Learning Blocking Schemes for Record Linkage*

Matthew Michelson and **Craig A. Knoblock**
University of Southern California
Information Sciences Institute,
4676 Admiralty Way
Marina del Rey, CA 90292 USA
{michelso,knoblock}@isi.edu

Abstract

Record linkage is the process of matching records across data sets that refer to the same entity. One issue within record linkage is determining which record pairs to consider, since a detailed comparison between all of the records is impractical. *Blocking* addresses this issue by generating candidate matches as a preprocessing step for record linkage. For example, in a person matching problem, blocking might return all people with the same last name as candidate matches. Two main problems in blocking are the selection of attributes for generating the candidate matches and deciding which methods to use to compare the selected attributes. These attribute and method choices constitute a *blocking scheme*. Previous approaches to record linkage address the blocking issue in a largely ad-hoc fashion. This paper presents a machine learning approach to automatically learn effective blocking schemes. We validate our approach with experiments that show our learned blocking schemes outperform the ad-hoc blocking schemes of non-experts and perform comparably to those manually built by a domain expert.

Introduction

Record linkage is the process of matching records between data sets that refer to the same entity. For example, given databases of AI researchers and Census data, record linkage finds the common people between them, as in Figure 1. Since record linkage needs to compare each record from each dataset, scalability is an issue. Consider the above case, and assume each dataset consists of just 5,000 records. This would require 25 million detailed comparisons if all records are compared. Clearly, this is impractical.

To deal with this issue, a preprocessing step compares all of the records between the data sets with fast, approximate methods to generated candidate matches. This step is called *blocking* because it partitions the full cross product of record comparisons into mutually exclusive blocks (Newcombe 1967). That is, to block on an attribute, first we sort or cluster the data sets by the attribute. Then we apply the comparison method to only a single member of a block. After blocking, the candidate matches are examined in detail to discover true matches. This paper focuses on blocking.

Census Data

First Name	Last Name	Phone	Zip
Matt	Michelson	555-5555	12345
Jane	Jones	555-1111	12345
Joe	Smith	555-0011	12345

A.I. Researchers

First Name	Last Name	Phone	Zip
Matthew	Michelson	555-5555	12345
Jim	Jones	555-1111	12345
Joe	Smeth	555-0011	12345

match match

Figure 1: Record linkage example

There are two main goals of blocking. First, the number of candidate matches generated should be small to minimize the number of detailed comparisons in the record linkage step. Second, the candidate set should not leave out any possible true matches, since only record pairs in the candidate set are examined in detail during record linkage. These blocking goals represent a trade off. On the one hand, the goal of record linkage is to find all matching records, but the process also needs to scale. This makes blocking a challenging problem.

Most blocking techniques rely on the *multi-pass approach* of (Hernández & Stolfo 1998). The general idea of the multi-pass approach is to generate candidate matches using different attributes and methods across independent runs. For example, using Figure 1, one blocking run could generate candidates by matching tokens on the *last name* attribute along with first letter of the *first name*. This returns one true match (M Michelson pair) and one false match (J Jones pair). Another run could match tokens on the *zip* attribute. Intuitively, different runs cover different true matches, so the union should cover most of the true matches.

The effectiveness of a multi-pass approach depends on which attributes are chosen and the methods used. For ex-

*This research was sponsored by the Air Force Research Laboratory, Air Force Material Command, USAF, under Contract number FA8750-05-C-0116. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL or the U.S. Government.

ample, generating candidates by matching the *zip* attribute returns all true matches, but also, unfortunately, every possible pair. Method choice is just as important. If we change our *last name* and *first name* rule to match on the first three letters of each attribute, we drop the false match (J Jones). While past research has improved the methods for comparing attributes, such as bi-gram indexing (Baxter, Christen, & Churches 2003), or improved the ways in which the data is clustered into blocks (McCallum, Nigam, & Ungar 2000), this has not addressed the more fundamental issue of blocking research: which attributes should be used and which methods should be applied to the chosen attributes. As (Winkler 2005) puts it, "most sets of blocking criteria are found by trial-and-error based on experience."

