Proceedings of the ICML-2002 Workshop on Text Learning,
pp. 18-27, Sydney, Australia, July 2002

Two Approaches to Handling Noisy Variation in Text Mining

Un Yong Nahm PEBRONIA@CS.UTEXAS.EDU
Mikhail Bilenko MBILENKO@CS.UTEXAS.EDU
Raymond J. Mooney MOONEY@CS.UTEXAS.EDU
Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712-1188 USA

Abstract

Variation and noise in textual database entries can prevent text mining algorithms from discovering important regularities. We present two novel methods to cope with this problem: (1) an adaptive approach to "hardening" noisy databases by identifying duplicate records, and (2) mining "soft" association rules. For identifying approximately duplicate records, we present a domain-independent two-level method for improving duplicate detection accuracy based on machine learning. For mining soft matching rules, we introduce an algorithm that discovers association rules by allowing partial matching of items based on a textual similarity metric such as edit distance or cosine similarity. Experimental results on real and synthetic datasets show that our methods outperform traditional techniques for noisy textual databases.

1. Introduction

Textual entries in database fields often exhibit minor variations that can prevent mining algorithms from discovering important regularities. Variations can arise from typographical errors, misspellings, abbreviations, as well as other sources. Variations are particularly pronounced in data that is automatically extracted from unstructured or semi-structured documents or web pages (Ghani et al., 2000; Nahm & Mooney, 2000). For example, in data on local job offerings that we automatically extracted from newsgroup postings, the Windows operating system is variously referred to as "Microsoft Windows", "MS Windows", "Windows 95/98/ME", etc. Sample duplicate records on Fig.1 illustrate textual variations in a restaurant database formed by merging two guidebooks.

Some previous work has addressed the problem of identifying duplicate records, where it is referred to as record linkage (Winkler, 1999), the merge/purge problem (Hernández & Stolfo, 1995), duplicate detection (Monge & Elkan, 1997), hardening soft databases (Cohen et al., 2000), and reference matching (McCallum et al., 2000). Typically, a fixed textual similarity metric such as edit distance (Gusfield, 1997) or vector-space cosine similarity (Salton, 1989) is used to determine whether two values or records are alike enough to be duplicates. However, the similarity of two strings can depend on the domain and field under consideration. Rather than hand-tuning a distance metric for each field in each domain, we present a method for automatically learning an appropriate string-similarity metric from small corpora of hand-labeled examples. Our system, MARLIN (Multiply Adaptive Record Linkage with INduction), employs a two-level learning approach. First, a set of similarity metrics are trained to appropriately determine the similarity for different database fields. Next, a final predicate for detecting duplicate records is learned from multiple similarity metrics for each of the individual fields using Support Vector Machines (SVM's) (Vapnik, 1995).

With this "de-duping" approach, however, discovered associations are not able to capture all similarities between different items. Since the similarity relation for textual items is non-transitive, sometimes a fixed clustering encounters a dilemma, e.g. when "WinNT" partially matches "NT", but it also matches "Windows". By allowing partial matching of items, we explore the alternative of directly mining "dirty" data by discovering "soft matching" association rules whose antecedents and consequents are evaluated based on sufficient similarity to database entries. Similarity of textual items is measured using standard "bag of words" metrics (Salton, 1989) and edit distance measures (Gusfield, 1997); other standard similarity metrics can be used for numerical and other data types. We generalize the standard APRIORI algorithm for discovering association rules (Agrawal & Srikant, 1994) to allow for soft matching given a similarity metric for each field.

