Active Learning Based Entity Resolution Using Markov Logic

Jeffrey Fisher(✉), Peter Christen, and Qing Wang

Research School of Computer Science, Australian National University, Canberra, ACT 0200, Australia
{jeffrey.fisher,peter.christen,qing.wang}@anu.edu.au

Abstract. Entity resolution is a common data cleaning and data integration problem that involves determining which records in one or more data sets refer to the same real-world entities. It has numerous applications for commercial, academic and government organisations. For most practical entity resolution applications, training data does not exist which limits the type of classification models that can be applied. This also prevents complex techniques such as Markov logic networks from being used on real-world problems. In this paper we apply an active learning based technique to generate training data for a Markov logic network based entity resolution model and learn the weights for the formulae in a Markov logic network. We evaluate our technique on real-world data sets and show that we can generate balanced training data and learn and also learn approximate weights for the formulae in the Markov logic network.

1 Introduction

Entity resolution (ER) is a common data cleaning and data integration task that involves determining which records in one or more data sets refer to the same real-world entities. It has numerous applications for commercial, academic and government organisations including matching customer databases following a corporate merger, combining different data sets for research purposes, and detecting persons of interest for national security [4].

In many applications, a domain expert can perform the ER task manually, albeit in a very time-consuming fashion [4]. If a domain expert is presented with two records and their context, they can usually determine whether or not the two records refer to the same real-world entity. As a result, one approach that has been used successfully for ER is active learning [1,2,23,29], where an active learning algorithm selects pairs of records to present to an expert who then manually classifies them as either matches or non-matches. The results are then used to build an automated classification model.

However, in domains such as group linkage [11,20] or population reconstruction [6], a simple pair-wise classification model may be insufficient to accurately perform ER and the classification model may need to be more complex to capture the characteristics of the data sets involved. Collective classification techniques such as

J. Bailey et al. (Eds.): PAKDD 2016, Part II, LNAI 9652, pp. 338–349, 2016.
DOI: 10.1007/978-3-319-31750-2_27

those proposed by Bhattacharya and Getoor [3], Kalashnikov and Mehrotra [15], and Markov logic networks (MLNs) [22,27], are all effective at capturing the intricacies of complex ER problems such as matching bibliographic data, group linkage and population reconstruction. In this paper we demonstrate how we can use active learning to incorporate the domain knowledge of experts into an active learning based framework to generate and refine the rules for an MLN based ER model. While the expert still classifies pairs of records as either matches or non-matches, the rule-based approach of MLNs allows a much more sophisticated ER model to be developed.

The rest of this paper is structured as follows: In Sect. 2 we describe recent literature relating to active learning, MLNs and ER, and provide a short background on MLNs. In Sect. 3 we describe our notation and formally define the problem we are attempting to solve. In Sect. 4 we then present our approach for using active learning to create the rules and weights for an MLN based ER model. In Sect. 5 we evaluate our approach and in Sect. 6 we present our conclusions and some directions for future work.

2 Related Work and Background

Entity resolution has been the subject of a large amount of literature, and several recent surveys have been conducted [9,17]. In this section we briefly outline recent techniques relating to MLN based ER and active learning based ER.

Markov logic networks have attracted considerable research and been applied to a range of problems. Proposed by Richardson and Domingos [22] they combine first-order logic and probabilistic graphical models into a single approach. First-order logic allows a compact representation and expression of knowledge, and probabilistic graphical models allow treatment of uncertainty. Combining these two techniques allows MLNs to perform inference on a wide variety of problems and to handle contradictory knowledge and uncertainty.

A number of techniques have been proposed to determine the parameters of an MLN using a variety of different methods [13,16,26]. However, these techniques all rely on training data being available which is often not the case for ER [4]. Moreover, due to the fact that the classes of matches and non-matches are typically unbalanced, it can become very expensive to generate training data [29]. For example, a random sampling might require reviewing hundreds or thousands of non-matches for every match found.

Markov logic networks have successfully been used to perform ER [27]. The scalability of MLNs is a problem which has been partially overcome by Rastogi et al. [21]. Their experiments showed that MLN based techniques can achieve very high quality ER in situations where training data exists and the scalability problems can be overcome. As a result, an approach that uses active learning to generate training data can be used to learn the rules and weights for an MLN. In doing so it would allow the full power of MLNs to be applied to many more entity resolution applications where training data does not exist.