This paper addresses this fundamental question of blocking research. We present a machine learning approach to discovering which methods and which attributes generate a small number of candidate matches while covering as many true matches as possible, fulfilling both goals for blocking. We believe these goals, taken together, are what define *effective* blocking, and their optimization is the goal we pursue.

The outline of this paper is as follows. In the next section we further define the multi-pass approach and the choice of which attributes and methods to use for blocking. After that, we describe our machine learning approach to blocking. Then, we present some experiments to validate our idea. Following that we describe some related work and we finish with some conclusions and future work.

Blocking Schemes

We can view the multi-pass approach as a disjunction of conjunctions, which we call a *Blocking Scheme*. The disjunction is the union of each independent run of the multi-pass approach. The runs themselves are conjunctions, where each conjunction is an intersection between {*method,attribute*} pairs.

Using our example from the introduction, one conjunction is ({*first-letter, first name*} $\wedge$ {*token-match, last name*}) and the other is ({*token-match, zip*}), so our blocking scheme becomes:
BS = ({*first-letter, first name*} $\wedge$ {*token-match, last name*}) $\cup$ ({*token-match, zip*})

A blocking scheme should include enough conjunctions to cover as many true matches as it can. For example, the first conjunct by itself does not cover all of the true matches, so we added the second conjunction to our blocking scheme. This is the same as adding more independent runs to the multi-pass approach.

However, since a blocking scheme includes as many conjunctions as it needs, these conjunctions should limit the number of candidates they generate. As we stated previously, the *zip* attribute might return the true positives, but it also returns every record pair as a candidate. By adding more {*method, attribute*} pairs to a conjunction, we can limit the number of candidates it generates. For example, if we change ({*token-match, zip*}) to ({*token-match, zip*} $\wedge$ {*token-match, first name*}) we still cover new true matches, but we only generate one additional candidate.

Therefore effective blocking schemes should learn conjunctions that minimize the false positives, but learn enough of these conjunctions to cover as many true matches as possible. These two goals of blocking can be clearly defined by the Reduction Ratio and Pairs Completeness (Elfeky, Verykios, & Elmagarmid 2002).

The *Reduction Ratio* (RR) quantifies how well the current blocking scheme minimizes the number of candidates. Let C be the number of candidate matches and N be the size of the cross product between both data sets.

$$RR = 1 - C/N$$

It should be clear that adding more {*method,attribute*} pairs to a conjunction increases its RR, as when we changed ({*token-match, zip*}) to ({*token-match, zip*} $\wedge$ {*token-match, first name*}).

Pairs Completeness (PC) measures the coverage of true positives, i.e., how many of the true matches are in the candidate set versus those in the entire set. If S_m is the number of true matches in the candidate set, and N_m is the number of matches in the entire dataset, then:

$$PC = S_m/N_m$$

Adding more disjuncts can increase our PC. For example, we added the second conjunction to our example blocking scheme because the first did not cover all of the matches.

So, if our blocking scheme can optimize both PC and RR, then our blocking scheme can fulfill both of our defined goals for blocking: it can reduce the candidate matches for record linkage without losing matches, which would hinder the accuracy of the record linkage.

Generating Blocking Schemes

To generate our blocking schemes, we use a modified version of the Sequential Covering Algorithm (SCA) which discovers disjunctive sets of rules from labeled training data (Mitchell 1997). Intuitively, SCA learns a conjunction of attributes , called a rule, that covers some set of positive examples from the training data. Then it removes these covered positives examples, and learns another rule, repeating this step until it can no longer discover a rule with performance above a threshold. In our case, the positive examples for SCA are matches between the data sets, and the attributes SCA chooses from in the algorithm are {method,attribute} pairs. The result of running SCA in our case will be a blocking scheme.

SCA, in its classic form is shown in Table 1.

SCA maps naturally for learning blocking schemes. At each iteration of the while loop, the *LEARN-ONE-RULE* step learns a conjunction that maximizes the reduction ratio

Table 1: Classic Sequential Covering Algorithm

SEQUENTIAL-COVERING$(class, attributes, examples, threshold)$
$LearnedRules \leftarrow \{\}$
$Rule \leftarrow$ *LEARN-ONE-RULE*$(class, attributes, examples)$
$While$ (*PERFORMANCE*$(Rule) > threshold$) do
$\quad LearnedRules \leftarrow LearnedRules \cup Rule$
$\quad Examples \leftarrow Examples$ - {*Examples covered by Rule*}
$\quad Rule \leftarrow$ *LEARN-ONE-RULE*$(class, attributes, examples)$
Return LearnedRules

(RR). Although *LEARN-ONE-RULE* is described in detail later, intuitively, it adds {method, attribute} pairs to conjunctions to yield higher and higher RR values, until the RR stops improving.