2. Adaptive Duplicate Detection Using Learned Similarity Metrics

Identifying duplicate records and fields in textual databases is beneficial both for text mining tasks such as association rule discovery, and for traditional database tasks such as obtaining counts and accurate query results. Our duplicate de-

Figure 1. Sample duplicate records from the RESTAURANT database

name	address	city	phone	cuisine
fenix	8358 sunset blvd. west	hollywood	213/848-6677	american
fenix at the argyle	8358 sunset blvd.	w. hollywood	213-848-6677	french(new)

tection system, MARLIN, utilizes training pairs of records labeled as duplicates to induce a separate text similarity measure for each field, and to learn a classifier that combines the similarities across fields in a meaningful manner to produce record similarity estimates. In this section, we describe the learning methods used in MARLIN and some experimental results that illustrate the benefits of our approach over non-adaptive duplicate detection methods that use static similarity metrics.

2.1. Learnable String Distance

2.1.1. EDIT DISTANCE

A common measure of textual similarity is *string edit distance*, originally proposed by Levenshtein (Levenshtein, 1966). It is defined as the minimum number of insertions, deletions or substitutions necessary to transform one string into another, and can be computed for two strings of lengths T and V using a dynamic programming algorithm in $O(TV)$ time. Needleman and Wunsch (Needleman & Wunsch, 1970) extended the distance model to allow contiguous sequences of mismatched characters, or gaps, in the alignment of two strings.

Most commonly the gap penalty is calculated using the *affine* model, which assigns a high penalty for starting a gap, and a lower linearly increasing penalty for extending a gap. With this model, distance $S(x^T, y^V)$, between strings x^T of length T and y^V of length V, can be computed using a dynamic programming algorithm that constructs three $T \times V$ matrices, see (Gusfield, 1997) for details.

Because different edit operations have varying significance in different domains, adapting a string similarity metric to a particular domain requires assigning character-specific costs to gap costs and different elementary edit operations (insertion, deletion, substitution). This task that has traditionally been performed manually either using domain knowledge or by trial and error. A more popular approach is to approximate edit distance by adopting fixed costs for edit operations across all characters.

2.1.2. LEARNABLE EDIT DISTANCE WITH AFFINE GAPS

Ristad and Yianilos (1998) developed a generative model for Levenshtein distance along with an Expectation-Maximization algorithm that learns model parameters using a training corpus of matched strings. We propose a similar stochastic model for the edit distance with affine gaps. Each of the three matrices of the original affine gap model corresponds to one of the states of the generative model in Fig.2. A pair of matched strings is generated by this model as a sequence of traversals along the edges accompanied by emissions of characters pairs, which are determined by the state that is reached via each traversal.

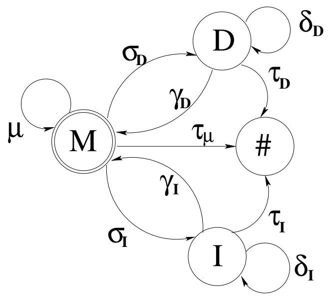

Figure 2. Generative model for string distance with affine gaps

The production starts in state M and terminates when special state $\#$ is reached. Transitions σ_D and σ_I from the matching state M to either the deletion state D or the insertion state I correspond to starting a gap in one of the strings. A gap is ended when edges γ_D and γ_I are traversed back to the matching state. Remaining in state M by taking edge μ corresponds to a sequence of substitutions or exact matches of characters, while remaining in states I and D is analogous to extending a gap in either the first or the second string.

Edit operations emitted in each state correspond to aligned pairs of characters: substitutions $\langle a, b\rangle$ and exact matches $\langle a, a\rangle$ in state M; deletions from the first string $\langle a, \epsilon\rangle$ in state D; and insertions of characters from the second string into the first string $\langle \epsilon, a\rangle$ in state I. Each edit operation e is assigned a probability $p(e)$; probabilities are normalized for each state. Edit operations with higher probabilities produce character pairs that are likely to be aligned in a given domain, such as substitution ("/", "-") for phone numbers, or deletion (".", ϵ) for addresses.

