However, for the connected component in (b), the algorithm fails to separate r_4 and r_6 apart even if there exists a hard conflict between them.[4] Since CC-PIVOT is a greedy algorithm, depending on which record is selected as the pivot first, the algorithm can actually return three different clustering results - $\{r_4, r_5, r_6\}$, $\{r_4, r_5\} \cup \{r_6\}$, $\{r_4, r_6\} \cup \{r_5\}$, but the chance to return a conflict-free clustering is only one in three.

4.4 Hierarchical Clustering

Hierarchical clustering groups data over a variety of scales by creating a cluster tree or dendrogram. The tree is not a single set of clusters, but rather a multi-level hierarchy, where clusters at one level are joined with clusters at the next level. Having a cluster tree allows us to quickly generate clusterings at different threshold scores without having to rerun an expensive algorithm.

[3]The optimal clustering is $\{r_1, r_2, r_3\}$ because $CC(\{r_1, r_2, r_3\}) = 0.02 + 0.1 + 0.7 = 0.82 < CC(\{r_1, r_2\} \cup \{r_3\}) = 0.02 + 0.3 + 0.9 = 1.22 < CC(\{r_2, r_3\} \cup \{r_1\}) = 0.1 + 0.3 + 0.98 = 1.38$.

[4]The optimal clustering is $\{r_4, r_5, r_6\}$ because $CC(\{r_4, r_5, r_6\}) = 0.05 + 0.1 + 0.8 = 0.95 < CC(\{r_4, r_5\} \cup \{r_6\}) = 0.05 + 0.9 + 0.2 = 1.15$.

Algorithm 1: Hierarchical Clustering

Input:
a set of records, $D = \{r_1, r_2, \cdots, r_n\}$,
a user-defined match function $\hat{M}(C, C')$
a user-defined threshold t
Output: a hierarchy of clusterings L

```
Initialization(D)
  for i = 1 to n do
    C_i = {r_i}
  L = {C_1, C_2, ..., C_n}
BuildClusters(L)
  while L.size > 1, max M̂(C, C') >= θ do
    C, C' = argmax_{C,C'∈L} M̂(C, C')
    remove C and C' from L
    add the new cluster C ∪ C' to L
```

Additionally, hierarchical clustering gives us the flexibility to tailor the algorithm to a specific problem by choosing a similarity function. The commonly used ones include single-link clustering, complete-link clustering, and group-average clustering. In single-link clustering, the similarity between clusters is defined as the highest similarity among any cluster pairs (i.e., the similarity between the nearest neighbors); in complete-link clustering, it is the lowest similarity among the cluster pairs (i.e., the similarity between the farthest neighbors); in group-average clustering, the similarity measure becomes the average similarity among all cluster pairs. We use $\hat{M}_{\max}$, $\hat{M}_{\min}$, and $\hat{M}_{\text{avg}}$ to denote the similarity functions in single-link, complete-link, and group-average clusterings, respectively, and we use the notation $\hat{M}(C, C')$ to distinguish from the similarity function between two records, $M(r, r')$.

Hierarchical clustering works as follows: Given a similarity function, hierarchical clustering initially places every record in its own singleton cluster. Then, at every step the two clusters that are most similar are merged until a stopping criterion is satisfied, usually when the highest similarity score drops below a threshold θ (see Algorithm 1).

However, none of the baseline approaches discussed above can successfully partition clusters when a conflict is detected and at the same time maintain the transitive closure when there is no hard conflict. In correlation clustering, minimizing the penalty function associated with the global disagreement is not entirely aligned with minimizing hard conflicts. In hierarchical clustering, single-link clustering exhibits conflict ignorance behavior, complete-link clustering focuses on conflict avoidance, and global-average clustering behaves somewhere in between. None of them can achieve the conflict resolution goal we stated above. The clustering results

of the examples in Figure 2 using different similarity functions are compared in Table 2.

