A Heterogeneous Field Matching Method for Record Linkage

Steven N. Minton and Claude Nanjo
Fetch Technologies
2041 Rosecrans Ave., Suite 245
El Segundo, CA 90245
{sminton, cnanjo}@fetch.com

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

Abstract

Record linkage is the process of determining that two records refer to the same entity. A key subprocess is evaluating how well the individual fields, or attributes, of the records match each other. One approach to matching fields is to use hand-written domain-specific rules. This "expert systems" approach may result in good performance for specific applications, but it is not scalable. This paper describes a new machine learning approach that creates expert-like rules for field matching. In our approach, the relationship between two field values is described by a set of heterogeneous transformations. Previous machine learning methods used simple models to evaluate the distance between two fields. However, our approach enables more sophisticated relationships to be modeled, which better capture the complex domain specific, common-sense phenomena that humans use to judge similarity. We compare our approach to methods that rely on simpler homogeneous models in several domains. By modeling more complex relationships we produce more accurate results.

1. Introduction

Record linkage is the process of recognizing when two database records are referring to the same entity. Record linkage is a fundamental problem when integrating multiple information sources. For example, one datasource may refer to "International Animal Productions" headquartered in "Los Angeles, CA", and another may refer to the same company as "Intl. Animal" of "Hollywood". The general problem of recognizing two references to the same entity occurs widely, and variants of the problem include "deduplication" [5, 9, 11], "object identification" [6], and "coreference resolution" [8].

Union Switch and Signal	2022 Hampton Ave	Manufacturing
JPM	115 Main St	Manufacturing
McDonald's	Corner of 5th and Main	Food Retail

Joint Pipe Manufacturers	115 Main Street	Plumbing Manufacturer
Union Sign	300 Hampton Ave	Signage
McDonald's Restaurant	532 West Main St.	Restaurant

Figure 1. Matching Records in Two Tables

A critical part of matching two records is evaluating how well the individual *fields* (i.e., attributes) match. Record linkage systems generally employ similarity metrics that compare pairs of field values, such as two addresses, and return a measure of their similarity. Given field-level similarity judgments, an overall record-level judgment is made.

A common commercial approach to measuring field similarity writes domain-specific rules for each field. For example, when comparing person names, heuristic rules specify the relative importance of last names versus first names, handle culturally-specific cases (e.g., Spanish surnames), etc. This "expert systems" approach can produce acceptable performance, but developing and maintaining such rules is time-consuming and difficult.

Machine learning researchers have taken a different tack

to record linkage. They have developed systems that use sophisticated decision-making methods at the record-level, such as decision trees (e.g., [12]), support vector machines (e.g., [1]) and unsupervised statistical methods (e.g., [10]). However, for field-level similarity judgments, they have primarily relied on simple generic methods, such as TF-IDF and other string-similarity metrics. Unlike expert systems, these tend to be generic and homogeneous for each field.

Unfortunately, simplistic homogeneous models cannot capture many important fine-grained phenomena. For example, consider Figure 1, which presents two tables listing businesses. Each table contains three attributes: name, address, and business type. Note that JPM and Joint Pipe Manufacturers are the same business, while "Union Switch and Signal" is not the same as "Union Sign". We can make this judgment because JPM is clearly an acronym for Joint Pipe Manufacturers, and the two records have minor variations of the same address and business type. In contrast, "Union Switch and Signal" and "Union Sign" share only textual similarities in their name and address.

Simple homogeneous metrics cannot make these types of common-sense determinations. We believe that it is important to accurately model the specific relationships between values at the field-level, including phenomena such as acronyms, synonyms, nicknames, spelling errors, etc, in order to distinguish between a meaningful match and a surface similarity. We refer to the process of establishing the relationship between two values as "field matching". We hypothesize that doing a better job of field matching will enable more accurate record linkage.

In this article, we present a new algorithm for field matching, HFM (Hybrid Field Matcher), that combines the best aspects of the machine learning and expert systems approaches. We use a library of heterogeneous "transformations" that enables us to capture complex relationships between two field values, like the expert systems approach. Machine learning techniques are then used to automatically customize these transformations for a given domain, so that highly accurate decisions can be made. HFM builds upon previous work in the Active Atlas system [12], where this approach was originally explored in a more primitive form.