Active learning has been used in many fields as a way to generate training data so that supervised classification techniques can be applied. It has been

extensively studied and a survey of relevant literature has been conducted by Settles [24]. In its traditional form, it involves presenting examples to an expert for manual classification and the features of these examples are used to build a classification model.

Active learning has been successfully applied to the ER problem [1,2,23,29]. As described above, one particular problem when using active learning for entity resolution is that the classes of matches and non-matches are typically very unbalanced. As a result, a challenge with each of these techniques is to select representative examples of each class to present to the expert for manual classification. This has been addressed in several ways including exploiting cluster structure in the data [29] and using a multi-stage algorithm that prunes redundant pairs [7]. However, active learning has not been used to generate a training set for use in creating an MLN based ER model.

Active learning techniques vary based on the way examples are selected for manual classification. Of particular relevance to this paper are techniques dealing with variance reduction and density weighted methods. Variance reduction based techniques aim to reduce generalisation errors by minimising output variance and have been applied to classification models such as neural networks [18] and conditional random fields [25]. Density based techniques such as that proposed by Settles and Craven [25] aim to ensure that the examples chosen for manual classification are not only uncertain, but also somehow representative of the underlying distribution. This means that the expert does not waste time classifying outliers, which may be uncertain, but have minimal impact on the overall classification quality.

Finally, while traditional active learning techniques make use of an all-knowing domain expert to perform the manual classification, from a practical perspective this may not always be possible. Variations have been proposed which deal with noisy oracles, i.e. experts who return the wrong classification result [8]. Instead of a domain expert, other techniques make use of crowdsourcing to perform the labelling [28]. While our technique does not explicitly deal with these variations, there is no reason why different forms of oracles could not be used instead of the traditional domain expert.

2.1 Markov Logic Networks

An MLN consists of a set of weighted first-order logical formulae (rules). In a traditional first-order knowledge base, a single violation of a formula invalidates the entire knowledge base. However, for an MLN, the weight of a formula indicates the relative strength of the formula, i.e. its likelihood of being true. As a result, an MLN can handle noisy or inconsistent data without requiring a huge number of very specific formulae. Formally, given a set of first-order formulae and their weights, $\{(u_i, w_i)\}$, and a finite set of constants, we can instantiate an MLN as a Markov random field where each node is a grounding of a predicate and each feature is a grounding of one of the formulae. The joint probability distribution is given by:

$$P(\mathbf{X} = x) = \frac{1}{\mathbf{Z}} exp\left(\sum_i w_i n_i(x)\right) \quad (1)$$

where n_i is the number of times the i^{th} formula is satisified in world x, and $\mathbf{Z}$ is a normalisation constant [22].

The formulae in an MLN can be defined by an expert or determined using inductive logic programming [16], and a number of techniques have been developed to determine their weights using training data, both in batch mode [13,26] and online [14,19].

3 Notation

We briefly describe the notation we will use in this paper. We begin with a data set $\mathbf{R}$. We assume a finite list of *rules* $\mathbf{U} = \langle u_1, u_2, \ldots u_k \rangle$, where each rule $u \in \mathbf{U}$ is a function that takes as input two records $r_i, r_j \in \mathbf{R}$ and returns either $True$ or $False$. We denote the number of rules in $\mathbf{U}$ as $|\mathbf{U}|$. Each rule $u \in \mathbf{U}$ has a real valued weight denoted $w(u)$. An example of a rule is:

$$u(r_1, r_2) = HasName(r_1, n_1) \wedge HasName(r_2, n_2) \wedge SameName(n_1, n_2)$$

which returns $True$ if records r_1 and r_2 have values n_1 and n_2 respectively in the *Name* attribute, and n_1 and n_2 are the same, and $False$ otherwise. Rules with positive weights indicate positive evidence that records r_i and r_j refer to the same entity (match), while rules with negative weights indicate negative evidence that r_i and r_j refer to the same entity (non-match). By creating rules in this format and relating them to whether the records are a match or non-match, these rules can later be easily converted to formulae for the MLN model [27]. We assume that the initial list of rules is provided by a domain expert who will be performing the manual classification in the active learning process.