Standard forward and backward algorithms can be used to obtain the probability of generating a pair of strings using this model. Given a corpus of n matched strings corresponding to pairs of duplicates, $C = \{(x^{T_1}, y^{V_1}), \ldots, (x^{T_n}, y^{V_n})\}$, this model can be trained using the Baum-Welch algorithm, which is a variant of the Expectation-Maximization procedure for learning parameters of generative models (Rabiner, 1989). See (Bilenko & Mooney, 2002) for a full description of the model and details of the algorithms. Thus, learnable edit distance with affine gaps provides us a domain-specific textual similarity measure can be adapted to reflect the notion of similarity that is specific to each field of a given database.

2.2. Combining similarity across multiple fields

When the distance between records composed of multiple fields is being calculated, it is necessary to combine similarity estimates for individual fields in a meaningful manner, weighting them according to their contribution to the true similarity between records.

While statistical aspects of combining similarity scores for individual fields have been addressed in previous work on record linkage (Winkler, 1999), availability of labeled duplicates allows a more direct approach that uses a binary classifier (Cohen & Richman, 2001). Given a database that contains records composed of k different fields and a set $D = \{d_1, \ldots, d_m\}$ of distance metrics, we can represent any pair of records by an mk-dimensional vector of "distance features". Each component of the vector represents similarity between two fields of the records calculated using one of the distance metrics. Matched pairs of duplicate records $R = \{(r_{10}, r_{11}), \ldots, (r_{n0}, r_{n1})\}$ can be used to construct a training set of such vectors by assigning them a positive class label. Pairs of records that are not labeled as duplicates form the complementary set of negative examples.

A binary classifier can then be trained using these vectors to discriminate between pairs of records corresponding to duplicates and non-duplicates. MARLIN utilizes Support Vector Machines (SVM's) (Vapnik, 1995), which are appropriate for this task due to their resistance to noise and ability to handle correlated features well. Confidence estimates of belonging to each class are naturally given by a datapoint's distance from the hyperplane that separates classes of duplicates and non-duplicates in high-dimensional space that is constructed by the SVM during training.

2.3. Duplicate detection algorithm

A confidence estimate of belonging to the class of duplicates for a given pair of records can be viewed as an overall measure of similarity between the records comprising the pair. Given a large database, producing all possible pairs of records and computing similarity between them is too expensive since it would require $\frac{n^2-1}{2}$ distance computations. There are two methods which can be used to cut down the number of potential duplicate pairs: the sorted neighborhood method (Hernández & Stolfo, 1995) and the canopies clustering method (McCallum et al., 2000). The former utilizes sorting the database using different fields as keys in multiple passes, sliding a window of fixed size over the sorted database during each pass, and adding all pairs of records that co-occur within the window as potential duplicates. The canopies clustering method utilizes some computationally inexpensive metric d_c, such as Jaccard similarity based on an inverted index, to separate records into overlapping clusters ("canopies") of potential duplicates, and subsequently adds all pairs of records that fall in each cluster as potential duplicates.

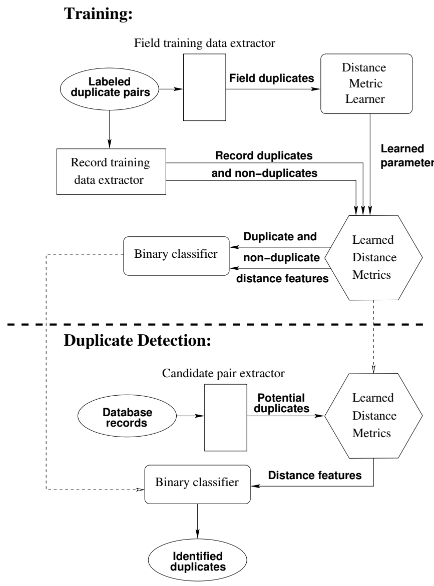

Figure 3. MARLIN overview

An overall view of our system, MARLIN, is presented in Fig.3. In the training phase, the learnable distance metrics are trained for each record field. The training corpus of paired field-level duplicates is obtained by taking pairs of values for each field from the user-provided set of paired duplicate records. Because duplicate records may contain individual fields that are not equivalent, training data can be noisy. This does not pose a serious problem for our approach, since particularly noisy fields that are unhelpful for identifying record-level duplicates will be ignored by the binary classifier as irrelevant distance features.

