Integrating User Feedback under Identity Uncertainty in Knowledge Base Construction

Anonymous authors

Abstract

Users have tremendous potential to aid in the construction and maintenance of knowledges bases (KBs) through the contribution of feedback that identifies incorrect and missing entity attributes and relations. However, as new data is added to the KB, the KB entities, which are constructed by running entity resolution (ER), can change, rendering the intended targets of user feedback unknown–a problem we term *identity uncertainty*. In this work, we present a framework for integrating user feedback into KBs in the presence of identity uncertainty. Our approach is based on having user feedback participate alongside mentions in ER. We propose a specific representation of user feedback as *feedback mentions* and introduce a new online algorithm for integrating these mentions into an existing KB. In experiments, we demonstrate that our proposed approach outperforms the baselines in 70% of experimental conditions.

1. Introduction

Structured knowledge bases (KBs) of entities and relations are often incomplete and noisy, whether constructed by hand or automatically. For example, it has been reported that 71% of people in Freebase are missing a place of birth attribute and 75% of people have no known nationality [Dong et al., 2014a]. Similarly, while YAGO2 has 95% accuracy of its facts, this means there are roughly 22 million incorrect facts out of YAGO2's 447 million facts involving 9.8 million entities [Hoffart et al., 2013]. Vast research in cleaning and correction of databases is evidence of the permeation of errors throughout the construction of KBs in multiple domains [Dong et al., 2014a, Li et al., 2017, Wang et al., 2015, Dong et al., 2014b]. As the primary consumers of KBs, and the applications reliant on KB data, human users have significant potential to aid in KB construction and maintenance.

From a user's standpoint, a KB contains a set of entities that possess attributes and participate in relationships with other KB entities. However, the data that gives rise to a KB is a collection of raw evidence, which can be understood as *mentions*, requiring clustering by entity resolution (ER) into a set of *inferred entities*. The attributes and relations of the KB entities with which the user interacts are drawn from this underlying clustering of the

mentions. Thus, errors that appear to users as spurious attributes and relationships among KB entities, may stem from a variety of sources, including: noisy mentions produced by information extraction, mistakes in ER, missing data, etc.

In light of new data that is continually being added to a KB, inferred KB entities (i.e., the underlying clustering of the mentions) may change, resulting in new KB entities, attributes and relationships. Modifying the set of inferred KB entities poses a formidable challenge to the task of integrating user feedback with KB content in that it introduces *identity uncertainty*, i.e., since the inferred KB entities may change as a result of new data being add to the KB, the precise target of any piece of user feedback is uncertain. As an example, consider a KB of scientists presenting the scientist's topical expertise, affiliation history, scientific collaborations and previous workall gathered from research papers and other mentions of the scientists. While interacting with the KB, a user notices the entity Fernando Pereira and submits feedback asserting that Fernando Pereira *collaborated with a scientist named John Blitzer*. Later, another user notices an error in Fernando Pereiras affiliation history. The error stems from ER incorrectly merging new mentions of a *second* Fernando Pereira with the mentions in the Fernando Pereira KB entity. Attempting to correct the error, which manifests as an incorrect affiliation, the user submits feedback claiming that Fernando Pereira *who is affiliated with the organization Google was never affiliated with a Instituto de Telecomunicaes in Portugal.* This second piece of feedback signals to ER that Fernando Pereira should be split into at least 2 entities. But, after splitting Fernando Pereira, the target of the *first* piece of user feedback is unknown.

In this paper, we present a new framework for reasoning about user feedback under identity uncertainty. Our approach is founded on the idea that user feedback should participate in ER alongside standard mentions. That is, when an inferred entity that was previously the target of user feedback is split–as in the example above–the decision about the new target of the feedback should be made by the ER algorithm. Including feedback in ER, rather than applying feedback to the inferred entities as a post-hoc step, is powerful: the feedback provides additional evidence for mergers and splits, making ER more robust and effective.

It may seem that representing user feedback as *additional mentions* is a sufficient approach, however this may result in additional ER errors. In the example above, if the second piece of feedback was represented as a mention that was incompatible with KB entities working in Portugal, ER would not have associated the feedback with the intended target (who, at the time the feedback was supplied, was affiliated with the university in Portugal, according to the KB). Therefore, we advocate for the representation of user feedback as *feedback mentions* (FMs), that include two components: a *packaging* and a *payload*. The packaging contains attributes used by ER to initially determine a set of mentions similar to the feedback. After this initial set is discovered, attributes contained in the payload are applied, introducing missing attributes, correcting spurious attribute inferences and influencing future ER decisions. In the example above, a FM with packaging claiming an affiliation with Google would help to guide ER to associate the feedback with the mentions that constitute KB entity Fernando Pereira. Afterward, the FMs payload, which includes the incompatibility with the university in Portugal helps ER to correctly split the mentions into two inferred entities.

User feedback (and additional data) naturally arrives over time, rather than all at once, and thus demands an incremental integration scheme. In this work, we also present a new incremental ER algorithm for integrating new mentions and FMs with an existing KB. Our algorithm builds a hierarchical clustering of the data, one data point at a time. The crux of the algorithm is a `graft` subroutine, which allows new data to trigger cascading mergers of disparate parts of the tree to produce new inferred entities and split existing inferred entities.