We define a *rule vector* as a binary vector v, and the set of such rule vectors as $\mathbf{V}$. We generate a rule vector by applying each $u \in \mathbf{U}$ to a pair of records $r_i, r_j \in \mathbf{R}$ using a function $\Psi(\mathbf{U}, r_i, r_j)$ which returns a rule vector v of length $|\mathbf{U}|$ where $v[k] = 1$ if $u_k(r_i, r_j) = True$ and $v[k] = 0$ if $u_k(r_i, r_j) = False$ for $1 \leq k \leq |\mathbf{U}|$. Since different pairs of records may produce the same rule vector, for each rule vector v_x we define a set of record pairs $P(v_x) = \{\langle r_y, r_z \rangle : r_y, r_z \in \mathbf{R}, \Psi(\mathbf{U}, r_y, r_z) = v_x\}$. The cardinality of $P(v_x)$ is denoted $|P(v_x)|$. Since $\mathbf{V}$ only includes rule vectors that are produced by a pair of records in $\mathbf{R}$, its size is limited to the smaller of $O(|\mathbf{R}|^2)$ and $O(2^{|\mathbf{U}|})$ and in practice it is significantly smaller than both these limits. The weight of each v is defined as $w(v) = (\sum w(u_k) : 1 \leq k \leq |\mathbf{U}|$ and $v[k] = 1)$. In essence, $w(v)$ is the sum of the weights of those rules that are set to $True$ in v.

Our technique uses a *domain expert* $\mathbf{E}$ (or an oracle, crowd sourcing, etc. as per other active learning techniques [24]) to perform manual classification as part of our approach. However, typically the expert can only classify a certain number of record pairs so we assume a total *budget* of manual classifications b with $b \geq 1$. The budget is divided into q manual classifications per round,

over n rounds, where $q * n = b$. In future work we intend to investigate ways to adaptively distribute the budget [29]. We denote the output of the manual classification of records r_j and r_k as o_{jk} where $o_{jk} = True$ if r_j and r_k are classified as a match by the expert and $o_{jk} = False$ if they are classified as a non-match.

During the active learning process we need a way of determining which record pairs $\langle r_i, r_j \rangle$ to present to the expert for manual classification. We define a *strategy* S as an ordering of the rule vectors in $\mathbf{V}$. After ordering $\mathbf{V}$ by S, we select one candidate pair from each of $P(v_1) \ldots P(v_q)$ to be presented to the expert, where q is the number of manual classifications each round. Some possible strategies for S include selecting: (1) the most definite positive examples ($S^+ :=$ descending sort of $w(v)$), (2) the most definite negative examples ($S^- :=$ ascending sort of $w(v)$), (3) the most definite examples of either type ($S^d :=$ descending sort of $|w(v)|$), (4) the most ambiguous cases ($S^a :=$ ascending sort of $|w(v)|$, since the most ambiguous cases have a total weight close to 0), (5) the most commonly occuring vectors ($S^c :=$ descending sort of $|P(v)|$) and (6) $S^r :=$ random ordering.

Problem Statement: Given a data set $\mathbf{R}$, an initial set of rules $\mathbf{U}_s$, a budget b, a strategy S, and an expert $\mathbf{E}$ who can make manual classifications, the problem is to generate a balanced set of training examples for an MLN based ER model, a final set of rules $\mathbf{U}_n$, and values of $w(u_i)$ for each $u_i \in \mathbf{U}_n$.

4 Methodology

The basic idea of our approach is that we apply traditional active learning techniques [23] by presenting pairs of records to an expert for manual classification. However, in addition to classifying each pair as either a match or a non-match we ask the expert to specify the reason for their decisions - i.e. the rule(s) that most influenced their decisions. In the case where the expert's classification of a pair contradicts the weight of the corresponding rule vector, i.e. a record pair which produces a rule vector with $w(v) > 0$ but the expert classifies the pair as a non-match, or $w(v) < 0$ but the expert classifies the pair as a match, we allow the expert to specify a new rule that would cover the particular pair. We could also use inductive logic programming based techniques [16] to specify the new rule instead. This new rule is then added to the current list of rules and is used to evaluate record pairs in the next iteration of the active learning process.

Our approach is described in Algorithm 1. We begin with an initial list of rules $\mathbf{U}_s$ created by the domain expert, along with a selected strategy S. Throughout the process, we refine the list of rules along with the weights in order to end up with a final list of rules that will be converted to formulae in an MLN based ER model. Our algorithm assumes the budget is split into n rounds, with q manual classifications per round, such that $q * n = b$. Each iteration of the main loop (lines 2 – 22) is one round of the algorithm.