After individual similarity metrics are learned, they are used to compute distances for each field of duplicate and non-duplicate record pairs to obtain training data for the binary classifier in the form of vectors composed of distance features. Because we are employing a classifier that does not depend on the relative sizes of training data for the two classes, it is sufficient to randomly add n non-duplicate record pairs to the training set.

The duplicate detection phase starts with the generation of potential duplicate pairs using either the sorted neighborhood or canopies method. Next, learned distance metrics are used to calculate distances for each field of each pair of potential duplicate records, thus creating distance feature vectors for the classifier. Confidence estimates for belonging to the class of duplicates are then produced by the binary classifier for each candidate pair, and pairs are sorted by increasing confidence.

The problem of finding a similarity threshold for separating duplicates from non-duplicates arises at this point. A traditional approach to this problem (Winkler, 1999) requires assigning two thresholds: one that separates pairs of records that are high-confidence duplicates, and another for possible duplicates that should be reviewed by a human expert. Since relative costs of labeling a non-duplicate as a duplicate (false positives) and overlooking true duplicates (false negatives) can vary from database to database, there is no "silver bullet" solution to this problem. Availability of labeled data, however, allows us to provide precision-recall estimates for any threshold value and thus offer a way to control the trade-off between false and unidentified duplicates by selecting threshold values that are appropriate for a particular database.

It is highly likely that several identified duplicate pairs will contain the same record. Since the "duplicate of" relation is transitive, it is necessary to compute the transitive closure of equivalent pairs to complete the identification process. Following (Monge & Elkan, 1997), we utilize the union-find data structure to store all database records in this step, which allows updating the transitive closure of identified duplicates incrementally in an efficient manner.

2.4. Experimental Evaluation

2.4.1. Datasets

We have used three different datasets for our experiments. RESTAURANT is a database of 864 restaurant names and addresses containing 112 duplicates assembled by Sheila Tejada from Fodor's and Zagat's guidebooks. The second dataset, CORA, is a collection of 1295 distinct references to 122 Computer Science research papers from the Cora Computer Science research paper search engine[1]. Finally, we used the database generator of Hernández and Stolfo (Hernández & Stolfo, 1995) that randomly corrupts records to introduce duplicates into a mailing list database to create the MAILING dataset of 1200 records corresponding to 400 original entries. Fig.1 contains sample duplicate records from the RESTAURANT database.

2.4.2. Experimental Methodology

All experiments were conducted using 10-fold cross validation. To create the folds, duplicate records were grouped together, and the resulting clusters were randomly assigned to the folds. Because the sizes of our datasets allowed computing distances between all pairs of records, we did not employ the sorted neighborhood or canopies approaches to limit the number of potential duplicates. Either of the approaches, however, could be used for evaluating accuracy of duplicate detection on larger datasets.

After computing distances between all pairs of potential duplicates, the pair of records with the highest similarity was labeled as a duplicate, and the transitive closure of groups of duplicates was updated. Precision, recall and F-measure defined over pairs of duplicates were computed after each iteration, where precision is the fraction of identified duplicate pairs that are correct, recall is the fraction of actual duplicate pairs that were identified, and F-measure is the harmonic mean of precision and recall.

Precision was interpolated at 20 standard recall levels following the traditional procedure in information retrieval (Salton, 1989). Only those portions of the curves that exhibit differences between approaches are shown.

2.4.3. Results and Discussion

Detecting duplicate field values To evaluate the usefulness of adapting character-based distance metrics to a specific domain, we compared learned similarity metrics with their fixed-cost equivalents for the task of identifying equivalent field values. We chose the most meaningful fields from the three datasets for these experiments: CORA `paper title` field, RESTAURANT `name` and `address` fields, and MAILING `street address` and `name` fields (the latter is a concatenation of `first name` and `last name` fields).