Previous work studies user feedback for ER, but is focused on user supplied, mention-level pairwise constraints, which are insufficient for providing feedback about KB entities, their attributes and relations. For example, it is impossible to supply missing attributes or correct for noise in the underlying mentions with pairwise constraints. Similarly, many types of desirable user feedback–as in the example above–are inexpressible in the language of pairwise constraints. Collecting pairwise feedback is also a burden: it may require the creation of specialized UIs, can be costly, and demands specialized strategies for selecting which pairs to label and in what order [Wang et al., 2012, 2013, Firmani et al., 2016].

Although user feedback can be used to correct arbitrary KB errors, in this paper we focus on leveraging user feedback for correcting mistakes in ER. We conduct two experiments in the context of *author disambiguation*–a particular instantiation of ER. In the first, we automatically generate user feedback that includes the areas of expertise, represented as a set of keywords, for various authors. In the second experiment, we generate user feedback that identifies missing and incorrectly attributed publications with respect to a set of currently inferred KB entities. We propose 3 baseline approaches for integrating user feedback and 2 feedback simulation schemes and measure the number of pieces of feedback required to recover the ground-truth entities under each experimental setting. Our results show that our proposed approach based on FMs outperform the baselines in 70% of experimental conditions. Our work initiates the investigation of user feedback integration amidst identity uncertainty in KBs, an under-explored and important problem whose solution would dramatically improve the effectiveness of users in the process of KB construction and maintenance.

2. User Feedback & Identity Uncertainty in KBs

Our goal is to construct a framework for integrating user feedback under identity uncertain in knowledge bases. In this section we provide formal models for mentions, entities, user feedback and identity uncertainty, which serve as building blocks for the remainder of our discussion.

2.1 Formal Model of Mentions, Entities and Attributes

Formally, we assume that the KB is comprised of a set of mentions $\mathcal{M} = \{x_0, \cdots, x_n\}$ which refer to a set of ground truth entities $\mathcal{E}^\star = \{e_0^\star, \cdots, e_k^\star\}$. Each mention, x_i, corresponds to exactly 1 ground-truth entity, denoted $e^\star(x_i)$. The goal in ER is to construct a partition of the mentions, $\hat{\mathcal{E}} = \{\hat{e}_0, \cdots, \hat{e}_l\}$ as similar to $\mathcal{E}^\star$ as possible. Each $\hat{e} \in \hat{\mathcal{E}}$ is known as an *inferred entity*.

Mentions and entities (ground-truth and inferred) are comprised of *attributes*. Each mention, $x \in \mathcal{M}$, has a corresponding set of attributes, $\mathcal{A}(x) = \{a_0, \cdots, a_m\}$. Any sub-

set of mentions (including inferred entities), e, also has a corresponding set of attributes, $\mathcal{A}(e)$, that is derived from its underlying mentions in a process called *canonicalization*. For example, a simple method of canonicalization computes an inferred entity's attributes as the union of the attributes of its underlying mentions. Later, we will discuss more complex methods of canonicalization in the context of user feedback (Section 3). Our model of mentions, entities and attributes is reminiscent of previously proposed Bayesian models used for ER [Steorts et al., 2016].

During ER, raw evidence (i.e., mentions) are clustered into inferred entities. After the clustering, inferred entity attributes are derived via canonicalization. User feedback is supplied with respect to these inferred entities and their attributes.

2.2 Identity Uncertainty

As new evidence and user feedback is added to the KB, ER must update the set of inferred entities. In such an update, new entities may be inferred, two inferred entities may be merged or a single inferred entity maybe be split. Additionally, such an update may modify the attributes of an inferred entity while keeping its mentions intact.

Updates made by ER to the set of inferred entities pose formidable challenges for appropriately handling user feedback. For example, consider user feedback, f, that targets an inferred entity, $\hat{e}$, existing in the KB at time t. At time $t+1$, new evidence is added to the KB, which causes $\hat{e}$ to be split into multiple inferred entities. After such a split, the target of f is unknown since $\hat{e}$ not longer exists in the KB. Because of their potential to merge, split and vanish, inferred entities are said to be *ephemeral* [Wick et al., 2013].

We advocate for allowing user feedback to participate in ER alongside standard mentions. This facilitates automated routing of user feedback to mention sets with which is it compatible and naturally resolves the issues posed by identity uncertainty.

2.3 Feedback Mentions

Precisely, each piece of user feedback is represented as a *feedback mention* (FM), f. Like standard mentions, FMs possess attributes that are used during ER. However, an FM's attributes have associated weights that affect their compatibility with standard mentions, inferred entities and other FMs during ER. An FM may contain attributes with *negative* weight, corresponding to *incompatibility* with other mentions and entities possessing those attributes (with positive weights). The set of attributes and corresponding weights are known as an *attribute map*, $f.m : A \to \mathbb{R}$.

A user supplied FM that contains missing or incorrect attributes of a currently inferred entity may be routed incorrectly by ER. Consider an inferred entity, $\hat{e}$, exhibiting an attribute a and a feedback mention f designed to remove a from $\mathcal{A}(\hat{e})$. Since the feedback would contain a negative weight corresponding to attribute a, by the discussion above, it would incur some degree of incompatibility with $\hat{e}$.