In the next section, we present our record linkage approach, and then focus our discussion on the HFM field matching component. We show experiments that HFM produces superior results in domains where simpler field matching metrics, including Active Atlas, fail to capture important distinctions.

2. Overview Of Our Approach

The record linkage approach described here assumes that we are linking records in two database tables, A and B, such that there are corresponding attributes in each table, as in the example in Figure 1. That is, the *i*th column in each table contains elements of the same type. In many applications, there are additional complexities; for instance, one table might have two attributes, such as "first name" and "last name", and the other table might have attributes such as "full name". These complexities can generally be handled in a pre-processing phase (e.g., concatenating "first name" and "last name"), and for this reason our approach is applicable in a wide variety of settings.

Our record linkage process has several phases. First, we parse each cell in each record into a set of tokens, where each token is an individual word, number, or symbol. Optionally, we also label the tokens with a semantic category (e.g., parsing a full name into first name, optional middle initial, and last name), and also optionally apply a set of normalization operators to standardize the tokens.

Second, we use a blocking algorithm to identify pairs of records that have the potential to match. This eliminates the need to evaluate the entire cross product. In our implementation, we use a reverse index to identify potentially matching pairs, similar to the methodology described in [12] which takes into account many of the transformations described in the next section.

Next, we take each pair of candidate records (A_j, B_k) and compare them field by field. I.e., we evaluate each pair of values, (A_{ji}, B_{ki}) using a learned distance metric F_i. Creating this distance metric is the focus of this paper.

Once we have computed the distance for each field of the record pair, we use a support vector machine to determine the overall goodness of the match. In the following section, we describe how the distance metric F_i is learned.

2.1. Training the Field Learner

The key to our approach is the use of *transformations* to relate two values. For instance, Figure 2 shows how we might relate the two company names "Intl. Animal" and "International Animal Productions". "Intl." is an abbreviation for "International", the word "Animal" is found in both, and "Productions" is missing. Transformations that we use for string values include: **Equal, Synonym, Misspelling, Abbreviation, Prefix, Acronym, Concatenation, Suffix, Soundex** and **Missing**. These transformations are generic, in the sense that they may be applicable in any domain. A few of them, however, require domain-specific customization. For example, the **"Synonym"** transformation requires an association list of synonyms to be developed for each domain. (These customizations can be made manually, or in many cases, automatically learned. However, learning methods for individual transformations are not the focus of this paper.)

Given two values, *a* and *b*, which each consist of a sequence of tokens (numbers or strings), a transformation *t* maps tokens in *a* to tokens in *b*. (A special transformation,

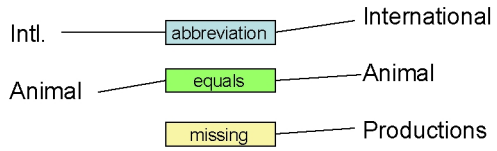

Figure 2. A Transformation Graph

"Missing", can map a token to the empty set.) A *transformation graph* G is a set of transformations relating two tokenized values, *a* and *b*, such as the set of transformations shown in Figure 2, or the set shown in Figure 3. Obviously, for any given pair of values, there is a potentially large space of possible transformation graphs, each one representing a different way to relate the tokens. The key questions we consider here are how to build the transformation graph that best represents the actual relationship between a pair of values, and how to "weight" the transformations in a transformation graph when evaluating whether a pair of values match or not. We can then determine, for instance, whether a pair with a spelling mistake and a synonym is a closer match than a pair with one equal word and one missing word.