Within each round, we start by creating an empty set of rule vectors (line 3). Because calculating a rule vector for each pair $r_i, r_j \in \mathbf{R}$ has time complexity

Algorithm 1. Active Weight Learning

Input:
- Set of records: $\mathbf{R}$
- Initial list of rules: $\mathbf{U}_s$
- Strategy: S
- Budget: b split into q, n such that $q * n = b$ // Questions per round, number of rounds
- Expert: $\mathbf{E}$

Output:
- Final list of rules: $\mathbf{U}_n$

1: $i := 1$, $\mathbf{U}_0 := \langle\ \rangle$ // Set $\mathbf{U}_0$ to an empty list
2: while $i \leq n$ and $\mathbf{U}_i \neq \mathbf{U}_{i-1}$ do:
3: $\mathbf{V} := \emptyset$, $\mathbf{U}_i := \mathbf{U}_{i-1}$ // Create $\mathbf{U}_i$, for $i = 1$, $\mathbf{U}_1 := \mathbf{U}_s$
4: $\mathbf{B} := GenerateBlocks(\mathbf{R})$ // Create a set of blocks
5: foreach $b_a \in \mathbf{B}$ do: // In practice we only do this for a proportion of blocks in $\mathbf{B}$
6: foreach $\langle r_j, r_k \rangle \in b_a$ do:
7: $v_{jk} := \Psi(\mathbf{U}_i, r_j, r_k)$ // v_{jk} is generated by applying the rules in $\mathbf{U}_i$ to r_j, r_k
8: if $v_{jk} \in \mathbf{V}$ do:
9: $UpdateRuleVector(\mathbf{V}, v_{jk})$ // Update $P(v_{jk})$ if we have seen v_{jk} before
10: else do:
11: $\mathbf{V}.add(v_{jk})$ // Otherwise add v_{jk} to $\mathbf{V}$ with $P(v_{jk}) = \{\langle r_j, r_k \rangle\}$
12: $\mathbf{V}^* := Order(\mathbf{V}, S)$ // Order $\mathbf{V}$ based on strategy S
13: for $v_j \in \mathbf{V}^*$ and $1 \leq j \leq q$ do: // For the first q rule vectors in $\mathbf{V}^*$
14: $\langle r_a, r_b \rangle := P(v_j).random$ // Randomly select a candidate pair from $P(v_j)$
15: $o_{ab} := ManualClassify(r_a, r_b, \mathbf{E})$ // Get the true match status from expert
16: $UpdateRuleWeights(\mathbf{U}_i, o_{ab})$ // Update rule weights based on expert
17: if $(o_{ab} = True$ and $w(v_j) < 0)$ or $(o_{ab} = False$ and $w(v_j) > 0)$ do:
18: $u_{new} := GetNewRule$ // Get a new rule to cover the pair
19: $\mathbf{U}_i.append(u_{new})$
20: while $|\mathbf{U}_i| > |\mathbf{U}_s|$ do: // Check if we added some rules
21: $RemoveLowestPredictiveRule(\mathbf{U}_i)$ // And if we did, remove the least useful rules
22: $i++$
23: return $\mathbf{U}_n$

$O(|\mathbf{R}|^2)$, we make use of blocking to reduce the number of rule vectors calculated. Blocking is the process whereby a data set is split into subsets called blocks and only records within the same block are compared [5]. We assume the blocking process is a black box where we pass a data set and get back a set of blocks (line 4). Within each block, we evaluate the rules on each pair of records to generate the rule vectors (lines 5–11).

We apply the strategy S to the rule vectors in $\mathbf{V}$ to produce an ordering $\mathbf{V}^*$ (line 12). Then, we select a candidate pair from each of the first q rule vectors in $\mathbf{V}^*$ to be presented to the expert $\mathbf{E}$ for manual classification (lines 14–15). Based on the output of this manual classification we update the weights of the rules in $\mathbf{U}_i$, and if the manual classification result is contrary to what the weights of the rule vectors indicate, i.e. $w(v_{ij}) > 0$ but $o_{ij} = False$ or $w(v_{ij}) < 0$ but $o_{ij} = True$, we ask the expert to create a new rule that would cover the incorrectly classified pair (line 18). Alternatively, techniques such as inductive logic programming based learning [16] could also be used to refine the rules in $\mathbf{U}_i$ so that the incorrectly classified pair is fixed.