Example	Single-Link ($\hat{M}_{\max}$)	Group-Average ($\hat{M}_{\text{avg}}$)	Complete-Link ($\hat{M}_{\min}$)
(a)	$\{r_1, r_2, r_3\}$	$\{r_1, r_2, r_3\}$	$\{r_1, r_2\} \cup \{r_3\}$
(b)	$\{r_4, r_5, r_6\}$	$\{r_4, r_5, r_6\}$	$\{r_4, r_5\} \cup \{r_6\}$

Table 2: Comparison of the clustering results from applying three different hierarchical clustering algorithms to the two connected component examples in Figure 2.

4.5 A Special Hierarchical Clustering with the Conflict Resolution Property

We propose a novel variant of the hierarchical clustering framework with a new similarity function denoted as $\hat{M}_{\text{fusion}}$, and defined as follows:

$$\hat{M}_{\text{fusion}}(C, C') = M(\mathbf{A}, \mathbf{A}') \,, \tag{12}$$

where $\mathbf{A}$ and $\mathbf{A}'$ are the tuples of cluster-level attributes, i.e.,

$$\mathbf{A} = \text{tuple}(A_1, A_2, \cdots, A_M) \,, \tag{13}$$

and each cluster-level attribute is constructed by the union of record attributes in that cluster (see Eq. (6)). In FUSION the classifier for record pairs is already generalized to handle multiple values per attribute field, and is learned to reject hard conflicts on the pairwise level, s.t.,

$$\exists\, \mathrm{S}_{\min}(A_j) < t_j \rightarrow M(r, r') < \theta \,, \tag{14}$$

where θ is the binary classifier threshold or one of the ordinal classifier thresholds, and t_j is the minimally required similarity for the attribute j. By extending the pairwise classifier M to the cluster level (i.e., scoring $\mathbf{A}$ and $\mathbf{A}'$), we can detect cluster-level conflicts. In each merge operation of hierarchical clustering, $M(\mathbf{A}, \mathbf{A}')$ associated with the cluster pair C and C' is evaluated. If it is below the classifier threshold θ for all cluster pairs, the algorithm terminates. Therefore, we can guarantee that no hard conflict exists in final clusters, and we can keep records connected as far as the classifier allows. This way, we also maintain the logical consistency between pairwise matching and clustering.

Computationally, $\hat{M}_{\text{fusion}}$ is more expensive than $\hat{M}_{\max}$, $\hat{M}_{\text{avg}}$, and $\hat{M}_{\min}$ because a simple operation on the pairwise classifier scores is not enough, and the evaluation on the similarity of cluster-level attributes is required. Fortunately, since the attribute similarity for each record pair has already been calculated in the pairwise classification step, the minimum of the cluster-level attribute similarity can be simplified to a `min` operation, i.e.,

$$S_{\min}(A_j \cup A'_j) = \min_{r \in C, r' \in C'} S_j(r[a_j], r'[a_j]) \,. \tag{15}$$

Clustering Algorithm	Example (a)	Example (b)
CC	Succeed	Fail
HC with $M_{\max}$	Succeed	Fail
HC with M_{avg}	Succeed	Fail
HC with $M_{\min}$	Fail	Succeed
HC with M_{fusion}	Succeed	Succeed

Table 3: The summary of results by applying different clustering algorithms to the examples in Figure 2.

Since $S_j(r[a_j], r'[a_j])$ is cached in the previous process, expensive computations such as Levenshtein distance of attribute values can be avoided.

Table 3 shows the comparison of clustering results using different clustering algorithms based on the same example in Figure 2. Figure 5 illustrates the procedure and the outcome of applying our clustering algorithm to the toy example in Table 1. The three versions of clusterings can be generated given three different thresholds, and we can see the hard-conflicts between (r_3, r_6), (r_4, r_6), (r_5, r_6) do not exist in any of the final clusterings.

- `strong-match`: $\{r_1, r_2, r_3\}, \{r_4\}, \{r_5\}, \{r_6\}$;
- `moderate-match`: $\{r_1, r_2, r_3, r_4\}, \{r_5\}, \{r_6\}$,
- `weak-match`: $\{r_1, r_2, r_3, r_4, r_5\}, \{r_6\}$.