So, even though a restrictive conjunction might not cover all of the true matches, as SCA adds more disjuncts, more of the true matches will be covered, increasing the blocking scheme's pairs completeness (PC). However, the classic SCA needs to be modified so that we learn as many conjunctions as are necessary to cover all of the example true matches. This helps maximize the PC for the training data. Also, since the algorithm greedily constructs and learns the conjunctions, it may learn a simpler conjunction at a later iteration, which covers the true matches of a previous conjunction. This is an opportunity to simplify the blocking scheme by replacing the more restrictive conjunction with the less restrictive one. These two issues are addressed by our slight modifications to SCA, shown in **bold** in Table 2.

Table 2: Modified Sequential Covering Algorithm

SEQUENTIAL-COVERING$(class, attributes, examples)$
$LearnedRules \leftarrow \{\}$
$Rule \leftarrow$ *LEARN-ONE-RULE*$(class, attributes, examples)$
While examples left to cover, do
  $LearnedRules \leftarrow LearnedRules \cup Rule$
  $Examples \leftarrow Examples$ - {*Examples covered by Rule*}
  $Rule \leftarrow$ *LEARN-ONE-RULE*$(class, attributes, examples)$
  If Rule contains any previously learned rules, remove these contained rules.
Return LearnedRules

The first modification ensures that we keep learning rules while there are still true matches not covered by a previously learned rule. This forces our blocking scheme to maximize PC for the training data.

SCA learns conjunctions independently of each other, since *LEARN-ONE-RULE* does not take into account any previously learned rules. So we can check the previously learned rules for containment within the newly learned rule. Since each of the conjunctions are disjoined together, any records covered by a more restrictive conjunction will be covered by a less restrictive conjunction. For example, if our current scheme includes the conjunction ({*token-match,last name*} $\wedge$ {*token-match, phone*}) and we just learned the conjunction ({*token-match, last name*}) then we can remove the longer conjunction from our learned rules, because all of the records covered by the more restrictive rule will also be covered by the less specific one. This is really just an optimization step because the same candidate set will be generated using both rules versus just the simpler one. However, fewer rules require fewer steps to do the blocking, which is desirable as a preprocessing step for record linkage.

Note that we can do the conjunction containment check as we learn the rules. This is preferred to checking all of the conjunctions against each other when we finish running SCA. As will be shown later, we can guarantee that each newly learned rule is simpler than the rules learned before it that cover the same attribute(s). The proof of this is given below, where we define the *LEARN-ONE-RULE* step.

Learning each conjunction

As mentioned previously, the algorithm learns each conjunction, or "blocking criteria," during the *LEARN-ONE-RULE* step of the modified SCA.

To make each step generate as few candidates as possible, *LEARN-ONE-RULE* learns a conjunction with as high a reduction ratio (RR) as possible. The algorithm itself, shown in Table 3, is straightforward and intuitive. Starting with an empty conjunction, we try each {method,attribute} pair, and keep the ones that yield the higher RR while maintaining a pairs completeness (PC) above a minimum threshold. Then, we keep adding {method, attribute} pairs to the top conjunctions until the RR no longer improves, while still maintaining a PC greater than the threshold.

Note that *LEARN-ONE-RULE* uses a greedy, general-to-specific beam search. General-to-specific beam search makes each conjunction as restrictive as possible because at each iteration we add another {method, attribute} pair to the best conjunct. Although any individual rule learned by general-to-specific beam search might only have a minimum PC, the disjunction of the final rules, as outlined above, will combine these rules to increase the PC, just as in the multi-pass approach. Thus, the goal of each *LEARN-ONE-RULE* is to learn a rule that maximizes RR as much as it can, so when the rule is disjoined with the other rules, it contributes as few false-positive, candidate matches to the final candidate set as possible. We use a beam search to allow for some backtracking as well, since we use a greedy approach.