Therefore, FMs are composed of two attribute maps (rather than 1): the first is called the *packaging*, $f.m_{\texttt{pack}}$, and second is called the *payload*, $f.m_{\texttt{pay}}$. Intuitively, the packaging contains a set of attributes used to guide the placement of the feedback among the KB mentions and entities during ER, and the payload contains a second set of attributes, perhaps incompatible with target inferred entity, that are used in subsequent ER decisions

(after initial placement of the feedback) and also during canonicalization. Payload attributes can be used to add missing and remove incorrect inferred entity attributes.

3. Automatic Integration of Feedback Mentions

In real-world settings, new data and feedback are created daily and must be integrated into the KB. Therefore, we develop an online approach to ER and feedback integration. In the online setting, data points (i.e., mentions and FMs) arrive and are added to the KB one at a time, and may result in mergers, deletions and modifications to the set of inferred entities. Like many instances of previous work in ER, we choose to model inferred entities hierarchically [Culotta et al., 2007, Singh et al., 2011, Wick et al., 2012b, Levin et al., 2012, Liu et al., 2014, Zhang et al., 2018]. Hierarchical modeling allows for the usage of a learned entity-level *linkage function* during inference, i.e, a function $g : \mathcal{M}^2 \times \mathcal{M}^2 \to \mathbb{R}$, which scores the compatibility of two sets of mentions (rather than pairs of mentions). Learned linkage functions have been shown to improve entity resolution systems by helping to identify set-level inconsistencies and similarities with respect to attributes like: gender, animacy and number (singular/plural) [Raghunathan et al., 2010, Durrett et al., 2013, Clark and Manning, 2016]. Additionally, hierarchical models promote efficiency in inference and facilitate the representation of uncertainty by encoding multiple partitions of the underlying mentions simultaneously [Zhang et al., 1996, Fichtenberger et al., 2013, Kobren et al., 2017].

3.1 A Hierarchical Model for Mentions and User Feedback

We model the set of inferred entities as a binary tree $\mathcal{T}$. Each leaf, $l \in \mathcal{T}$, stores a unique mention or FM, $l.x = x_i$. Each internal node, $v \in \mathcal{T}$ represents that set of mentions stored at its descendant leaves, $\texttt{lvs}(v)$. The compatibility of any two nodes in the tree can be scored via the linkage function, g. Each node, v, stores its *linkage score*, $v.\sigma$, where the linkage score of a node is computed by evaluating g on its two children, $\texttt{ch}(v)$. The linkage score of each leaf is positive infinity.

The tree, $\mathcal{T}$, encodes multiple *tree-consistent partitions* of the mentions. Each such partition corresponds to a set of inferred entities, $\hat{\mathcal{E}}$. At test time, $\hat{\mathcal{E}}$ is extracted from $\mathcal{T}$ using a threshold, τ. To do so, first, construct the set of nodes in $\mathcal{T}$ whose linkage scores are greater than the threshold,

$$S = \{v \in \mathcal{T} : v.\sigma > \tau\}.$$

Construct the set of inferred entities by finding the leaves all nodes in S whose ancestors do not appear in S,

$$\hat{\mathcal{E}} = \{\texttt{lvs}(v) : v \in S \wedge a \notin S,\ \forall a \in \texttt{ancs}(v)\},$$

where $\texttt{ancs}(\cdot)$ returns the ancestors of its argument. The threshold τ is tuned at training time on a development set (Section 4).

3.1.1 Node Representation

Each node $v \in \mathcal{T}$ stores an *attribute map*, $a.m : A \to \mathbb{R}$, that maps a set of attributes to their corresponding *weights*. For a leaf, l, that stores a standard mention, $l.m$ maps each

attribute in $l.x$ to 1. For a leaf, l', that contains a FM, $l'.m$ is set to be the attribute map stored in that FM's packaging.

The attribute map of an internal node v, is computed recursively and has two stages. In the first stage, construct:

$$v.m[a] = \sum_{c \in \texttt{ch}(v)} c.m[a],$$

where $\texttt{ch}(\cdot)$ returns the children of its arguments. In words, the weight of an attribute in $v.m$ is the sum of that attribute's weight in v's childrens' maps. In the second stage, if either child is a leaf storing a FM that contains an non-empty payload, the attribute map at that payload is added to $v.m$ in a similar fashion. Since g scores the compatibility of two nodes based on their attribute maps, when a leaf containing a FM with non-empty payload is scored against other nodes in $\mathcal{T}$, its payload is not considered. However, after being merged into the tree, the attributes in its payload are added to its parent's attribute map.

The process of constructing the attribute map at an internal node is called *canonicalization*. Since the attributes stored in packagings and payloads may have *negative* weights, FMs can be used to effectively *remove* attributes from their ancestors in $\mathcal{T}$ by reducing their weights to 0. Furthermore, an FM with a sufficiently large negative weight associated with some attribute a can increase *incompatibility* among its ancestors and other nodes in $\mathcal{T}$ that posses positive weights for the attribute a. While we employ a relatively straightforward procedure for canonicalization (aside from the special handling of packaging and payload), canonicalization may be arbitrarily complex.