5 ASSIGNING CLUSTER IDENTIFIERS

Entity matching never exists in isolation and most of the time it needs to be integrated and synchronized with downstream data management systems. Typically these systems will accept new information on a daily or real-time basis which will in turn trigger the re-clustering or updating of existing entity clusters. Downstream applications of entity resolution will typically need ways to "refer" to a canonical entity even though our beliefs about which records constitute an entity may change over time. This creates several problems which must be addressed to successfully utilize entity matching in a practical setting:

- How to create a unique identifier to track each real-world entity even if the cluster of records associated with the entity changes over time?
- How to identify if an entity (not a record) is added to or deleted from the system?
- How to provide a single metric to evaluate the disparity between two clustering results and, with that, monitor system stability?

In this section, we explain how Amperity's ID assignment process works and how it can help to answer the questions above. First, we provide a definition of the cluster ID assignment problem.

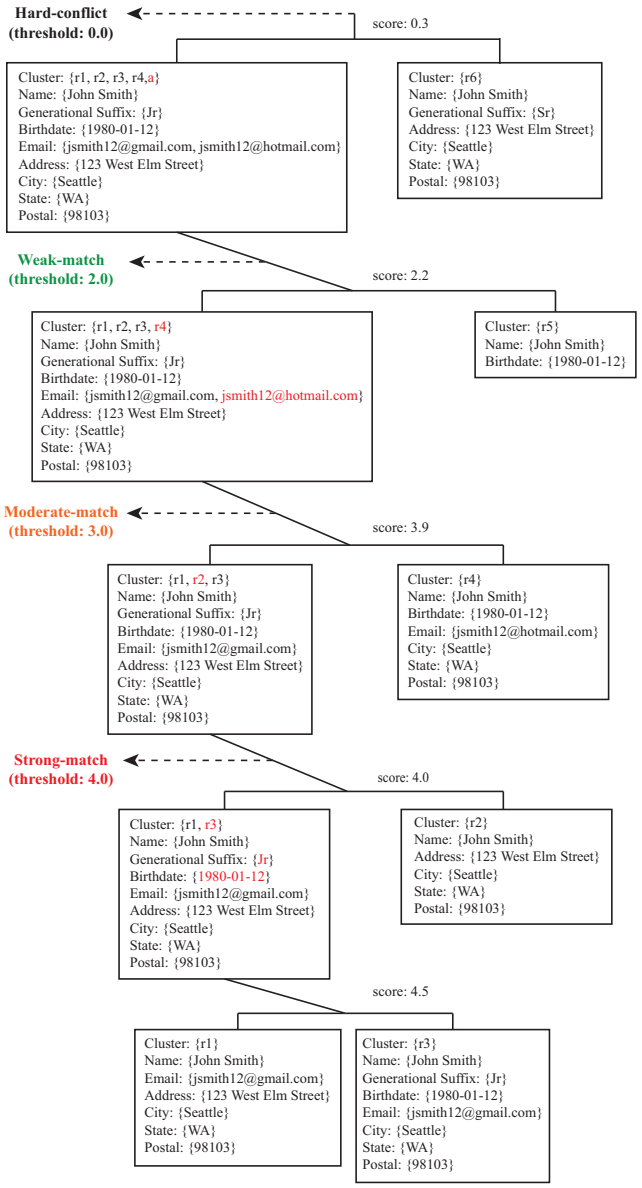

Figure 5: Illustration of the clustering process by applying the proposed clustering algorithm to the toy example in Table 1. The merged values in each step are highlighted in red.