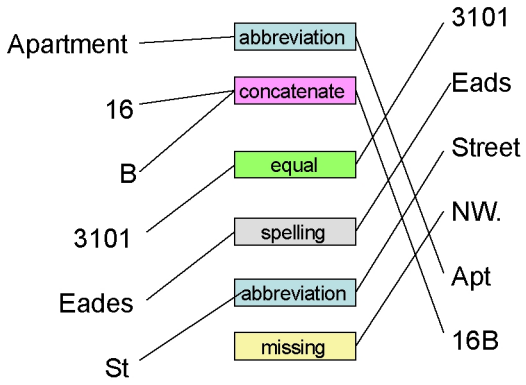

Figure 3. Another Transformation Graph

Our approach employs supervised learning. For training data we use a set of records that have been previously linked, so that we know which records are matches and which are not. Every pair of field values (*a*, *b*) in each record pair that satisfies the blocking criterion constitutes a training example. If the pair (*a*, *b*) is found in at least one pair of matching records, then it is labeled as positive, otherwise it is assumed to be negative. (Note that two field values may match, even though the records they are in do not match. Because we are training at the field level, rather than the record level, we cannot be sure that a training example is negative. This invariably generates some noise.) We also assume that we have a generic preference ordering over the types of transformations, such that $T_1 > T_2$ implies that a transformation of type T_1 is generally more indicative of a match than transformation of type T_2. For example, the **"Equal"** transformation is generally more indicative of a match than a **"Synonym"** transformation, so **Equal** > **Synonym**. Similarly, **Synonym** > **Misspelling**, and **Misspelling** > **Missing**. In general, it is straightforward to choose a reasonable total order. (Because the learning system eventually assigns its own weights, it is not necessary to get the order exactly "right" in order for the system to produce reasonable results.)

For each training pair (*a*,*b*) we build a transformation graph which relates *a* to *b*. A transformation graph is *complete* if every token in *a* and in *b* participates in a transformation and *consistent* if no token participates in more than one transformation. To find a complete and consistent transformation graph, we use a greedy algorithm that starts with a null graph (i.e,. an empty set of transformations), and incrementally builds a complete, consistent transformation graph. The algorithm considers in turn each transformation type in the total order, starting with the **Equal** transformation, which is the highest weighted transformation type. For each transformation type, the algorithm considers the available tokens (which are not yet participating in any transformation) and selects a consistent transformation of that type and adds it to the set of transformations in the graph. Since the least-preferred transformation type, **Missing**, can be applied to any token, this procedure necessarily produces a complete, consistent transformation graph.

Once we have constructed a transformation graph for each matched example pair (our positive examples) and each unmatched example pair (our negative examples), we can determine how to weight each transformation type when scoring a new, unseen example. In particular, given an unlabeled example (*a*, *b*), which we want to classify as either matched M or as not matched ¬M, we first create a transformation graph using the procedure above where t_1, t_2,...,t_n are transformations in the graph. Let **V** be a corresponding set of variables such that the value of v_i is the type of ith transformation in the graph, e.g., if t_2 is a **Synonym** transformation, then v_2 = **Synonym**. To score an unseen example, (*a*,*b*), we can estimate the probability that (*a*,*b*) is a match given the types of transformations in the graph, that is $p(M| v_1, v_2, ...,v_n)$. We make the Naïve Bayes assumption that the probability of the transformations are independent given the classification of the example, i.e.,

$$p(M|\mathbf{V}) = p(M|v_1, v_2, ..., v_n) = \frac{p(M)\prod_{i=1...n} p(v_i|M)}{\prod_{i=1...n} p(v_i)}$$

Since the order of the transformations is arbitrary, for any *i*, we estimate $p(v_i \mid M)$ by counting the number of transformations of each type observed in the training data.

This approach is similar to the use of Naïve Bayes for text learning, but rather than using *words* as features, we use

transformations as our features. Intuitively, the transformations that are ranked higher in the preference ordering, such as **Equal**, **Synonym**, etc, will be relatively more common in matched pairs compared to lower ranked transformations, such as **Missing**. P(M|**V**) can then be used as a measure of the "distance" between two attribute values. Finally, a normalized value for the score is calculated as follows:

$$Score_{HFM} = \frac{p(M|\mathbf{V})}{p(M|\mathbf{V}) + p(\neg M|\mathbf{V})} =$$

$$\frac{p(M)\prod_{i=1}^{n} p(v_i|M)}{p(M)\prod_{i=1}^{n} p(v_i|M) + p(\neg M)\prod_{i=1}^{n} p(v_i|\neg M)}$$

For example, consider a case where we are evaluating the similarity between two restaurant names: "Giovani Italian Cucina Int'l" and "Giovani Italian Kitchen International". In a fashion similar to what was shown in Figure 2, the transformation graph consists of two **Equal** transformations (Giovani to Giovani, Italian to Italian), one **Synonym** transformations (Cucina to Kitchen), and one **Abbreviation** transformation (Int'l to International). Let us assume that in the training set, the Equal transformation occurs with a probability of 0.17 in the match set but only with a probability of 0.027 in the non-match set. Similarly the Synonym transformation occurs with a probability of 0.29 in the match set and a probability of 0.14 in the non-match set and the Abbreviation transformation occurs with a probability of 0.11 in the match set as opposed to 0.03 in the non-match set. Also assume that the probability of the pair belonging to the match set P(M) is 0.31 whereas the probability of belonging to the non-match set is P(¬M) = 0.69. Then, we get P(M) $\prod$ (v_i| M) = (0.31)(0.17)2(0.29)(0.11)= 2.86E-4 and P(¬M) $\prod$ p(v_i|¬M) = (0.69)(0.027)2(0.14)(0.03)= 2.11E-6. This results in a score of 0.993 for the pair, which is highly indicative of a match.