We have compared four distance metrics:

- Levenshtein edit distance (Gusfield, 1997), calculated as the minimum number of character deletions, insertions and substitutions of unit cost;
- Learned Levenshtein edit distance based on a generative model and trained using the Expectation-Maximization procedure described in (Ristad & Yianilos, 1998);
- String distance with affine gaps (Gusfield, 1997) using a substitution cost of 3, gap opening cost of 3, and gap extension cost of 1, which are commonly used parameters;
- Learned string distance with affine gaps described in Section 2.1.2

Results for field-level duplicate detection experiments are summarized in Table 1. Each entry in table contains the average of maximum F-measure values over 10 folds. Results for experiments where the difference between the learned and corresponding unlearned metric is significant at the 0.05 level using a 1-tailed t-test are presented in bold font. Fig. 4 contains recall-precision curves for the performance of MARLIN on the CORA `paper title` field.

Performance improvements achieved when learned distance metrics were used instead of fixed-cost distance metrics for detecting field duplicates demonstrate that learnable distance metrics are able to approximate the relative importance of differences between strings for a specific field, as can be seen from higher maximum F-measure values in Table 1. Results of all experiments except for the `address` field of the MAILING database demonstrate that

[1] `http://cora.whizbang.com`

Table 1. Maximum F-measure for detecting duplicate field values

Distance metric	CORA title	RESTAURANT name	RESTAURANT address	MAILING name	MAILING address
Levenshtein	0.870	0.843	0.950	0.867	0.878
Learned Levenshtein	0.902	**0.886**	0.975	**0.899**	0.897
Affine	0.917	0.883	0.870	0.923	0.886
Learned Affine	**0.971**	**0.967**	**0.929**	**0.959**	0.892

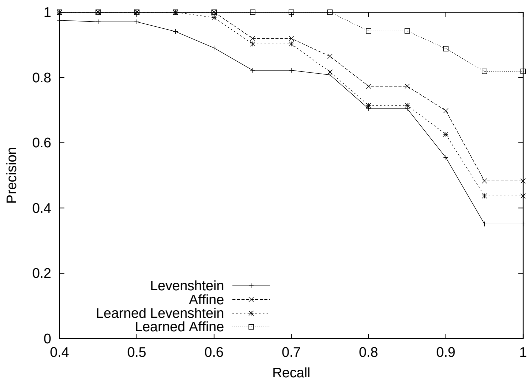

Figure 4. `Title` duplicate field-value detection results for the CORA dataset

taking gaps into account when constructing string alignments results in better estimates of string similarity for the task of detecting approximate duplicate field values. The fact that the results of all metrics were not significantly different on that field can be explained by the fact that a certain fraction of entries was heavily corrupted by substituting PO Box addresses, which are effectively impossible to match against the corresponding street address without using other fields such as `name` and `city`.

Record-level duplicate detection Next, we evaluated the performance of MARLIN for multi-field (record-level) duplicate detection. The SVM implementation from the WEKA toolkit (Witten & Frank, 1999) was used as the binary classifier. We have compared classifier-based similarity estimation to using the sum of distances from different fields as a non-trained record-level similarity measure. Either simple affine gap distance or learned string distance with affine gaps described above were used for computing similarity between values of each record field. Classifier-based experiments are denoted by "SVM", while experiments that used a sum of distances across fields are labeled as "Sum" in Table 2. We also conducted additional experiments using the SVM for record-level classification based on Jaccard similarity as the distance metric for individual fields.

Results for all experiments are summarized in Table 2. Again, results in bold font correspond to those experiments in which differences between using the learned and unlearned string metrics are significant at the 0.05 level using a 1-tailed t-test. All differences between the SVM and Sum approaches are significant at the 0.05 level using a 1-tailed t-test, except for the experiments that use unlearned string distance with affine gaps for the RESTAURANT dataset, and those that use learned string distance with affine gaps for the MAILING dataset.