5.1 Problem Statement

Cluster ID assignment is a mapping problem between clusterings from two generations (e.g., two different timestamps) of entity matching results.[5] A *clustering* L at a given generation can be defined as a partition of a database D into clusters $C_1, C_2, \cdots, C_K$ that form a non-overlapping cover over D; more formally,

$$C_k \cap C_j = \emptyset \quad \text{and} \quad \cup_{k=1}^{K} C_k = D, \quad \forall\, k, j \in \{1, \cdots, K\}\,. \tag{16}$$

Let the number of records in D and in cluster C_k be n and n_k, respectively, and we have $n = \sum_{k=1}^{K} n_k$. Similarly, we use $L' = \{C'_1, \cdots, C'_{K'}\}$ to denote a clustering from the next generation, and we have, $\cup_{k'=1}^{K'} C'_{k'} = D'$ and $n' = \sum_{k'=1}^{K'} n'_{k'}$, where D' represents an updated version of D.[6]

The cluster ID assignment process can be interpreted as follows: for each cluster in L', deciding which cluster in L it is most closely related to. Intuitively, we know we want the number of intersecting records of the two clusters to be small. Here we use $n_{kk'}$ to denote this number

$$n_{kk'} = |C_k \cap C'_{k'}|\,. \tag{17}$$

An optimal mapping σ^* should maximize the total number of intersecting records between L and L' such that

$$\sigma^*(k) = \text{argmax}_\sigma \sum_{k=1}^{K} n_{k,\sigma(k)}\,, \tag{18}$$

where $\sigma(k)$ is a mapping of $\{1, \cdots, K\}$ into $\{1, \cdots, K', \text{NULL}\}$, where NULL is a special symbol denoting there is no analogue of the input cluster. Assuming L is from an earlier generation than L', when assigning cluster IDs for L', we can use the mapping σ^* to decide which previous cluster IDs the new clusters should inherit.

With this clustering mapping, we are able to answer some other important questions such as which customers have left the system, which have just joined, which are existing customers, and among the existing customers, which have their information updated. Each question is associated with one type of clusters, defined as follows:

- If two clusters $C_k \in L$ and $C'_{k'} \in L'$ are equal, then we say that $C'_{k'}$ is **unchanged**.
- A cluster $C'_{k'} \in L'$ is an **updated cluster** if $\exists\, \sigma^*(k) = k'$, but $C_k \neq C'_{k'}$.
  Here, the mapping between C_k and $C'_{k'}$ is established, but the two clusters are not identical. This can be caused by the splitting of an old cluster into multiple clusters; merging an old cluster with other clusters; or both. Because C_k and $C'_{k'}$ are not perfectly aligned, we also call them **partially-aligned clusters**.
- A cluster $C_k \in L$ is a **deleted cluster** if $\sigma(k)$ is null.
  An entity is only considered as deleted if all the records associated with that entity are deleted in the database. As a result, there is no matching cluster in the new clustering L'.

[5] The concepts of assigning cluster IDs and assigning entity IDs are interchangeable because at the end of Fusion each entity is represented as a cluster of records.

[6] We borrow the notations used in [17] for the clustering definition.

- A cluster $C'_{k'} \in L'$ is **new** if $\sigma^{-1}(k')$ is null, where $\sigma^{-1}(k')$ denotes the inverse of $\sigma(k)$, the mapping from $\{1, \cdots, K'\}$ into $\{1, \cdots, K\}$,

Since partially-aligned clusters represent the updated entities, the ratio between them and the total cluster volume can be used as an indicator for data stability. In Fusion, we use a metric derived from Eq. (18), named as partition distance or classification error [17, 22], to monitor this difference between two clusterings. It is defined as follows

$$d(C, C') = 1 - \frac{1}{n} \max_{\sigma} \sum_{k=1}^{K} n_{k,\sigma(k)} \,. \tag{19}$$

In the next section we present our algorithm to finding this optimal mapping between two generations of clusterings in order to solve the clustering ID assignment problem.

5.2 Algorithm

From a computational perspective, it is prohibitive to enumerate all possible mappings $\sigma(\cdot)$, ruling out a brute force search. Fortunately, the problem is identical to finding the maximum-weight matchings in bipartite graphs, sometimes also called the assignment problem [5]. In this context, a bipartite graph can easily be represented by a *confusion matrix*, where the entries of the matrix are the number of intersecting records between two clusters, i.e., $n_{kk'}$ in Eq. (17). This assignment problem can be solved using the well-known Hungarian algorithm with an $O(n^3)$ running time. However, this is still too expensive since n is on the scale of hundreds of millions of clusters for a given business.