2.2. Finer-Grained Transformations

The approach described in the previous section is not sensitive to word frequency. As a result, when two person names share a token such as "Smith" the algorithm weights these as heavily as two names that share the token "Ivanescuskka". Similarly, when matching company names, the token "Incorporated" is weighted just as heavily as "Corrugated".

This problem can be dealt with by developing finer-grained transformations that capture these distinctions. In particular, rather than treating all tokens similarly, we can categorize tokens based on their frequency and refine the transformations accordingly. For example, in the experiments described later, we assigned tokens into one of three categories based on frequency: low-frequency, medium-frequency and high-frequency.

The transformations can then be refined so that they are applicable only to specific token categories. For example, we create three forms of the **Equal** transformation, one for high-frequency tokens, one for intermediate-frequency tokens, and one for low-frequency tokens. We can do the same for the **Missing** transformation. In general, this partitioning strategy can be applied to other transformations but in practice we only use it for **Equal** and **Missing** because it reduces the amount of training data per category.

The same concept can be adapted for other distinctions as well. For example, consider two street addresses with surface similarities, such as "201 Carte St., Apt. 23" and "Apt 201, 23 Carte St". A string comparison system that counted the number of shared tokens would consider these strings to be highly similar. A more intelligent approach is to first parse the strings into tokens that are labeled with their semantic category, such as "house number", "street name" and "apartment number". We can then refine the transformations accordingly. For example, we can have two versions of the **Equal** transformation, one for tokens that are equal and assigned the same semantic category, and one for tokens that are equal but not assigned the same semantic category.

2.3. Global Transformations

In some cases, transformations between individual tokens or sets of tokens cannot fully describe the relationship between two field values. In such cases, we have found it useful to augment our set of transformations with "global transformations". These transformations are applied after the regular transformation graph is computed, and augment the original graph. For instance, in some domains it is important to distinguish cases where *all* the tokens from one value are involved in transformations other than **Missing**; i.e., one value is essentially a subset of the other. For this case we have a global "subset" transformation, that augments the regular transformations. Another global transformation, the **Reordering** transformation, is useful for testing whether the corresponding tokens in two values are in the same order or not. Domain-specific versions of this transformation are also useful for testing the ordering of specific semantic categories, for example, whether first name proceeds last name.

2.4. Comparison to Active Atlas Algorithm

This work improves upon ideas explored by Tejada et al. [12] in the Active Atlas system. Active Atlas evaluated the similarity between field pairs using transformations, as in HFM. The fine-grained transformations and global transfor-

mations introduced in the previous sections are new capabilities, however. Moreover, Active Atlas used a more primitive learning scheme, in which the weight of each transformation was determined by comparing its frequency in the positive labeled matches versus its overall frequency (in both positive and negative matches). In comparison, HFM considers two additional pieces of information when calculating the similarity between field pairs.

First, HFM considers the likelihood that any given pair is a match given the distribution of matches in the training data, independent of the transformation frequency distributions. Because Active Atlas uses only transformation frequencies, it may be misled. For example, if a transformation occurs equally frequently in the match and non-match set, than Active Atlas would suggest an equal likelihood that the pair matches or does not. However, if beyond the transformation distribution we know that field pairs are highly unlikely to match because there are many less matching pairs in the labeled data, HFM would reflect this with a lower similarity score.