Table 2. Maximum F-measure for duplicate detection based on multiple fields

Classifier	Metric	CORA	RESTAURANT	MAILING
Sum	Affine	0.561	0.847	0.9431
Sum	Learned Affine	0.564	0.832	**0.991**
SVM	Affine	0.959	0.861	0.992
SVM	Learned Affine	0.958	**0.971**	**0.996**
SVM	Jaccard	0.983	0.971	0.961

From the results on the RESTAURANT and MAILING datasets we can conclude that using adaptive string distance metrics to compute similarity between field values makes a positive contribution when similarities from multiple fields are combined either in a simplistic manner by adding them, or by using them as record-pair attributes for classification.

We also ran trials which combined character-based metrics (static and adaptive string distance with affine gaps) and token-based metrics (Jaccard similarity). These experiments resulted in near-100% precision and recall, without significant differences between static and adaptive field-level metrics. Similar results were obtained when common prefix and common suffix lengths were used as field-level distance metrics along with the character-based metrics used above. This demonstrates that combining character- and token-based distance metrics, such as learned string distance with affine gaps and Jaccard similarity, is clearly an advantage of the two-level approach implemented in MARLIN. Current datasets did not allow us to show the benefits of adaptive metrics over their static prototypes in this scenario, but our initial results suggest that this can be demonstrated on more challenging datasets.

3. Mining Soft Association Rules

In this section, we generalize the standard APRIORI algorithm for discovering association rules (Agrawal & Srikant, 1994) to allow for soft matching based on a given similarity metric for each field. Detailed descriptions for SOFTAPRIORI can be found in (Nahm & Mooney, 2002).

3.1. Background: Mining Association Rules

Given a database $\mathcal{D}$, the problem of mining association rules is to discover all association rules that have support and confidence greater than the user-specified minimum support and confidence. An association rule from a supermarket database, "$beer \Rightarrow pretzels$ $[5\%, 80\%]$" indicates that 5% of customers bought beer and pretzels together and 80% of those who bought beer also bought pretzels. One of the popular algorithms for mining association rules is APRIORI (Agrawal & Srikant, 1994) where the closure property of itemset support was introduced.

3.2. Mining Soft Association Rules

We introduce the problem of mining *soft* association rules from databases and investigate how to utilize an existing association rule mining algorithm to incorporate similarity in discovering associations. We present two implementations using a string edit distance and a cosine similarity as the primary similarity metrics.

3.2.1. SOFT ASSOCIATION RULES

Soft relations are defined as follows. We assume that a function, $similarity(x, y)$, is given for measuring the similarity between two items x and y. The range of the similarity function is the set of real numbers between 0 to 1 inclusive, and $similarity(x, y) = 1$ iff $x = y$.

Definition 1 (is-similar-to) *An item x is similar to an item y ($x \sim y$) iff $similarity(x, y) \geq T$ (where the threshold T is a predefined constant between 0 and 1).*

Definition 2 (is-a-soft-element-of) *An item x is a soft-element of an itemset I ($x \in_{soft} I$) iff there exists an $x' \in I$ such that $x' \sim x$.*

Definition 3 (is-a-soft-subset-of) *An itemset I is a soft-subset of an itemset J ($I \subseteq_{soft} J$) iff for every item in I there is a distinct similar item in J, i.e. for every item $x_i \in I$, $I = \{x_1, ..., x_m\}$, there is an item $y_j \in J$ such that $x_i \sim y_i$ and $y_i \neq y_j$ for all $j \neq i$, $1 \leq j \leq m$.*