In Fusion, we implement a greedy approximation algorithm to achieve much faster performance (see Step 4 in Algorithm 2). Although an optimal solution is not guaranteed, the computational complexity is linear, and this approach has shown to be a good approximation [16]. Through empirical observation in our applications, we find that the ratio between the number of partial-aligned clusters and the total number of clusters is very small and the greedy algorithm solution is often very close to the optimal solution. With its linear performance and our distributed algorithm implementation in Spark, in most cases, the ID assignment process can complete within ten minutes (see Section 6.3 for the experimental results).

Our cluster ID assignment algorithm is presented in Algorithm 2. We also use a toy example in Figure 6 to explain how the algorithm works. Figure 6 (a) compares the two clusterings from the (i-1)-th and i-th runs. Before the IDs are assigned to the newer clustering, temporary IDs are created for easy reference. Between the two clusterings, we have unchanged clusters a-1 and b-1, a deleted cluster a-5, a new cluster b-5, and partially-aligned clusters including a-2, a-3, a-4 from the (i-1)-th run and b-2, b-3, b-4 from i-th run. A 3×3 confusion matrix is built based on the partially-aligned clusters. By running the greedy assignment algorithm, the mapping between these partially-aligned clusters is constructed, i.e., (a-2, b-2), (a-3, b-4), and (a-4, b-3). Finally, we replace the temporary IDs, b-1, b-2, b-3, and b-4, with the existing IDs, a-1, a-2, a-4, and a-3, respectively, Since b-5 is identified as a new cluster, it keeps its newly created ID. At the end of the ID assignment process, we also generate a summary of changed data, at both the cluster and record level. It provides insights and metrics for the business, and more importantly, helps the downstream systems to synchronize with the change and stabilizes the overall data flow.

Algorithm 2: Cluster ID Assignment

Input: $L = \{C_1, \cdots, C_k\}$ and $L' = \{C'_1, \cdots, C'_{k'}\}$
Output: $id(C'_{k'}),\ \forall C'_{k'} \in L'$

Step 1: Only keep partially-aligned clusters by removing new-clusters from L', deleted-clusters from L, and unchanged-clusters from both L and L'.
Step 2: Update the mapping between the unchanged-clusters in L and L', i.e., $\forall\ C_k = C'_{k'}$, make $\sigma^*(k) = k'$.
Step 3: Build a confusion matrix M for the remaining partially-aligned clusters in L and L', assigning $n_{kk'}$ as the entry value.
Step 4: Run GREEDY(M)

```
E = SORT(M)    // Sort the entries of M in desc order
while E ≠ null do
    E_max = E[0]
    (i, j) = LOCATENODE(E_max, M)    // E_max = M_{i,j}
    σ*(i) = j
    DELETENODEBYROW(E, i)    // Delete all entries in E associated with i-th row in M
    DELETENODEBYCOLUMN(E, j)    // Delete all entries in E associated with j-th column in M
Return σ*
```

Step 5: Assign cluster IDs:

1. Unchanged- and updated-clusters: $id(C'_{\sigma^*(k)}) = id(C_k)$.
2. New-cluster:
   A new ID is created. ID collision is performed to ensure there are no duplicated IDs in system.
3. Deleted-cluster:
   The ID is removed from the current and downstream systems.