Second, HFM also considers the "saturation" and "dilution" of transformations when calculating the similarity score. Consider the number of occurrences of a transformation f in the match set and the non-match set to be f^+ and f^- respectively, and the occurrences of all transformations in the match set and non-match set to be F^+ and F^-. Furthermore, define R^- as the ratio between f^- and F^- and R^+ as the ratio between f^+ and F^+. Now, assume f^+ and f^- remain almost constant while F^+ remains constant and F^- increases. Such a situation could happen if we introduce a new transformation(s) that almost exclusively applies to the non-match set. In this case, R^- decreases, since f^- composes less of the total transformations in F^-. We call this situation "dilution." However, since the R^+ remains constant, the difference between R^+ and R^- increases. This increase implies that f is more indicative of a match because R^+ has increased relative to R^- and the similarity score should then increase to reflect this idea. Similarly, if we add a transformation that applies almost exclusively to F^+ we have a "saturated" situation, so the score should decrease. HFM reflects this expected behavior, while Active Atlas gives the same score in the "saturation" and "dilution" cases since the new transformation(s) does not apply.

3. Experimental Results

In a set of experiments we evaluated HFM and compared it to three alternative approaches. One alternative is TF-IDF, also known as vector-based cosine similarity. This is a simple but effective metric for gauging the similarity of two tokenized strings frequently used in information retrieval and integration. TF-IDF calculates the similarity between two strings as the ratio of the matching tokens between the strings over the frequency of those matching tokens in the whole record set. For our implementation we measure all document frequency statistics from the full corpus of records to be matched following the approach in [3].

We also compared HFM to the learned string edit metric used in Marlin [1]. Marlin is notable because it is one of the few systems that attempts to move beyond generic field-level textual similarity. Specifically, Marlin uses a variant of string edit distance (with affine gaps), where substitution, deletion and insertion operators have learned weights.

Both TF-IDF and Marlin are relatively homogeneous in their linkage approach. In fact, TF-IDF applies the same similarity metric in every case. Marlin is more sophisticated, distinguishing between insertions, deletions, substitutions and matches, which can be weighted differently. However, in comparison to HFM, there is no attempt to explicitly model a variety of phenomena, such as acronyms, synonyms, misspellings, etc. That is, Marlin works at the "string edit" level, whereas HFM operates at a higher level where transformations such as acronyms and synonyms exist.

Finally, we compare our approach to Active Atlas' field matching algorithm. This allows for a comparison between previous work and the significantly refined HFM technique. We note that in Active Atlas, the field matching component was embedded in a an ambitious active learning system, and was never directly compared to alternative techniques. For our Active Atlas comparison, we used the same basic transformations, but as described earlier, a more primitive weighting scheme was used. Also, global transformations were not employed in Active Atlas, and finer-grained distinctions such as frequency were omitted.

Each approach calculates a vector of feature scores for each record. These feature vectors are then passed to a Support Vector Machine (SVM) trained to classify them as matches or non-matches. In all of the experiments the SVM^{Light} [7] implementation[1] was used, with a Radial Bias kernel function, where γ = 10.0. Following the procedure of [1], we built curves of interpolated precision at given recall by comparing the labeled data to the matches produced by the SVM.

We used a blocking stage to generate candidate matches because comparing the Cartesian product of records is infeasible. Marlin employs its own blocking scheme. HFM does blocking according to the technique described in the **Overview** section. We needed a blocking scheme for TF-IDF (to avoid overwhelming our SVM implementation with too much data), so TF-IDF was provided with the candidates generated by HFM's blocking mechanism to allow for a fair comparison.

We compared the three approaches on four data sets.