Finally, if our number of rules has increased in the current round (line 20), we look at removing the least informative rules from $\mathbf{U}_i$ based on a combination of the absolute values of their weights and their coverage. For each rule $u_j \in \mathbf{U}_i$, we calculate a score $s = |w(u_j)| * log(|T(u_j)|)$ where $T(u_j) = \{r_a, r_b : r_a, r_b \in \mathbf{R}, u_j(r_a, r_b) = True\}$. We then remove the rules with the lowest score (line 21) which allows us to avoid overfitting and keeps rules with a balance of high predictive power (both positive and negative) and high coverage.

4.1 Building Blocks of the Algorithm

We discuss some aspects of the approach in more detail, namely blocking, the choice of strategy S, and some considerations to prevent overfitting.

Blocking. In our approach we treat the blocking step in line 4 as a black box. However, there are some blocking techniques that are more appropriate to our approach than others. One practical consideration is that ideally we will use the same blocks from our algorithm in the MLN based ER model. Since the scalability of MLNs is typically very poor, this means we need to limit the maximum block size by using sorted neighbourhood based blocking [12] or a size-constrained clustering approach similar to Fisher et al. [10].

In addition, the rules in $\mathbf{U}_i$ can be used to inform the blocking. If we sort the rules in $\mathbf{U}_i$ based on $w(u)$ and use the rules with the highest weights to generate blocks, we typically get a blocking approach that is well aligned with an MLN based ER model. For example, if having the same surname is strong evidence that the records are a match (as indicated by a large positive weight to the associated rule), then by placing records with the same surname into blocks, we know that during the ER process, the records being compared have a higher likelihood of being matches since they satisfy at least one rule with a large positive weight. After we have generated the set of blocks $\mathbf{B}$, we need to use the candidate pairs in each $b \in \mathbf{B}$ to generate our rule vectors.

Even when blocking is used, computing $\Psi(\mathbf{U}, r_i, r_j)$ for all candidate pairs within all blocks is equivalent to the complete matching step in a traditional ER approach [4]. Since it is impossible to present all the pairs to the expert for manual classification anyway, we instead sample a large number of pairs. While this does not mathematically guarantee that every rule vector is generated, in practice it means that the frequently ocurring ones are present. In addition, after the first round (i.e. once $i > 1$) if a rule vector v_k is missing but was present in an earlier round, we can apply $\mathbf{U}$ to the pairs in $P(v_k)$ from the earlier round to ensure that v_k is also represented in the current round.

Selection Strategy. In Sect. 3 we described several possible strategies for S based on $w(v)$ and $|P(v)|$. In practice we also wish to record the number of manually classified examples for each $v \in \mathbf{V}$, along with their counts of $True$ and $False$ from the manual classifications. This is because if we keep the same S for each round, it is likely the ordering of $\mathbf{V}$ does not change significantly between rounds. As a result, we end up sampling candidate pairs for the same rule vectors in each round. We can avoid this problem by incorporating previous results into the ordering. If the manual classification for a rule vector always returns the same value of $True$ or $False$ we can lower its priority in the ordering. However, if a rule vector produces a mix of $True$ and $False$ values when being manually classified, we may wish to increase its priority in the ordering. It is also possible to change the strategy S between rounds. For example, we can start by looking at the most frequently occuring rule vectors (highest values of $|P(v)|$) and once

we have a good list of rules for the frequent pairs we can change strategies to deal with the difficult cases (lowest values of $|w(v)|$). In the future we intend to investigate ways to determine the optimal choice of strategy.

Overfitting. In practice care needs to be taken to prevent overfitting. In the implementation of our approach described in Algorithm 1, we assume the starting number of rules is the desired number for the final model. However, there is no reason this needs to be the case and we can use a different mechanism to limit the number of rules such as specifying a different maximum number of rules or a minimum score s_{min}, using the scoring method described in Sect. 4.0. In practice it can be the case that the newly added rule has the lowest score and we end up removing it straight away. This essentially means that the expert **E** has ended up manually classifying an outlier or exception, so using it to inform other matching decisions will lead to overfitting.

5 Experimental Evaluation

We have evaluated our approach on two data sets. (1) Cora: This is a public bibliographic data set of scientific papers that has previously been used to evaluate ER techniques [27]. This data set contains 1,295 records and truth data is available. (2) UKCD: This data set consists of census data for the years 1851 to 1901 in 10 year intervals for the town of Rawtenstall and surrounds in the United Kingdom. It contains approximately 150,000 individual records of 32,000 households. A portion of this data (nearly 5,000 records) has been manually linked by domain experts. Fu et al. [11] have used this data set for household based group linkage where the task was to link households across time.