3.2 The Online Grafting Algorithm

At a high level, the goal of inference is to construct a tree, $\mathcal{T}$, in which similar sets of mentions are near to one another in terms of distance in the tree. Internal nodes at the root of compatible sets of mentions will have large linkage scores and are likely to contain mentions that are clustered together in the ground-truth (assuming that g is a good model of compatibility for sets of mentions and user feedback). Trees that contain many internal nodes with large linkage scores are thus likely to contain a tree-consistent partition that corresponds well with the set of ground-truth entities. In designing an online tree construction algorithm, we consider two primary uses of user feedback: splitting and merging inferred entities. With this in mind, our algorithm includes global rearrangement step that allows for a single piece of user feedback to cause a merger between two previously disparate subtrees.

Our proposed online ER algorithm, the online grafting (OG) algorithm, is comprised of two recursive subprocedures: swap (`swap_l`) and `graft`, which promote local and global optimality, respectively. When a new data point, x, arrives, it is added as a sibling of its nearest neighbor leaf, v. Note that adding x as a sibling of v makes v's previous sibling, a, into the aunt of both x and v. Next, we invoke the `swap_l` subroutine, recursively. During a `swap_l`, consider x, v and a and check whether

$$g(v, x) < g(v, a),$$

Algorithm 1 $\texttt{Insert}(x, \mathcal{T}, g)$

$v =$ nearestNeighbor$(x, g, \mathcal{T})$
split_down(v, x)
$a =$ aunt(v)
while swap_l$(\mathcal{T}, g, x, v, a)$ & $v.\sigma > \tau$) **do**
  $v =$ sibling(x)
  $a =$ aunt(v)
graft$(\mathcal{T}, g,$ par$(x))$

Algorithm 2 $\texttt{swap_l}(\mathcal{T}, g, x, v, a)$

if $g(v, x) < g(v, a)$ **then**
  Swap x and a in $\mathcal{T}$
  return true
else
  return false

Algorithm 3 $\texttt{graft}(\mathcal{T}, g, p)$

while $p.\sigma > \tau$ **do**
  $v' = \text{argmax}_{\ell \in \texttt{lvs}(\mathcal{T}) \setminus \texttt{lvs}(p)}\, g(p, \ell)$
  if $g(p, v') > \max\{g(p, \texttt{sib}(p)), g(v', \texttt{sib}(v')), \tau\}$
  then
    Move v' to be the sibling of p
    p = par(p)
  else
    break

Figure 1: Online Grafting Algorithm

i.e., whether the v and it's previous sibling, a, are more compatible than v and x, according to g. If so, swap x and a. Repeat this procedure until x's sibling is more compatible with x than it is to its aunt, or until x's sibling is the root of an inferred entity. See Algorithms 1 and 2 for pseudocode.

After swapping terminates, the `graft` subroutine is invoked recursively from the parent of x, $\texttt{par}(x) = p$. A `graft` invoked from a node whose linkage score is below the threshold terminates immediately and no further grafts are attempted. If $p.\sigma > \tau$, search the leaves of $\mathcal{T}$ for, v', the most compatible leaf with p that is not a descendant of p. Test whether

$$g(p, v') > \max\{g(p, \texttt{sib}(p)), g(v', \texttt{sib}(v')), \tau\}$$

i.e., p and v' are more compatible with each other than to their respective siblings, and their linkage score is higher than the threshold τ. If the test succeeds, make v' the sibling of p, re-invoke the `graft` subroutine from $\texttt{par}(p)$. If the test fails, consider 3 cases:

1. if $g(p, v') \leq \tau$ then re-invoke the `graft` subroutine from $\texttt{par}(p)$;
2. if $g(p, \texttt{sib}(p)) > g(p, v')$ then repeat the test between $\texttt{par}(p)$ and v';
3. if $g(v', \texttt{sib}(v')) > g(p, v')$, then repeat the test between p and $\texttt{par}(v')$.

Intuitively, the `graft` subroutine recursively attempts mergers between ancestors of x and nodes compatible with those ancestors in $\mathcal{T}$. Notice that a merger between two nodes in $\mathcal{T}$ can only occur if: 1) both nodes are more compatible with each other than with their siblings and 2) their resultant parent has a linkage score higher than the threshold, i.e., the two nodes belong to the same inferred entity. The `graft` subroutine promotes global optimality and helps to make ER more robust to data point arrive order. See Algorithm 3 for pseudocode.

4. Training

Recall that the linkage function, g, takes two nodes, $v, v' \in \mathcal{T}$ and returns their compatibility. Define the precision of a node pair (v, v') to be

$$\texttt{pre}(v, v') = \frac{|\{(l, l') : l \in \texttt{lvs}(v), l' \in \texttt{lvs}(v'), e^\star(l) = e^\star(l')\}|}{|\texttt{lvs}(v)| \times |\texttt{lvs}(v')|}.$$

In words, the precision of a pair is the fraction of leaf-pairs, where one leaf is a descendent of v and the other a descendent of v', where both leaves belong to the same ground-truth entity. Note that FMs have no ground-truth label and are not included in this calculation. We train g to regress to the precision of its input node pair. Our training procedure is inspired by work in entity and event coreference that trains a linkage function to regress to precision of mergers in the context of bottom up agglomerative clustering [Lee et al., 2012].