[1] http://svmlight.joachims.org/

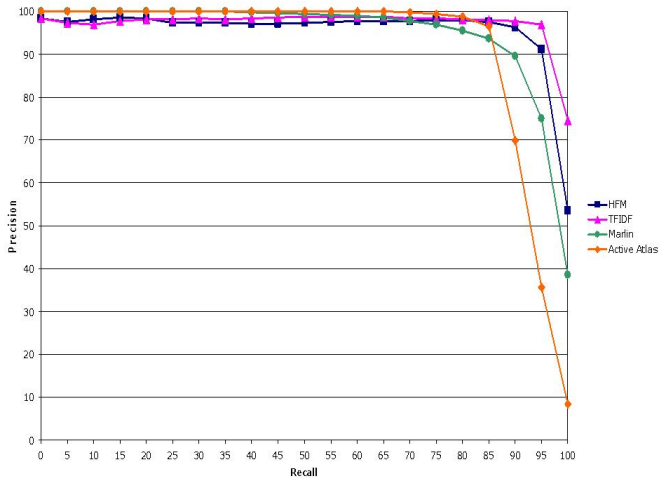

Figure 4. "Marlin Restaurants" Results

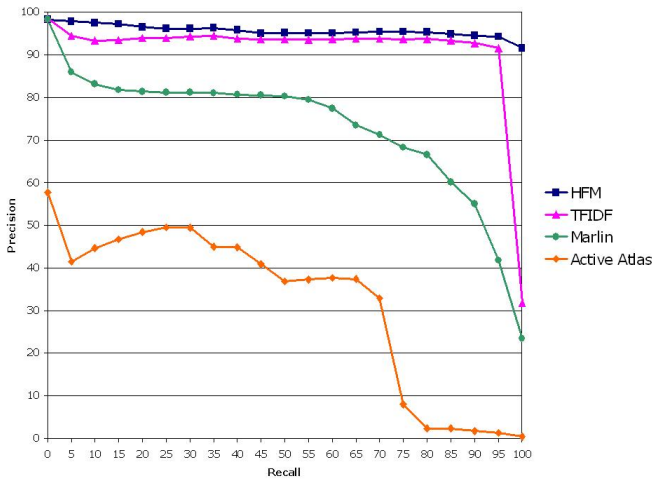

Figure 5. MD Restaurant Subset Results

For each domain, 20 experimental trials were run using the same cross-validation procedure described in the Marlin experiments [1]. Basic results are reported in Table 1. Each entry in the table contains the average of maximum F-measure values over the 20 evaluated folds. The best performer in each domain is shown in **bold**. All comparisons except for one (i.e., pairs of results) are significant at the 0.05 level using a 1-tailed paired t-test. The single insignificant result (shown in *italics*) is between Marlin and Active Atlas on the "Marlin restaurants" dataset.

The first dataset, "Marlin Restaurants" (MR), was a relatively smaller dataset used in [1] to evaluate Marlin. This data set consists of two tables of restaurants, one from Fodors and one from Zagats. Each table has four attributes: name, address, city and cuisine. The tables have 534 records and 330 records, respectively, and there are 112 matches. The results on this data set are shown in Figure 4.

This data set is fairly easy, and all of the systems do quite well. Surprisingly TF-IDF performed best by a small margin, but the number of matches (112) is so small that the difference came down to just a few examples. To create a more substantial test in the same domain, we created a second data set consisting of "restaurant names" and "restaurant addresses" from the LA County Health Dept Website and Yahoo LA restaurants. These tables contained 3701 records and 438 records, respectively, and they share 303 matches. The results for all 4 systems are reported in Figure 5.

In this data set there was a good deal of similarity between the addresses of matching records (i.e., many equal tokens), to the point that address alone could generally be used to discriminate matches. Due to this, and the fact that addresses have large numbers of tokens (so that matching addresses almost always had several equal tokens), TF-IDF performs quite well. However, it was unable to catch some hard cases that HFM recognized. These cases required additional transformations such as Abbreviation and Synonym which are not available to TF-IDF and Marlin. This is exhibited especially at the highest levels of recall where TF-IDF could no longer discriminate between matches and non-matches. Marlin, which considers whole fields as a single string, does not benefit from this token level similarity either.

One point of interest is the poor performance of Active Atlas. Active Atlas only regards the frequencies of transformations in the match and non-match set independent of the relative frequencies those transformations exhibit versus all other transformations that apply. This is a problem when the number of non-matches in the candidate set far outweighs the number of matches. This happens in the larger restaurant dataset where many restaurants share similar names but different addresses. This is one example of HFM's improvement over the original Active Atlas algorithm as described in Section 2.

The third dataset consisted of records describing automobile models from the Edmunds site and Kelly Blue Book site. The Edmunds data source consisted of 3171 records, containing make, model, trim and year, while the Kelly Blue Book data set had 2777 records, with the same attributes. Between them, there are 2909 matches, because it is possible to have a 1-to-N matching.