The constraint that a conjunction has a minimum PC ensures that the learned conjunction does not over-fit to the data. If this restriction were not in place, it would be possible for *LEARN-ONE-RULE* to learn a conjunction that returns no candidates, uselessly producing an optimal RR.

The algorithm's behavior is well defined for the minimum PC threshold. Consider, the case where the algorithm is learning as restrictive a rule as it can with the minimum coverage. In this case, the parameter ends up partitioning the space of the cross product of example records by the threshold amount. That is, if we set the threshold amount to 50% of the examples covered, the most restrictive first rule covers 50% of the examples. The next rule covers 50% of what is remaining, which is 25% of the examples. The next will cover 12.5% of the examples, etc. In this sense, the parameter is well defined. If we set the threshold high, we will learn fewer, less restrictive conjunctions, possibly limiting our RR, although this may increase PC slightly. If we set it lower, we cover more examples, but we need to learn more

Table 3: Learning a conjunction of {method, attribute} pairs

LEARN-ONE-RULE$(attributes, examples, min\_thresh, k)$
Best-Conjunction $\leftarrow \{\}$
Candidate-conjunctions $\leftarrow$ *all {method, attribute} pairs*
While Candidate-conjunctions not empty, do
  For each ch $\in$ *Candidate-conjunctions*
    If not first iteration
      ch $\leftarrow$ *ch* $\cup$ {method,attribute}
    Remove any ch that are duplicates, inconsistent or not max. specific
    if REDUCTION-RATIO(ch) $>$ *REDUCTION-RATIO(Best-Conjunction)*
    and PAIRS-COMPLETENESS(ch) $\geq$ *min_thresh*
      Best-Conjunction $\leftarrow$ *ch*
  Candidate-conjunctions $\leftarrow$ *best k members of Candidate-conjunctions*
return Best-conjunction

conjuncts. These newer conjuncts, in turn, may be subsumed by later conjuncts, so they will be a waste of time to learn. So, as long as this parameter is small enough, it should not affect the coverage of the final blocking scheme, and smaller than that just slows down the learning. We left this parameter at 50% for our experiments.[1]

Since we have described the greedy algorithm, we can now prove that each rule learned during an iteration is simpler (has less {method, attribute} pairs) than the rules learned before it that contain at least one {method, attribute} pair in common.

Our proof is done by contradiction. Assume we have two attributes A and B, and a method X. Also, assume that our previously learned rules contain the following conjunction, ($\{X, A\}$) and we currently learned the rule ($\{X, A\} \wedge \{X, B\}$). That is, we assume our learned rules contains a rule that is less specific than the currently learned rule. If this were the case, then there must be at least one training example covered by ($\{X, A\} \wedge \{X, B\}$) that is not covered by ($\{X, A\}$), since SCA dictates that we remove all examples covered by ($\{X, A\}$) when we learn it. Clearly, this cannot happen, since any examples covered by the more specific ($\{X, A\} \wedge \{X, B\}$) would have been covered by ($\{X, A\}$) already and removed, which means we could not have learned the rule ($\{X, A\} \wedge \{X, B\}$). Thus, a contradiction, and the assumption is incorrect.

Experimental Evaluation

This section compares our algorithm, which we call BSL (**B**locking **S**cheme **L**earner), to other blocking schemes. We show BSL outperforms blocking schemes built for ad-hoc record linkage experiments. We also demonstrate that BSL performs on par with the laboriously hand-crafted rules of a domain expert.

Our technique uses supervised machine learning, so we must address the issue of the amount training data. Since blocking is a preprocessing step for record linkage, we looked at the record linkage literature, and use the experimental methods most often employed for supervised record linkage. Many of the papers use 2-fold cross validation with two similarly sized folds. This equates to labeling roughly 50% of the matches in the data for training. So, we use 2-fold cross validation, with equally-sized folds, across 10 trials. However, it is unrealistic to label 50% of the data, so we also present results using 10% of the data for training. We use the average reduction ratio (RR) and pairs completeness (PC) for our comparisons.