Instead of presenting candidate pairs to the expert **E** for manual classification, we limited our evaluation to the portions of the two data sets that truth data was available and o_{ij} was known for all pairs of $r_i, r_j \in \mathbf{R}$, i.e. the entirety of Cora and about 5,000 records in UKCD. Instead of allowing the expert to select significant rules and altering their weights appropriately we adjusted the weights automatically. For each rule vector v_{ij}, where a candidate pair $\langle r_i, r_j \rangle \in P(v_{ij})$ was selected for manual classification, we adjusted the weights for each $u_k \in \mathbf{U}$ where $u_k(r_i, r_j) = True$ as follows: if $o_{ij} = True$ and $w(u_k) > 0$ we increased $w(u_k)$ by 10 % and if $o_{ij} = True$ and $w(u_k) < 0$ we decreased $w(u_k)$ by 10 %. If $o_{ij} = False$ we performed the opposite adjustments. Essentially this meant that rules that were $True$ and correctly predicted the classification result had their weights increased while those that were $True$ but incorrectly predicted the classification result had their weights decreased.

All our experiments were performed on a Macbook Air with an Intel I5 1.3 Ghz CPU, 4 GBytes of memory and running OS X. All programs were written in Python 3. None of our experiments took longer than five minutes to run 20 rounds of 10 manual classifications each (200 manual classifications in total). Since in a practical ER application each manual classification is likely to take a

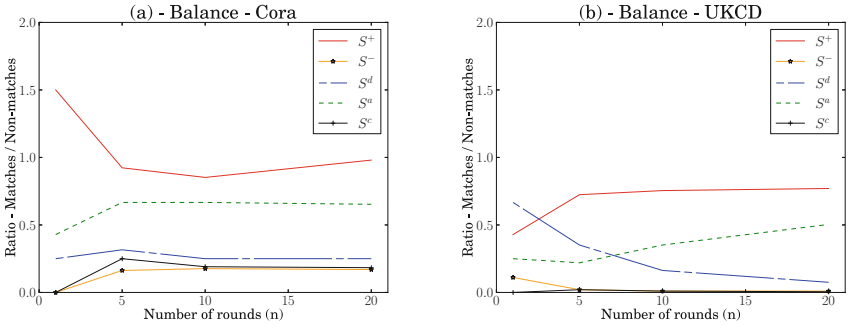

Fig. 1. The ratio of matches / non-matches for different strategies and different values of n.

few minutes, our approach runs fast enough that the slowest part of the process will be most likely be the manual classification step.

For our evaluation we consider two measures, the ratio of candidate pairs classified as matches to the candidate pairs classified as non-matches by the expert **E**, and the rule weights generated by our techniques. To generate a balanced training set the ratio between matches and non-matches should be as close to 1 as possible. We also examine the weights that are generated for the rules in $\mathbf{U}_n$ at the end of our algorithm as ideally these will be used as the weights for the corresponding formulae in the MLN. We consider both the total sum of the weights calculated as $|\sum(w(u_i) : u_i \in \mathbf{U}_n)|$ as well as the maximum value of $|w(u_i)| : u_i \in \mathbf{U}_n$.

For both data sets, we created a list of rules based on simple attribute equality and assigned weights of 1 to those rules that were positive evidence that the two records might be the same (i.e. the values in the attribute were the same) and weights of -1 to those rules that were negative evidence that the two records might be the same (the values in the attribute were different). For Cora we used a total of eight rules, four with positive weights and four with negative weights, and for UKCD we used 10 rules, five with positive weights and five with negative weights. For both data sets we set $q = 10$ and varied the value of n to simulate different budgets b.

Figure 1 shows the ratio of candidate pairs classified as matches vs non-matches in the manual classification step for the strategies described in Sect. 3. Figure 2 shows the sum of the rule weights for the rules in $\mathbf{U}_n$ and Fig. 3 shows the largest value of $|w(u_i)|$ for $u_i \in \mathbf{U}_n$. As can be seen in the results, the strategies that produce the most balanced splits of the training data between matches and non-matches are S^+ and S^a with the other strategies generally performing poorly. The strategy S^+ orders the rule vectors by descending walue of $w(v)$ which means that a candidate pair from rule vectors with the highest positive weights are always presented to the expert for manual classification in each round. Since apriori these are the rule vectors most likely to be matches, this strategy is effective at overcoming the class imbalance between matches and