The following is a formal statement of the problem of mining soft association rules: Let $I = \{i_1, i_2, ..., i_m\}$ be a set of literals, called items. Let $\mathcal{D}$ be a set of records, where each record R is a set of items such that $R \subseteq I$. A soft association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and X and Y are *soft-disjoint* (Two itemsets I and J are soft-disjoint when no item in I is a soft-element of J.). The problem of mining soft association rules is to find all soft association rules, $X \Rightarrow Y$, such that the *soft-support* and the *soft-confidence* of $X \Rightarrow Y$ are greater than the user-defined minimum values (called *minsup* and *minconf* respectively). Formal definition for soft-support is given below. Soft-confidence is a straightforward generalization of the traditional one.

```
Input: D (set of records).
Output: L_k (frequent k-itemsets).
Function SoftApriori (D)
L_1 := FindFrequentItemsets(D).
k := 2.
while (L_{k-1} ≠ ∅) do
    C_k := GenerateCandidates(L_{k-1})
    forall records r ∈ D do
        forall c ∈ C_k do
            if c ⊆_soft r
            then c.count := c.count + 1.
    L_k := All candidates in C_k with minsups.
    k := k + 1
return ⋃_k L_k.
```

Figure 5. The SOFTAPRIORI algorithm

Definition 4 (soft-support) *The soft-support of an itemset X in a set of records (database) $\mathcal{D}$, denoted as $softsup(X)$, is the number of records, $R \in \mathcal{D}$, such that $X \subseteq_{soft} R$. The soft-support of a rule $X \Rightarrow Y$ in a database $\mathcal{D}$, denoted as $softsup(X \Rightarrow Y)$, is the number of records $R \in \mathcal{D}$ such that $X \cup Y \subseteq_{soft} R$.*

3.2.2. THE SOFTAPRIORI ALGORITHM

In this section, we consider discovering frequent itemsets, or finding all frequent itemsets with higher soft-support than the user-specified minimum. To discover frequent itemsets for soft association rules, we generalize the existing itemset mining algorithm presented in (Agrawal & Srikant, 1994) in a straightforward way. Since the notion of equality in the traditional definition of an association rule is replaced by similarity, we need to compute the soft-support of each item and itemset by Definition 4. Similarity between items is computed once and cached for future references. Fig. 5 gives pseudocode for the SOFTAPRIORI algorithm.

The first step of the algorithm determines the frequent 1-itemsets. The set of frequent 1-itemsets L_1 in SOFTAPRIORI is the set of all 1-itemsets whose soft-support is greater than the user-given minimum support. By Definition 4, the soft-support of each item is calculated by summing the number of occurrences of all similar items. Formally, the soft support of a 1-itemset $\{x\}$ where x is an element of the set of all items I ($x \in I$) is computed as follows:

$$softsup_I(\{x\}) = \sum_{y \in I} similar(x, y) \times support(y)$$

While counting the occurrences of all items, we measure the similarity of every pair of items and construct an $m \times m$ matrix $similar(i, j)$, where m is the total number of items in the database. To determine frequent 1-itemsets, the soft-supports of all items are computed. Intuitively, we construct a cluster of items containing the items similar to each given "central" item, and sum the support of all items in the cluster. Soft items are then treated the same as items in the original APRIORI algorithm.

In a manner similar to the initial construction of frequent items, itemsets are grown by computing the soft-support of candidates and discarding those with low soft-support. For each itemset that is a soft subset of r, or for each set of items that have similar items in r, soft-support of an k-itemset is again computed by the equation in Definition 4, counting the number of soft-matching items, instead of simply counting the number of occurrences of each item.

By treating every pair of items in different fields as non-similar, we are able to lower the number of similarity computations which was originally $O(m^2)$ to $\sum_{k=1}^{N} m_k{}^2$ whereas N is the number of fields, m_k is the number of items in field k, and m is the total number of items. Depending on the particular similarity metric, additional optimizations are possible. For example, items in numeric fields can be sorted and then similar items can be quickly determined by checking neighboring items only. Additional optimizations for string edit distance and cosine similarity are presented in the following sections.

3.2.3. IMPLEMENTATION I: STRING EDIT DISTANCE

We implemented SOFTAPRIORI by modifying a publicly available version of APRIORI (Borgelt, 2000) with the affine model (Gusfield, 1997). To measure similarity of string-valued items, we defined $similarity(x, y)$ as $1 - normalized\_edit\_distance(x, y)$ where normalized edit distance is scaled to always be between 0 and 1 (based on the lengths of the two strings). The similarity of items in numeric fields is defined as the normalized absolute difference.