Figure 6 shows the experimental results for the car domain. In this domain, HFM greatly outperforms TF-IDF, due to the frequency of difficult transformations between the Trim field. Trim benefited from the synonym transformation, which, for example, maps "Hatchback" to "Liftback". It also benefited from the concatenation transfor-

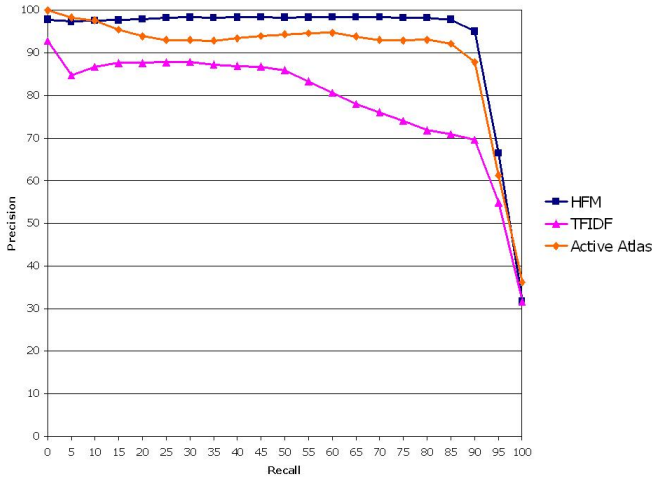

Figure 6. Cars Results

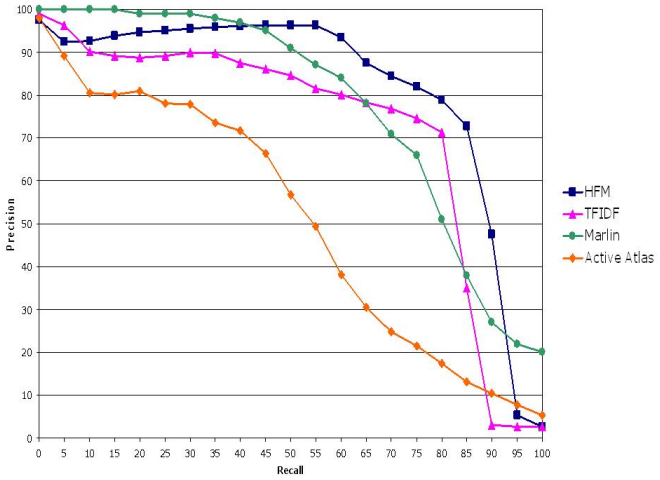

Figure 7. Bidding For Travel Results

Matching Technique	Domain			
	Marlin res.	MD res.	Cars	BFT
HFM	94.64	**95.77**	**92.48**	**79.52**
Active Atlas	*92.31*	45.09	88.97	56.95
TF-IDF	**96.86**	93.52	78.52	75.65
Marlin	*91.39*	76.29	N/A	75.54

Table 1. Average maximum F-measure for detecting record matches

mation, which recognizes the similarity between "200 ZX" and "200ZX". These transformations allowed HFM to score fields more accurately, which led to better record level matching results. Unforunately, Marlin did not scale for this dataset, therefore no comparisons were possible.

The fourth domain is based on posts about hotels appearing on the "Bidding For Travel" (BFT) website. Each one of these posts was manually parsed into a star rating, a hotel name and a hotel area. Since this data comes from user-entered text, there are numerous misspellings, abbreviations and other such transformations. We try to match these parsed fields to a clean data set of star ratings, hotel names and hotel locations, with no duplicates. The "posts" data set consists of 1125 records, and the "clean" data set has 132 records. There are 1028 matches between the sets. Note that not all posts have a match in the clean set, and each "clean" record can match many "posts". The results are reported in Figure 7.

As in the cars domain, there were certain transformations unique to HFM that helped it outperform TF-IDF. For example, the concatenation transformation identified similarities such as "Double Tree" to "Doubletree" on a hotel name, and the abbreviation transformation was able to relate "Dwntwn" and "DT" to "Downtown". Even though both techniques exhibit low precision at a very high level of recall, HFM still maintains a significant increase in precision up to that point. Specifically, it has a higher average maximum F-Measure, as shown in Table 1. Marlin was able to outperform HFM at the highest and lowest levels of recall. The many misspellings amongst the fields are suitable for Marlin's approach of adaptive edit distance, which is able to capture these types of data discrepancy. However, HFM has difficulty in the case where multiple corrections are required for a given word, since it does not (yet) allow multiple transformations per token.

4. Related Work

Most record linkage systems use fairly simple textual similarity measures at the field level. Previous work in machine learning has primarily focused on applying intelligence at the record level, that is, determining how best to discriminate matches based on the multiple fields being compared. Our approach attempts to improve the accuracy of the record linkage process by improving the performance of an underlying field-level matcher.