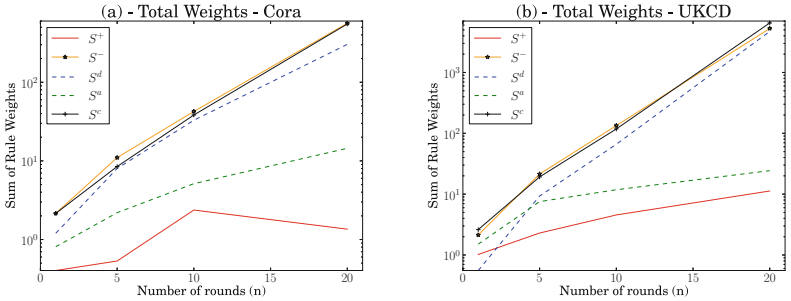

Fig. 2. The total weight of the rules in $\mathbf{U}_n$ for different strategies and different values of n.

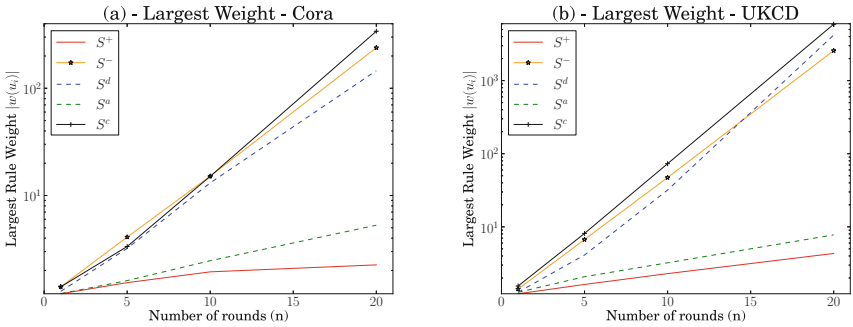

Fig. 3. The maximum value of $|w(u_i)|$ for $u_i \in \mathbf{U}_n$ for different strategies and different values of n.

non-matches. While S^a selects candidate pairs from rule vectors that are hard to classify, i.e. $w(v)$ is close to 0, it means that although it starts by selecting a higher proportion of non-matches, as the algorithm proceeds the negative rule weights get further away from 0. This means that the candidate pairs that are presented to the expert are more likely to have satisfied positive rules and thus be matches which leads to a degree of balance in the classifications. This is also true of the weights generated by these two strategies which do not get very large, either in terms of the sum of the weights or the largest absolute weight.

However, strategies S^-, S^d and S^c all produced very poor results overall. This is because after the initial rounds, these strategies mean that the expert is almost exclusively being presented with non-matches which creates a compounding feedback loop where the weights on the negative rules get more and more negative. As a result, none of these strategies produce balanced training sets and the weights they produce for the rules are not useful at all for an MLN based entity resolution model.

While the weights generated by the strategies S^+ and S^a are plausible for an MLN based ER model, they do not resolve to an equilibrium around the correct weights like we had hoped for. As a result, in the future we intend to combine

our technique with an online weight learning algorithm such as those proposed by Mihalkova and Mooney [19] or Huynh and Mooney [14]. By linking the weight learning with the generation of training data through active learning we hope to be able to generate weights for an MLN with our technique.

6 Conclusions and Future Work

In this paper we have presented an active learning technique for generating training examples for a Markov logic network based entity resolution model, as well as a technique for learning the necessary weights for the MLN formulae. We have also presented a method which allows a domain expert to add new rules to the MLN to capture pairs of records that are not being correctly classified by the existing model. We show that our technique is effective at generating balanced training sets to be used for learning an MLN based ER model, however it is currently less effective at generating appropriate weights for the MLN formulae.

In the future we intend to extend the work in several directions. We aim to investigate ways of adaptively distributing the budget of manual classifications, both in terms of the number of questions per round and the number of rounds similar to Wang et al. [29]. We also aim to test our techniques on other data sets and ER problems such as population reconstruction [6]. Finally, due to the fact that our techniques were not as successful at learning correct weights for the formulae in the MLN, we intend to investigate using our technique in conjunction with an online weight learning approach for MLNs, such as those proposed by Mihalkova and Mooney [19] or Huynh and Mooney [14].