We select nodes for training in the context of running the OG algorithm. When a new mention, x, arrives, we generate training examples between it and all other leaves (mimicking the nearest neighbor search). Next, we generate training examples between x and ancestors of its nearest neighbor (according to model score) resembling the `swap_l` subroutine. Finally, we insert x into the tree and generate training examples between the parent of x (from which the `graft` subroutine is initiated) and all nodes that can be viably grafted. The same procedure is repeated for each incoming mention.

The linkage function, g, also needs to be trained to appropriately handle user feedback. After a batch of N training mentions have been added to a tree, we generate up to N pieces of user feedback (generated via the Detailed scheme, discussed in Section 5). The generated feedback is either *positive* and intended to encourage a graft, or *negative* and intended to encourage a split of some inferred entity. The training feedback are also inserted into the tree, one at a time, using the OG algorithm. For each piece of feedback, we generate a training example between the feedback and its intended sibling in the tree (designated at feedback generation time), and set the precision of the example to be 1.0. Next, a merge between the feedback and its intended sibling is hallucinated and a training example is generated between the resulting parent node and the feedback's *target*. For positive feedback, the target is the root of a subtree that belongs near the feedback's sibling in the tree, and for negative feedback, the target is a node with which the feedback and its sibling are incompatible. These two types of examples help to train the model to use positive feedback to encourage grafting and negative feedback to encourage splitting.

We tune the threshold, τ, on a development set. In particular, at regular intervals throughout training, a hierarchical clustering of a set of development mentions is constructed using the OG algorithm and the current linkage function model parameters. After the clustering is built, a search is performed for the value of τ that, when used to select a tree consistent partition of the hierarchy (Section 3.1), yields the partition with the highest F-score. During each such evaluation, the model parameters and optimal value of τ are stored. At test time, the parameters and threshold that resulted in the best partition of the hierarchy (measured by F-score) are used.

5. Experiments

We perform experiments with various feedback creation and integration strategies under identity uncertainty in *author disambiguation*, a particular instantiation of entity resolution in which the ground-truth entities are real-world authors. We use the Rexa author disambiguation dataset, which includes 8 author *canopies*; each canopy contains ambiguous mentions of authors with the same first initial and last name [Culotta et al., 2007]. The mentions are derived from publications and contain: coauthors, titles, publishing venue and year of publication. The goal is to partition the mentions by real-world author.

5.1 Setup

We evaluate our proposed user feedback approach in an online ER setting. Our experimental setup is composed of two phases. In the first phase, the set of mentions arrive in a streaming fashion and are incrementally added to the hierarchical clustering, $\mathcal{T}$, using the OG algorithm (Section 3). The second phase proceeds in rounds. At the start of round t, a set of inferred entities, $\hat{\mathcal{E}}_t$, is constructed using a threshold, τ, tuned at training time on a development set (Section 4). If $\hat{\mathcal{E}}_t = \mathcal{E}^\star$, then the episode terminates. Otherwise, we simulate user interaction by generating feedback, f_t, made with respect to a randomly selected inferred entity, $\hat{e} \in \hat{\mathcal{E}}_t$. Then, f_t is added to $\mathcal{T}$ using the OG algorithm, potentially triggering a repartitioning of the mentions. No more than 400 rounds are permitted. Although rare, if after 400 rounds, the ground-truth entities have not been discovered, the method is recorded as having taken $400 + d$ rounds, where d is the number of mentions that would need to be swapped to discover $\mathcal{E}^\star$. We measure the mean number of rounds required to discover $\mathcal{E}^\star$ for each method, repeated over 25 trials, and report a paired-t statistic (and corresponding significance level) with respect to our proposed packaging-and-payload feedback mentions and each competing baseline.

5.2 Simulating Feedback

We simulate positive and negative feedback using node *purity* and *completeness*. A node $v \in \mathcal{T}$ is *pure* if:

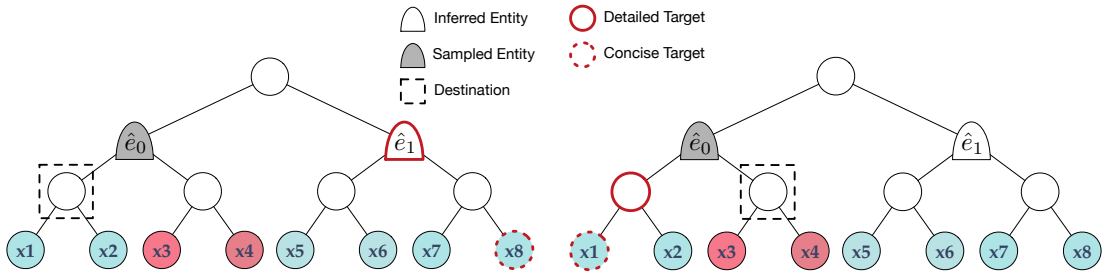
$$\exists i \, \forall l \in \texttt{lvs}(v) e^\star(l) = e_i^\star,$$

i.e., all of mentions stored at the leaves of v correspond to the same ground-truth entity, $e_i^\star$. A node $v \in \mathcal{T}$ is *complete* if:

$$\exists i \, \{l \in \texttt{lvs}(\mathcal{T}) : e^\star(l)\} \cap \{l' \in \texttt{lvs}(n) : e^\star(l')\} = e_i^\star,$$

i.e., that v's leaves contain *all* mentions of some ground-truth entity $e_i^\star$.