We also need to mention which methods and attributes we consider for our blocking schemes in each evaluation. To make fair comparisons, the set of {method, attribute} pairs our algorithm chooses from are the cross product of all methods and attributes used by the system we are comparing against. For example, if we tested against an ad-hoc blocking scheme: ({*token-match, last name*}) $\cup$ ({*1st-letter-match, first name*}), then our {method, attribute} pairs to learn from are {*token-match, last name*}, {*token-match, first name*}, {*1st-letter-match, last name*} and {*1st-letter-match, first name*}. This way it is the blocking scheme that improves, rather than better methods and attributes. Note also, in all experiments, we set the K of our beam search to 4 and the minimum threshold for pairs completeness for *LEARN-ONE-RULE* to 50%.

[1] Setting this parameter lower than 50% had an insignificant effect on our results, and setting it much higher, to 90%, only increased the PC by a small amount (if at all), while decreasing the RR.

Our first experiment compares our technique to the blocking method used by the Marlin system (Bilenko & Mooney 2003). For the experiment we use the restaurants data set from the paper, matching records from Fodors to Zagat. The blocking method used by Marlin is an Unnormalized Jaccard Similarity on the tokens between attributes, with a threshold set to 1. This is equivalent to finding a matching token between the attributes, and we call this method *token*. The attributes for this data are {name, address, cuisine, city}. The Marlin system used a blocking scheme meant to maximize the PC, so as not to hinder the record linkage:

$$BS = (\{token, name\}) \cup (\{token, address\}) \cup (\{token, cuisine\}) \cup (\{token, city\})$$

A blocking scheme learned by BSL is:

$$BS = (\{token, name\} \wedge \{token, address\}) \cup (\{token, name\} \wedge \{token, cuisine\})$$

Table 4 shows the performance comparison. The blocking scheme employed by Marlin returned all of the true matches but at the expense of the size of the candidate set. The PC of the generated blocking scheme is 1.84% less, but this only represents missing 1.005 of the true matches on average. So, the generated blocking scheme returned a comparable number of true positives, while considerably reducing the number of candidates. Note that all results are significant with a two-tailed t-test and α=0.05.

Table 4: Blocking scheme results on restaurants

	RR	PC
BSL	99.26	98.16
Marlin	55.35	100.00

We also compare to the blocking scheme used by the Hybrid-Field Matcher (HFM) record linkage system (Minton *et al.* 2005). To compare against HFM we use the Cars data presented in the paper, along with their blocking scheme, which maximized PC. On top of the token method described earlier, the HFM blocking scheme employed two other methods: 1st-letter-match, which we will call *first*, and a synonym match, which is a match if tokens between records are synonyms, such as *hatchback* = *liftback* . In this domain, there are 4 attributes, {make, model, trim, year}. The blocking scheme used by HFM is:

$$BS = (\{token, make\} \wedge \{token, model\} \wedge \{token, year\}) \cup (\{first, make\} \wedge \{first, model\} \wedge \{first, year\}) \cup (\{synonym, trim\})$$

A blocking scheme learned by BSL is:

$BS = (\{token, model\} \wedge \{token, year\} \wedge \{token, trim\}) \cup$
$(\{token, model\} \wedge \{token, year\} \wedge \{synonym, trim\})$

Again, the comparison in Table 5 shows that both perform equally well in terms of PC. In fact, there is no significant difference statistically with respect to each other, using a two-tailed t-test with α=0.05. Yet, despite the similar PC, our generated rules increase the RR by more than 50%.

Table 5: Blocking scheme results on cars

	RR	PC
BSL	99.86	99.92
HFM	47.92	99.97

In the final comparison, we use synthetic Census data, called "dataset4" from (Gu & Baxter 2004). Gu and Baxter present their *adaptive filtering* method using one method, bigram indexing (Baxter, Christen, & Churches 2003) on one attribute. We can not generate a different blocking scheme from theirs since there is only one method and one attribute to choose from. Instead, we took the attributes and methods from the 11 blocking criteria presented in (Winkler 2005) for matching the 2000 Decennial Census to an Accuracy and Coverage Estimation (ACE) file. We remove the *2-way switch* method because it is specific to two attributes, and we focus on methods that apply generally to the attributes. This left 3 methods to use: {*token, first, first-3*}. Token and first are described above. *First-3* is similar to first, but it matches the 1st three letters of an attribute, rather than just the 1st one. There are 8 attributes to choose from: {first name, surname, zip, date-of-birth, day-of-birth, month-of-birth, phone, house number}.