(a) Clusterings from two different generations

ID	Clusters in (i-1)-th run
a-1	{1}
a-2	{2}
a-3	{3, 4, 5, 6, 7}
a-4	{8}
a-5	{9, 10}

Temp ID	Clusters in i-th run
b-1	{1}
b-2	{2, 3}
b-3	{4}
b-4	{5, 6, 7, 8}
b-5	{11}

(b) Persistent ID assignment process

Step 1: Remove the unchanged-cluster, new-cluster, and deleted-cluster

a-2	{2}
a-3	{3, 4, 5, 6, 7}
a-4	{8}

vs

b-2	{2, 3}
b-3	{4}
b-4	{5, 6, 7, 8}

Step 2: Build the map between the unchanged-cluster pair

a-1 <=> b-1

Step 3: Build confusion matrix

	b-2	b-3	b-4
a-2	1	0	0
a-3	1	1	3
a-4	0	0	1

Step 4: Find the optimal matching

	b-2	b-3	b-4
a-2	1	0	0
a-4	0	0	1
a-3	1	1	3

Temp ID	Curr ID	Entity Status
b-1	a-1	Unchanged
b-2	a-2	Updated
b-3	a-4	Updated
b-4	a-3	Updated
b-5	b-5	Added
n/a	a-5	Deleted

Entity-level change

Record	Prev ID	Curr ID	ID Status
1	a-1	a-1	
2	a-2	a-2	
3	a-3	a-2	Changed
4	a-3	a-4	Changed
5	a-3	a-3	
6	a-3	a-3	
7	a-3	a-3	
8	a-4	a-3	Changed
9	a-5	n/a	Deleted
10	a-5	n/a	Deleted
11	n/a	b-5	Added

Record-level change

Figure 6: A toy example for Algorithm 2.

6 EXPERIMENTAL RESULTS

6.1 Classification Results

In this section, we empirically show that ordinal regression outperforms logistic regression when predicting fixed ordinal match levels. In order to compare with binary logistic regression, we discretize the probabilistic output range (e.g, $[0, 1]$) into multiple intervals matching the number of match levels; we optimized the thresholds between intervals by maximizing the F_1 score for each match level. In other words, we tried to find the "best" F_1 partition of the $[0, 1]$ range to map to match levels.

Since the focus of this work is on phenomena more common in applied settings, we used Amperity internal data to train and evaluate our system. The experimental dataset contains a collection of record pairs sampled from several of the customer databases of Amperity's clients. To alleviate the manual labeling effort without compromising the generality of this approach, we sampled 4,389 record pairs with each record pair having a distinct feature vector. These record pairs are manually labeled with one of the five ordinal classes. [7] To evaluate the model performance, we use both mean absolute error (MAE) and zero-one errors (ZOE).[8]

Table 4: Comparison of mean absolute error (MAE) and zero-one error (ZOE) on a customer dataset using the discretized binary regression and the threshold-based ordinal regression (used in this system). For these metrics, lower is better.

	LogReg	OrdReg	Error Reduction
MAE	0.781	0.396	49.2%
ZOE	0.473	0.236	51.0%

Table 5: Examples to demonstrate the efficacy of the weighted adjustment strategy in the ordinal regression. The Class-5 (`strong-match`) precision P_5 and class-3 (`weak-match`) recall (R_3) are compared using both the discretized binary regression and the ordinal regression with different weight.

	LogReg	OrdReg			
		γ=1	$\gamma_{3,2}$=10	$\gamma_{4,5}$=5	$\gamma_{3,2}$=10, $\gamma_{4,5}$=5
P_4	0.488	0.993	0.989	1.00	1.00
R_2	0.963	0.958	0.984	0.956	0.985

Table 4 shows that the threshold-based ordinal regression reduces error by 50% compared to the discretized logistic regression when trained on the same feature representation. Table 5 compares regular ordinal regression, discretized logistic regression, and ordinal regression with the weight-adjustment (see Section 3.2) to demonstrate its capacity to improve the precision or recall of a certain ordinal class.

6.2 Clustering Results

To evaluate the conflict-resolution property of our clustering algorithm, we apply it to a dataset constructed by 1M real customer records, and compare its clustering result to four other clustering algorithms - correlation clustering, hierarchical clusterings with single-link, group-average, and

[7] The match level distribution of the samples: 1165 (`non-match`), 223 (`non-conflict`), 462 (`non-conflict`), 831 (`moderate-match`), 1708 (`strong-match`) All models were trained and evaluated with the same 60/40 train/test split.