To generate both positive and negative feedback, we sample an intended *destination* and an intended *target*. The destination is a particular node in the tree to which the feedback is intended to be similar. The target of the feedback is a different node that the feedback is intended to be merged with or separated from the feedback's inferred entity upon insertion. Note that even with full knowledge of the destination and the target (and the ER algorithm),



(a) Positive feedback generation. (b) Negative feedback generation.

Figure 2: **Detailed and Concise feedback.** To generate either positive or negative feedback, begin by randomly sampling an inferred entity. Then, sample a *destination*–the root of a pure subtree that is also a descendant of the sampled entity. The packaging of the feedback contains the attributes at the destination. Finally, sample a *target*, which is used to construct the feedback's payload. The target is a sampled mention in the Concise setting, or the largest, pure ancestor of a sampled mention in the Detailed setting.

it is difficult to design feedback that will cause the intended tree rearrangements exactly. This is because the nodes in $\mathcal{T}$ exhibit complex relationships with each other.

5.2.1 Detailed and Concise Positive Feedback

Positive feedback is constructed with the intention of merging two subentities via a `graft`. To generate positive feedback, select the root of a pure and incomplete inferred entity, r, to be the destination of the feedback. Then, randomly select a mention, x, that is of the same ground-truth entity as the leaves of r, but is not a descendant of r. If constructing concise feedback, x is the target of the feedback; if constructing detailed feedback, traverse the ancestors of x until s, the first ancestor of x whose parent is impure. The node s becomes the target of the feedback. See Figure 2a for a visual illustration.

5.2.2 Detailed and Concise Negative Feedback

Negative feedback is constructed with the intention of splitting an inferred entity. We simulate negative feedback by randomly sampling an impure inferred entity and finding its root, r'. We construct the destination of the feedback by randomly sampling a mention $x' \in \texttt{lvs}(r')$ and finding s', the ancestor of x' closest to the root of $\mathcal{T}$ that is pure. If constructing concise feedback, sample a mention $x'' \in \texttt{lvs}(r') \setminus \texttt{lvs}(s')$ to be the target; if constructing detailed feedback, traverse the ancestors of x'' until s'', the ancestor of x'' closest to the root of $\mathcal{T}$ that is pure. In both cases, s'' becomes the target of the feedback. See Figure 2b for a visual illustration.

5.3 Baseline Methods

Integrating user feedback under identity uncertainty has not been the subject of significant study. Therefore, we propose and compare the following baseline methods:

1. **Feedback Mentions (FM)** - the approach advocated in this work. Feedback is constructed with packaging and payload attribute maps and is integrated with existing mentions via the OG algorithm.

2. **Packaging Mentions (pack)** - similar to our approach, but the feedback mentions have no payloads. All attributes that would have been included in a payload are instead added to the corresponding packaging.

3. **Hard Assignment (assign)** - generate feedback with both packaging and payload. But then, find the node $v \in \mathcal{T}$ to which it would have been made a sibling by the OG algorithm and *permanently* assign the feedback to v. If mentions are ever removed from v (e.g., by a `graft`), remove and delete all feedback assigned to v, since the target of the feedback is now unknown.

4. **Hard Mention Assignment (assign-m)** - similar to the assign approach but the feedback must be assigned to a mention in $\mathcal{T}$. Since mentions are atomic (rather than ephemeral, like inferred entities), the assigned feedback is never deleted.

5.4 User Feedback about Author Expertise

Our first experiment resembles a scenario in which users interact with a KB of scientists and provide feedback with respect to the KB's belief about a scientists expertise. The expertise of an inferred entity is represented as a bag of key phrases drawn from the titles of its underlying mentions. Users supply missing keywords and identify incorrect keywords. In this experiment, packaging contains the set of attributes at the sampled destination and the payload contains the keywords at the target (generated from mention titles).

Table 3a contains the results of the paired t-test between each baseline method and our proposed approach (FM), under DETAILED and CONCISE feedback generation schemes, with respect to the number of pieces of feedback required to discover the ground-truth partition of the mentions. Each row in the table represents a *canopy* in which the experiment is performed, and each column corresponds to a baseline method and feedback generation setting. Each cell contains the difference between the mean number of rounds required by the FM approach and a baseline approach to discover the ground-truth partition (higher is better). Positive numbers are bolded; asterisks (*) indicate statistical significance ($p < 0.05$) and two asterisks (**) indicate statistical significance ($p < 0.01$).

5.5 Authorship Feedback

Our second experiment resembles the scenario in which a user browses a KB of scientist profiles, similar to Google Scholar[1], and identifies incorrectly assigned and missing publications. Similar to the first experiment, the packaging is copied from the sampled destination. However, here, payloads contain titles stored in the sampled targets. Table 3b contains the results for this experiment of the paired t-test in the aforementioned format.