From these method and attributes, we learn blocking schemes such as the following:

$BS = (\{first\text{-}3, surname\} \wedge \{first, zip\}) \cup$
$(\{token, house\ number\} \wedge \{first\text{-}3, phone\}) \cup$
$(\{token, house\ number\} \wedge \{first\text{-}3, date\text{-}of\text{-}birth\}) \cup$
$(\{first\text{-}3, first\ name\}) \cup$
$(\{token, phone\}) \cup$
$(\{token, day\text{-}of\text{-}birth\} \wedge \{token, zip\} \wedge$
$\{first\text{-}3, date\text{-}of\text{-}birth\}) \cup$
$(\{token, house\ number\} \wedge \{first, first\ name\})$

For comparisons we present our results against a blocking scheme composed of the best 5 conjunctions of (Winkler 2005). As stated in the paper, the "best five" blocking scheme is:

$BS = (\{token, zip\} \wedge \{first, surname\}) \cup$
$(\{token, phone\}) \cup$
$(\{first\text{-}3, zip\} \wedge \{token, day\text{-}of\text{-}birth\} \wedge$
$\{token, month\text{-}of\text{-}birth\}) \cup$
$(\{token, zip\} \wedge \{token, house\ number\}) \cup$
$(\{first\text{-}3, surname\} \wedge \{first\text{-}3, first\ name\})$

Note that the blocking criteria constructed by Winkler are meant for real census data, not the synthetic data we use. However, the best-five rule still does impressively well on this data, as shown in Table 6. This is not surprising as Winkler is regarded as an expert on matching census data, and the synthetic data is not as difficult to match as the real census data. Encouragingly, Table 6 also shows our generated rules performed better on PC and only slightly less on RR as compared to the domain expert. Note that the RR and PC results are statistically significant with respect to each other.

Table 6: Blocking scheme results on synthetic census data

	RR	PC
BSL	98.12	99.85
"Best five"	99.52	99.16
Adaptive Filtering	99.9	92.7

Table 6 also compares the *adaptive filtering* method (Gu & Baxter 2004) to our method. Although the adaptive filtering results are for all the records in the set, rather than half as in our 2-fold cross validation, we still think comparing these results to ours sheds some insight. Observe that adaptive filtering maximizes RR tremendously. However, we believe that it is misguided to focus only on reducing the size of the candidate set, without addressing the coverage of true matches. In their best reported result (shown in Table 6), adaptive filtering achieves a PC which is 92.7. That would represent leaving out roughly 365 of the 5000 true matches in the candidate set.

As stated in the beginning of this section, we think training on 50% of the data is unrealistic. Labeling only 10% of the data for training represents a much more practical supervised learning scenario, so we ran our experiments again using 10% of the data for training and testing on the other 90%. Table 7 compares the results, averaged over 10 trials.

Table 7 shows that the algorithm scales well to less training data. In the cars and census experiments, the degradation in performance for the learned blocking schemes is small. The more interesting case is the restaurants data. Here the blocking scheme trained on 10% of the data did not learn to cover as many true matches as when we used 50% of the data for training. In this case, 10% of the data for training represents 54 records and 11.5 matches, on average, which was not enough for BSL to learn as good a blocking scheme. However, the small size of the restaurant data set makes the result appear worse than it is. Although the percentage drop in PC seems large, it really only represents missing 6.5 true matches on average versus missing 1.

This highlights a problem with our approach to learning blocking schemes. The learned blocking scheme's ability to cover true matches is limited by the number of true matches supplied for training. This is usually a concern with supervised learning techniques: the robustness of the learning is constrained by the amount of training data. In the case of the restaurant data, because the data set is small this problem can be highlighted easily. As Figure 2 shows, as we increase the training data, we can improve the performance of the PC.