[8] MAE = $\sum_{i=1}^{n} |y_i - z_i| / n$, and ZOE = $\sum_{i=1}^{n} \ell_{0/1}(y_i, z_i)/n$, where $\ell_{0/1}(y, z)$ equals 1 when $y = z$, and 0 otherwise.

Table 6: Comparison of clustering results: correlation clustering (CC), hierarchical clustering with the single-link (HC_{max}), group-average (HC_{avg}), complete-link (HC_{min}), and our algorithm (HC_{fusion}). The over-clustering rate r_{oc} denotes the ratio of clusters that contain more than one true entity, which is associated with the failure to resolve cluster conflicts. Similarly, the under-clustering rate is the ratio of clusters that represent the same true entity but got falsely split apart. We also evaluate the results using the cluster-level precision, recall, and F_1 defined in [18].

Measure	CC	HC_{max}	HC_{min}	HC_{avg}	HC_{fusion}
r_{oc}(%)	14.03%	4.18%	0.97%	0.00%	0.00%
r_{uc}(%)	27.75%	0.00%	37.44%	41.40%	0.00%
Precision	0.57	0.95	0.45	0.41	1.00
Recall	0.72	0.90	0.62	0.58	1.00
F_1	0.64	0.93	0.52	0.48	1.00

complete-link settings. The dataset is processed through our entity-matching pipeline Fusion with `weak-match` chosen as the matching threshold. Among the 1M records, 188,512 record pairs are classified as `weak-match` or higher ordinal match levels. Based on these matched pairs, 471,644 connected components are established. It is important to note that the cluster-level conflicts only arise in clusters of size larger than 2. So to simplify the comparison, we only keep connected components with the size larger than 2, which reduces the number of components to 215. These connected components are manually checked for conflicts, and partitioned into separate clusters when a conflict is detected, with the final results saved as the ground-truth clustering.

Several metrics were used to evaluate the cluster-level performance such as the over-clustering rate r_{oc}, the under-clustering rate r_{uc}, as well as the cluster-level precision, recall and F_1 [18]. As shown in the toy example, HC_{max} never demonstrates the under-clustering issue while having the highest over-clustering rate; HC_{min} takes the most conservative approach because it never over-clusters and has the highest under-clustering rate. The results of HC_{avg} and CC fall somewhere in between. Our algorithm, HC_{fusion}, however, outputs identical results as the ground truth clustering.

6.3 Cluster ID Assignment Results

The cluster ID assignment algorithm has been applied to the daily entity matching jobs for over 25 businesses. For the simplicity of this discussion, we show the results from five of them across five different industries, with data collected from the last 30 days. We use n_r and n_c to denote the number of total records and the number of total clusters present in the last 30 days, and use r_{churn} to represent the cluster ID churn rate (or reassignment rate) without our algorithm, aggregated from the data in the last 30 days. We assume that, without any process to relate the cluster IDs from previous generations, whenever a cluster composition changes a new cluster ID has to be created even if the cluster still represent the same underlying entity (new clusters are excluded from this calculation). t_{avg} shows the average time spent on the cluster ID assignment process. In Table 7, we can see that the reassignment rate varies significantly across different datasets (ranging from 0.1% to 12%), and it is usually related to the volume of daily data updates. The process usually finishes within 10 minutes, and takes longer with the data volume and reassignment rate increase.

Table 7: Cluster ID assignment results for five different datasets across five different industries, with the data aggregated from the last 30 days. The cluster ID churn rate n_c will keep going up without cluster ID assignment algorithm.