1. https://scholar.google.com/

	Detailed			Concise		
Canopy	vs. **assign**	vs. **assign-m**	vs. **pack**	vs. **assign**	vs. **assign-m**	vs. **pack**
allen_d	−2.07*	**4.03****	**3.41****	**1.88**	**2.20***	**0.75**
blum_a	−1.18	**1.59**	**1.27**	−0.85	**0.20**	**0.61**
jones_s	**1.37**	**7.74****	**6.67****	**1.11**	**0.65**	**4.04****
lee_l	−1.53	**1.02**	**4.04****	−0.22	−0.28	**0.95**
moore_a	**0.90**	**3.25****	**3.66****	−0.30	−0.71	−0.49
robinson_h	−0.99	−3.60**	−0.81	−0.76	−0.61	**0.27**
young_s	−0.71	**1.09**	**4.87****	**0.76**	**0.45**	**2.17***

(a) Experiment I: keyword feedback.

	Detailed			Concise		
Canopy	vs. **assign**	vs. **assign-m**	vs. **pack**	vs. **assign**	vs. **assign-m**	vs. **pack**
allen_d	**4.82****	**2.67***	−1.96	**5.86****	**2.01**	**1.98**
blum_a	**0.11**	−0.66	−0.98	**0.98**	**1.03**	**5.70****
jones_s	**1.49**	**1.99**	**0.85**	**1.00**	**1.02**	**3.03****
lee_l	**0.64**	**1.06**	−1.03	−1.44	**0.47**	−0.58
moore_a	**1.30**	**1.97**	−0.90	**1.94**	**1.09**	**1.05**
young_s	−1.87	**0.76**	−1.88	**0.93**	**0.91**	**0.97**

(b) Experiment II: title feedback.

Figure 3: **Paired-t statistic.** Each cell represents that difference in mean number of feedback-rounds required to discover the ground-truth entities over 25 runs between a baseline, denoted by the column heading, and our proposed approach (FM). Positive numbers indicate that FM requires *fewer* rounds of feedback than its competitor (larger numbers are better). Two asterisks (**) indicates that the statistic is significant at a 0.01 significance level; one asterisk indicates statistical significance at the 0.05 level. The `mcguire_j` canopy is excluded from Tables 3a and 3b and the `robinson_h` canopy is excluded from Table 3b since in these canopies, either: 0 or 1 edits are required to discover the ground-truth entities across baselines.

5.6 Discussion

The paired-t statistic results compares our proposed approach (FM) to the three baseline approaches. We find that FM outperforms pack in both the Detailed and Concise settings of Experiment I on all but two of the canopies. In seven of the fourteen canopies, FM outperforms pack in a statistically significant way. We hypothesize that these results indicate the importance of separating the packaging and payload attributes. We hypothesize that pack suffers from placing edits in incorrect locations in the tree due to combining the two. In Experiment II, we find similar results in the Concise setting, but find that pack requires fewer feedback rounds compared to FM in the Detailed setting (not statistically significant). We hypothesize that pack's improved performance in the Detailed setting could be due to negative title attribute feedback helping in correctly placing mentions in the tree by diverting the feedback mention away from all mentions with the negative title attributes. In comparing, FM and assign we find that our proposed approach typically per-

forms better in Experiment II while the baseline performs better in Experiment I. We note that the feedback in Experiment I is more ambiguous than Experiment II. We hypothesize that assign's performance in Experiment I is due to deleting feedback that was placed incorrectly in the tree. We find that FM frequently outperforms assign-m in both Experiment I and II. We hypothesize that assign-m frequently attaches feedback to the wrong mention. This highlights the dependence on the learned linkage function for accurately integrating feedback.

6. Related Work

Effective utilization of user feedback has been the subject of significant study in the context of KB construction. Early work, like NELL, primarily enlists humans for labeling data, which are used to train downstream models [Carlson et al., 2010]. Other work has used active learning in training relation extraction models [Angeli et al., 2014b]. Another approach employed by the DeepDive system asks humans to identify relevant features by writing feature extraction rules in support of KB construction [Ré et al., 2014, Angeli et al., 2014a].

The study of leveraging user feedback in ER has primarily focused on the solicitation of pairwise feedback. For example, given a set of mention pairs, the CrowdER system automatically prunes the set of pairs that are highly unlikely to be coreferent, and then constructs crowdsourcing HITs to collect binary labels for the remaining pairs [Wang et al., 2012]. In other similar work, human are asked to identify matching mentions across databases in data integration [Li, 2017]. Recent work studies online ER with an oracle, in which the goal is to design efficient strategies for soliciting humans for pairwise constraints among mentions [Vesdapunt et al., 2014, Mazumdar and Saha, 2016, 2017].

Recent work in author coreference also involves humans-in-the-loop [Zhang et al., 2018]. This work discusses both pairwise constraints as well as *identity constraints*. Unlike our work, their identity-level feedback is treated as a collection of pairwise constraints. As we point out, feedback that can be reduced to a set of pairwise constraints is insufficient for general KB feedback as pairwise feedback is only designed for correcting errors in ER (and not general KB errors). Similarly, many examples of user feedback are inexpressible using pairwise constraints.

The most closely related work to ours is a preliminary study of incorporating user feedback in the context of data integration [Wick et al., 2013, 2012a]. In this work, users supply pairs of mention-like records that posses either should-link or should-not-link factors, which either softly repel the pair or encourages their merger.