Table 7: Blocking scheme results varying the amount of training data

Data Set (Amnt. training data)	RR	PC
Cars (50%)	99.86	99.92
Cars (10%)	99.87	99.88
Census (50%)	98.12	99.85
Census (10%)	99.50	99.13
Restaurants (50%)	99.26	98.16
Restaurants (10%)	99.57	93.48

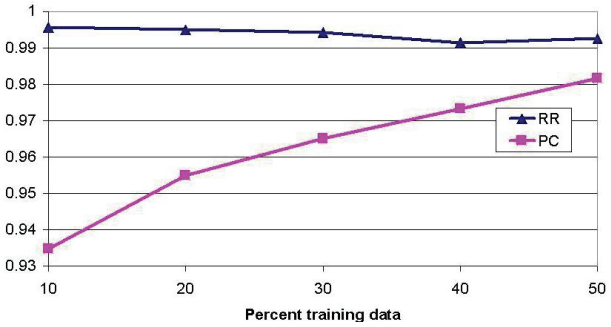

Figure 2: Graph of RR and PC for varying training data percentages on the restaurant data

Related Work

While there is little work on learning blocking schemes, there is research on improving methods to compare attributes. As already stated, (Baxter, Christen, & Churches 2003), created the bi-gram indexing method and Gu and Baxter (Gu & Baxter 2004) provided method refinements. Also, both (McCallum, Nigam, & Ungar 2000) and (Cohen & Richman 2002), have investigated novel clustering methods that apply to the blocking problem. Also there is the Sorted Neighborhood method which uses a sliding window over sorted records (Hernández & Stolfo 1998). Our framework is general enough that any of these methods can be included.

Our blocking scheme generation approach is similar to feature selection. Two good surveys of feature selection are (Dash & Liu 1997) and (Guyon & Elisseeff 2003). However, our problem differs slightly in that not only do we need to discover which features (attributes) of the data set to use, but we also need to discover which methods to use, all while optimizing the pairs completeness and reduction ratio.

Conclusion

In this paper, we presented an algorithm to learn good blocking strategies rather than creating them by hand, in an ad-hoc manner. We showed that our technique improves over the hand-generated blocking schemes produced by non-domain experts, and is comparable to those produced by a domain expert. We also demonstrated that our blocking scheme fulfills the two main goals of blocking: maximizing both PC and RR. As a further bonus, our generated blocking schemes are human readable, which can provide investigators some intuition into how to cluster or match their data.

In the future, we want to investigate using capture-recapture methods to estimate the PC of the generated rules (as done in (Winkler 2005)). This would make it possible to create an unsupervised and fully automatic method of learning blocking schemes. We also want to investigate the inclusion of more complicated blocking methods, such as canopy clustering (McCallum, Nigam, & Ungar 2000).

References

Baxter, R.; Christen, P.; and Churches, T. 2003. A comparison of fast blocking methods for record linkage. In *Proceedings of the ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Identification*.

Bilenko, M., and Mooney, R. J. 2003. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of ACM SIGKDD-03*, 39–48.

Cohen, W. W., and Richman, J. 2002. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of ACM SIGKDD-02*, 475–480.

Dash, M., and Liu, H. 1997. Feature selection for classification. *Intelligent Data Analysis* 1(3):131–156.

Elfeky, M. G.; Verykios, V. S.; and Elmagarmid, A. K. 2002. TAILOR: A record linkage toolbox. In *Proceedings of 18th International Conference on Data Engineering (ICDE)*.

Gu, L., and Baxter, R. 2004. Adaptive filtering for efficient record linkage. In *Proceedings of of the 4th SIAM International Conferece on Data Mining*.

Guyon, I., and Elisseeff, A. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research* 3:1157–1182.

Hernández, M. A., and Stolfo, S. J. 1998. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery* 2(1):9–37.

McCallum, A.; Nigam, K.; and Ungar, L. H. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of ACM SIGKDD-00*, 169–178.

Minton, S. N.; Nanjo, C.; Knoblock, C. A.; Michalowski, M.; and Michelson, M. 2005. A heterogeneous field matching method for record linkage. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM)*.

Mitchell, T. M. 1997. *Machine Learning*. New York: McGraw-Hill.

Newcombe, H. B. 1967. Record linkage: The design of efficient systems for linking records into individual and family histories. *American Journal of Human Genetics* 19(3).

Winkler, W. E. 2005. Approximate string comparator search strategies for very large administrative lists. Technical report, Statistical Research Report Series (Statistics 2005-02) U.S. Census Bureau.