Datasets (Industry)	n_r (MM)	n_c (MM)	r_{churn} (%)	t_{avg} (min)
A (Restaurant)	14.50	12.13	0.12	1.71
B (Sports)	395.08	58.33	12.40	4.80
C (Airline)	162.92	84.52	1.78	6.63
D (Hospitality)	205.91	131.49	1.97	9.37
E (Retail)	364.00	352.70	1.06	9.09

7 CONCLUSION

In this paper, we introduce Amperity's entity matching framework, Fusion, which formulates the entity matching task as an ordinal regression problem and, with a single trained model, can generate different versions of entity clusterings to fit a variety of business applications. We also design a novel clustering algorithm that can resolve conflicts and maintain consistent data quality at the cluster-level as the pairwise predictive model. For each version of the clustering, we perform a special ID assignment algorithm to assign each cluster a traceable identifier that continues to track the entity in the presence of data updates. In the experimental results, we show that ordinal regression cannot be replaced by binary classification in predicting ordinal variables. We also show that, compared to several popular clustering algorithms, our clustering algorithm is the only one that can prevent both over-clustering and under-clustering when resolving cluster-level conflicts.

REFERENCES

[1] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.

[2] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 56(1-3):89–113, 2004.

[3] Mikhail Bilenko, S Basil, and Mehran Sahami. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[4] Mikhail Bilenko, Beena Kamath, and Raymond J Mooney. Adaptive blocking: Learning to scale up record linkage. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 87–96. IEEE, 2006.

[5] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

[6] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 601–610, 2014.

[7] Xin Luna Dong and Divesh Srivastava. *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.

[8] Ivan P Fellegi and Alan B Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[9] Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. Incremental record linkage. *Proceedings of the VLDB Endowment*, 7(9):697–708, 2014.

[10] Thomas N Herzog, Fritz J Scheuren, and William E Winkler. *Data quality and record linkage techniques*. Springer Science & Business Media, 2007.

[11] H. V. Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M. Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Commun. ACM*, 57(7):86–94, 2014.

[12] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[13] Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[14] Katie Kalm, Rebecca Scully, Aria Haghighi, and Hilary Fagan. A report on the error rates of customer misidentification and why this is so bad for your business. *Amperity Whitepaper*, 2019.

[15] Hakan Kardes, Deepak Konidena, Siddharth Agrawal, Micah Huff, and Ang Sun. Graph-based approaches for organization entity resolution in mapreduce. *Proceedings of TextGraphs-8 Graph-based Methods for Natural Language Processing*, pages 70–78, 2013.

[16] Garrett Lewwen. A greedy approximation algorithm for the linear assignment problem, Mar 2017.

[17] Marina Meila. Comparing clusterings: an axiomatic view. In *Proceedings of the 22nd international conference on Machine learning*, pages 577–584. ACM, 2005.

[18] David Menestrina, Steven Euijong Whang, and Hector Garcia-Molina. Evaluating entity resolution results. *Proceedings of the VLDB Endowment*, 3(1-2):208–219, 2010.

[19] Matthew Michelson and Sofus A Macskassy. Record linkage measures in an entity centric world. In *Proceedings of the 4th workshop on Evaluation Methods for Machine Learning*, 2009.

[20] Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[21] Jason DM Rennie and Nathan Srebro. Loss functions for preference levels: Regression with discrete ordered labels. In *Proceedings of the IJCAI multidisciplinary workshop on advances in preference handling*, pages 180–186. Kluwer Norwell, MA, 2005.

[22] Giovanni Rossi. Partition distances. *arXiv preprint arXiv:1106.4579*, 2011.

[23] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning and data mining*. Springer Publishing Company, Incorporated, 2017.

[24] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using deepdive. *PVLDB*, 8(11):1310–1321, 2015.

[25] Csaba István Sidló, András József Molnár, Gábor Lukács, and András Benczúr. Theoretical foundations of entity resolution models. In *ANNALES UNIVERSITATIS SCIENTIARUM BUDAPESTINENSIS DE ROLANDO EOTVOS NOMINATAE SECTIO COMPUTATORICA*, volume 43, pages 39–56. ELTE, 2014.

[26] Vassilios S Verykios, George V Moustakides, and Mohamed G Elfeky. A bayesian decision model for cost optimal record matching. *The VLDB Journal?The International Journal on Very Large Data Bases*, 12(1):28–40, 2003.

[27] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016.