7. Conclusion

This work presents a framework for reasoning about user feedback for KBs under identity uncertainty. We introduce the idea of representing user feedback as feedback mentions that participate in ER alongside standard mentions. Our feedback mentions are endowed with a packaging–used to identify similar mentions during ER–and a payload–that is used to add missing attributes to inferred entities, correct mistakes and influence future ER decision. We give a hierarchical model of inferred entities and present the OG algorithm for performing online ER amongst standard and feedback mentions. In experiments, we

show that our approach often outperforms baseline approaches in terms of efficiency with respect to recovering the ground-truth partition in ER. Our work is a foundational step in addressing a significant and under-explored problem in automatic KB construction whose solution could improve the accuracy and efficacy of integrating expressive user feedback with KB content.

References

G. Angeli, S. Gupta, M. Jose, C. D Manning, C. Ré, J. Tibshirani, J. Y Wu, S. Wu, and C. Zhang. Stanfords 2014 slot filling systems. *TAC KBP*, 2014a.

G. Angeli, J. Tibshirani, J. Wu, and C. D Manning. Combining distant and partial supervision for relation extraction. *EMNLP*, 2014b.

A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

K. Clark and C. D Manning. Improving coreference resolution by learning entity-level distributed representations. In *Association for Computational Linguistics*, 2016.

A. Culotta, P. Kanani, R. Hall, M. Wick, and A. McCallum. Author disambiguation using error-driven machine learning with a ranking loss function. In *Workshop on Information Integration on the Web*, 2007.

X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. *KDD*, 2014a.

X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From data fusion to knowledge fusion. *VLDB*, 2014b.

G. Durrett, D. Hall, and D. Klein. Decentralized entity-level modeling for coreference resolution. In *Association for Computational Linguistics*, 2013.

H. Fichtenberger, M. Gillé, M. Schmidt, C. Schwiegelshohn, and C. Sohler. Bico: Birch meets coresets for k-means clustering. In *European Symposium on Algorithms*, 2013.

D. Firmani, B. Saha, and D. Srivastava. Online entity resolution using an oracle. *VLDB*, 2016.

J. Hoffart, F. M Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 2013.

A. Kobren, N. Monath, A. Krishnamurthy, and A. McCallum. A hierarchical algorithm for extreme clustering. In *Knowledge Discovery and Data Mining*, 2017.

H. Lee, M. Recasens, A. Chang, M. Surdeanu, and D. Jurafsky. Joint entity and event coreference resolution across documents. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012.

M. Levin, S. Krawczyk, S. Bethard, and D. Jurafsky. Citation-based bootstrapping for large-scale author disambiguation. *American Society for Information Science and Technology*, 2012.

Furong Li, Xin Luna Dong, Anno Langen, and Yang Li. Knowledge verification for long-tail verticals. *VLDB*, 2017.

G. Li. Human-in-the-loop data integration. *VLDB*, 2017.

W. Liu, R. Islamaj Doğan, S. Kim, D. C. Comeau, W. Kim, L. Yeganova, Z. Lu, and W. J. Wilbur. Author name disambiguation for pub med. *Information Science and Technology*, 2014.

J. Mayfield, D. Alexander, B. J Dorr, J Eisner, T. Elsayed, T. Finin, C. Fink, M. Freedman, N. Garera, P. McNamee, et al. Cross-document coreference resolution: A key technology for learning by reading. 2009.

A. Mazumdar and B. Saha. Clustering via crowdsourcing. *arXiv:1604.01839*, 2016.

A. Mazumdar and B. Saha. A theoretical analysis of first heuristics of crowdsourced entity resolution. In *AAAI Conference on Artificial Intelligence*, 2017.

K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning. A multi-pass sieve for coreference resolution. In *Empirical Methods in Natural Language Processing*, 2010.

C. Ré, A. A. Sadeghian, Z. Shan, J. Shin, F. Wang, S. Wu, and C. Zhang. Feature engineering for knowledge base construction. *arXiv:1407.6439*, 2014.

S. Singh, A. Subramanya, F. Pereira, and A. McCallum. Large-scale cross-document coreference using distributed inference and hierarchical models. In *Association for Computational Linguistics: Human Language Technologies*, 2011.

R. C. Steorts, R. Hall, and S. E. Fienberg. A bayesian approach to graphical record linkage and deduplication. *Journal of the American Statistical Association*, 2016.

N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. *VLDB*, 2014.

J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *VLDB*, 2012.

J. Wang, G. Li, T. Kraska, M. J Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. *ICDM*, 2013.

X. Wang, X. L. Dong, and A. Meliou. Data x-ray: A diagnostic tool for data errors. In *ICDM*, 2015.

M. Wick, K. Schultz, and A. McCallum. Human-machine cooperation with epistemological dbs: supporting user corrections to knowledge bases. In *Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*. ACL, 2012a.

M. Wick, S. Singh, and A. McCallum. A discriminative hierarchical model for fast coreference at large scale. In *ACL*, 2012b.

M. Wick, A. Kobren, and A. McCallum. Probabilistic reasoning about human edits in information integration. In *ICML Workshop on Machine Learning Meets Crowdsourcing*, 2013.

T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, 1996.

Y. Zhang, F. Zhang, P. Yao, and J. Tang. Name disambiguation in aminer: Clustering, maintenance, and human in the loop. In *Knowledge Discovery & Data Mining*, 2018.