Some previous systems have been built with this goal in mind. For example, the Marlin system [1] attempts to move beyond trivial field-level textual similarity. As described earlier, their approach is similar to ours in that it learns a similarity metric between two fields, then uses an SVM to determine if two records refer to the same entity. However, one difference between Marlin and HFM is that HFM uses a heterogeneous set of models to identify complex relationships between two values, whereas Marlin focuses on refining homogeneous string-based similarity metrics.

Cohen and Richman [4] presents an adaptive approach for identifier name matching that has some similarity to HFM in that the system employs a set of complex features such as "substring match", "prefix match", "edit distance", and so forth in its learning scheme. However, HFM does

not simply use multiple features; instead, for each pair it builds a transformation graph that can fully relate two complex values, a more ambitious goal.

The work here was largely inspired by the previous work of Tejada, Knoblock and Minton, called Active Atlas, in which transformations related two fields[12]. That work developed an initial approach to learning transformations, which led to the Naive Bayes approach described here. Active Atlas system is in some respects more ambitious than the method described here, in particular because it attempted to simultaneously refine a field similarity metric (using weighted transformations) as well as a record-level decision procedure (via decision trees) using an active learning approach. In comparison, for the work described here we used a more straightforward supervised learning approach. This enabled us to focus on the contribution of the transformation weighting scheme, and in particular, carry out a detailed comparison of HFM to other methods.

An alternative approach to using transformations is to normalize the data prior to record linkage [2]. For instance, as the data is parsed, one could expand all abbreviations as they are encountered. In general, if values can be reliably normalized prior to the record linkage process, it makes sense to do so. However, this strategy has limited applicability. There may be no normal form in some cases. For instance, the name "Caitlyn" may be associated with both the synonyms "Catherine" and "Lynne" (which in turn have widely differing sets of variants). And there are many domains in which important transformations may only be identified in the context of a pair of values, such as misspellings and acronyms.

Similarly, some values can be parsed into separate fields prior to record linkage, such as full name being parsed into first, middle and last names. However, if the parsing is not 100% reliable, this can create problems. In contrast, the technique introduced here (where parsed components are tagged and used to refine the transformations) is more robust with respect to such errors. For example, if "Steven Aster Tate" is parsed so that "Aster Tate" is the last name, and compared to "Steven A. Tate", where "Tate" is parsed as the last name, the error will be difficult to recover from if there are three separate fields, as compared to a single field.

5. Discussion

As we have shown, the ability to identify complex relationships between two values can make an important contribution to record linkage. In some domains, like the restaurant domain, the improvement is minor, because matching values normally have enough commonality that they can be identified using simple metrics. However, in other domains, like the car domain, the added power of transformations can result in larger improvements.

This contribution is meaningless if the technique can not scale to sizable data sets, which are more representative of the record linkage task in the real world. As shown experimentally, HFM is also able to perform well when the data sets are large and complex. This is important as information integration matures into real world domains where the data sets are huge and varied.

6. Acknowledgements

This research is based upon work supported in part by the United States Air Force under contract number F49620-02-C-0103, in part by the Air Force Office of Scientific Research under grant number FA9550-04-1-0105, and in part by the National Science Foundation under Award No. IIS-0324955. 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 any of the above organizations or any person connected with them.

References

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

[2] P. Christen, T. Churches, and J. X. Zhu. Probabilistic name and address cleaning and standardization. In *Proceedings of the Australasian Data Mining Workshop*, 2002.

[3] W. Cohen, P. Ravikumar, and S. Feinberg. A comparison of string metrics for matching names and records. In *KDD-2003 Workshop on Data Cleaning and Object Consolidation*, 2003.

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

[5] M. A. Hernandez and S. J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM SIGMOD Conference*, 1995.

[6] T. Huang and S. J. Russell. Object identification in a bayesian context. In *IJCAI-97*, pages 1276–1283, 1997.

[7] T. Joachims. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[8] A. McCallum and B. Wellner. Conditional models of identity uncertainty with application to noun coreference. In *Neural Information Processing Systems (NIPS)*, 2004.

[9] A. E. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of ACM SIGKDD-96*, pages 267–270, 1996.

[10] P. Ravikumar and W. W. Cohen. A hierarchical graphical model for record linkage. In *UAI 2004*, 2004.

[11] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of ACM SIGKDD-02*, Edmonton, Alberta, Canada, 2002.

[12] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of ACM SIGKDD-02*, Edmonton, Alberta, Canada, 2002.