References

1. Arasu, A., Götz, M., Kaushik, R.: On active learning of record matching packages. In: ACM SIGMOD, pp. 783–794, Indianapolis (2010)
2. Bellare, K., Iyengar, S., Parameswaran, A.G., Rastogi, V.: Active sampling for entity matching. In: ACM SIGKDD. ACM (2012)
3. Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. ACM TKDD **1**(1), 5 (2007)
4. Christen, V.: Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Data-Centric Systems and Applications. Springer, Heidelberg (2012)
5. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE TKDE **24**(9), 1537–1555 (2012)
6. Christen, P., Vatsalan, D., Fu, Z.: Advanced record linkage methods and privacy aspects for population reconstruction - a survey and case studies. In: Bloothooft, G., Christen, P., Mandemakers, K., Schraagen, M. (eds.) Population Reconstruction, pp. 87–110. Springer, Switzerland (2015)
7. Dal Bianco, G., Galante, R., Gonalves, M., Canuto, S., Heuser, C.: A practical and effective sampling selection strategy for large scale deduplication. IEEE KDE **27**(9), 2305–2319 (2015)
8. Du, J., Ling, C.: Active learning with human-like noisy oracle. In: IEEE ICDM, pp. 797–802 (2010)

9. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE TKDE **19**(1), 1–16 (2007)
10. Fisher, J., Christen, P., Wang, Q., Rahm, V.: A clustering-based framework to control block sizes for entity resolution. In: ACM SIGKDD (2015)
11. Fu, Z., Christen, P., Zhou, J.: A graph matching method for historical census household linkage. In: Tseng, V.S., Ho, T.B., Zhou, Z.-H., Chen, A.L.P., Kao, H.-Y. (eds.) PAKDD 2014, Part I. LNCS, vol. 8443, pp. 485–496. Springer, Heidelberg (2014)
12. Hernandez, M.A., Stolfo, S.J.: Real-world data is dirty: Data cleansing and the merge/purge problem. DMKD **2**(1), 9–37 (1998)
13. Huynh, T.N., Mooney, R.J.: Discriminative structure and parameter learning for Markov logic networks. In: ACM ICML (2008)
14. Huynh, T.N., Mooney, R.J.: Online max-margin weight learning for Markov logic networks. In: SDM, pp. 642–651 (2011)
15. Kalashnikov, D., Mehrotra, S.: Domain-independent data cleaning via analysis of entity-relationship graph. ACM TODS **31**(2), 716–767 (2006)
16. Kok, S., Domingos, P.: Learning the structure of Markov logic networks. In: ACM ICML (2005)
17. Köpcke, H., Rahm, E.: Frameworks for entity matching: a comparison. Data Knowl. Eng. **69**(2), 197–210 (2010)
18. MacKay, D.J.: Information-based objective functions for active data selection. Neural Comput. **4**(4), 590–604 (1992)
19. Mihalkova, L., Mooney, R.: Learning to disambiguate search queries from short sessions. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009, Part II. LNCS, vol. 5782, pp. 111–127. Springer, Heidelberg (2009)
20. On, B.W., Elmacioglu, E., Lee, D., Kang, J., Pei, J.: Improving grouped-entity resolution using quasi-cliques. In: IEEE ICDM, pp. 1008–1015 (2006)
21. Rastogi, V., Dalvi, N., Garofalakis, M.: Large-scale collective entity matching. VLDB Endowment **4**, 208–218 (2011)
22. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. **62**(1–2), 107–136 (2006)
23. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: ACM SIGKDD (2002)
24. Settles, B.: Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin, Madison (2010)
25. Settles, B., Craven, M.: An analysis of active learning strategies for sequence labeling tasks. In: ACL Empirical methods in NLP (2008)
26. Singla, P., Domingos, P.: Discriminative training of Markov logic networks. AAAI **5**, 868–873 (2005)
27. Singla, P., Domingos, P.: Entity resolution with Markov logic. In: IEEE ICDM, pp. 572–582 (2006)
28. Wang, J., Kraska, T., Franklin, M.J., Feng, J.: CrowdER: crowdsourcing entity resolution. Proc. VLDB Endow. **5**(11), 1483–1494 (2012)
29. Wang, Q., Vatsalan, D., Christen, P.: Efficient interactive training selection for large-scale entity resolution. In: Cao, T., Lim, E.-P., Zhou, Z.-H., Ho, T.-B., Cheung, D., Motoda, H. (eds.) PAKDD 2015. LNCS, vol. 9078, pp. 562–573. Springer, Heidelberg (2015)