Given a particular edit distance function, we can reduce the time complexity of determining similar items. Since edit distance counts the number of operations needed to change one string to another, two strings cannot be similar if their lengths are too different. By generalizing this observation, we obtain a test function to determe if two strings can **not** be similar under affine gap cost so that we are able to eliminate edit distance computations for very different strings.

We can reduce the number of comparisons between items even further by using an n-gram index (Kim & Shawe-Taylor, 1994). An n-gram is a substring of length n of a given string. It can be shown that a string x cannot be similar to y for a given threshold if they do not share at least k ($k \geq 0$) n-grams. In our implementation, we used a trigram index to efficiently retrieve a list of candidate similar strings for each string. Each string is indexed under every three-character substring that it contains. To find candidates for similar strings, we use the index to retrieve all strings that share at least a certain number of trigrams.

3.2.4. IMPLEMENTATION II: VECTOR-SPACE MODEL

The implementation presented in Section 3.2.3 focuses on textual data in which items are short strings for which edit distance is a suitable similarity metric. An obvious extension is to replace edit distance with "bag of words" similarity metrics such as vector-space cosine similarity from information retrieval (Salton, 1989). In vector-space model, a text is represented as a vector of real numbers, where each component corresponds to a word and the value is its frequency in the document. The similarity of two documents x and y is the *cosine of the angle* between two vectors $\vec{x}$ and $\vec{y}$ representing x and y respectively. We also used the standard TFIDF (Term Frequency, Inverse Document Frequency) weighting scheme to assign higher weights to distinguished terms.

In terms of computational performance, the major bottleneck is the worst-case quadratic time complexity of measuring the similarity of many pairs of items. Fortunately, there are well-known indexing methods that allow efficient identification of items that are close in cosine similarity (Salton, 1989; Cohen, 1998). In our implementation, we used an inverted index to retrieve similar items efficiently. Performance gains obtained by this optimization are shown in Section 3.3.4.

3.3. Experimental Evaluation

Our focus on mining soft-matching rules was motivated by text-mining research on discovering patterns in data automatically extracted from natural-language documents and web pages (Nahm & Mooney, 2000; Nahm & Mooney, 2001). Therefore, we evaluate SOFTAPRIORI on "dirty" databases extracted from text and compare prediction accuracies (measured on independent test data) of soft and hard association rules mined from the same training data.

3.3.1. DATASETS AND SAMPLE RULES

For the first dataset, 300 computer-science resume postings to the newsgroup `misc.jobs.resumes` were collected and information on programming languages, platforms, applications, areas, hardwares, software engineering skills, job title, major, degree, years of experience, and post date were identified by human tagging to construct a textual database of job skills. Second, 3,000 science fiction (SF) book descriptions are automatically extracted from the `Amazon` online bookstore. The information extractor for Amazon was developed manually and is highly accurate. 12 fields (title, author, reviews, synopses, type, publisher, date, subjects, related books and authors, price, and rating) are identified and 10 of them except synopses and reviews are used for the Implemention I (edit distance) while only 5 fields (title, author, reviews, synopses, subjects) are used for Implementation II (vector-space model).

Examples of interesting soft association rules mined from the four sets of data are shown in Fig. 6 and Fig. 7. Items similar to a given item are shown in parentheses, and values for softsup and softconf are shown in brackets. With the same values of confidence and support, SOFTAPRIORI

1. unix ∈ platform ⇒ html (d/html, dhtml, shtml) ∈ programming-language [22.7%, 54.4%]

2. unix ∈ programming-language ⇒ visual basic (visual basic 5.0, visual basic 6.0, visual basic 4.0, ms visual basic, visual basic 4, visual basic 5/6) ∈ programming-language [13.0%, 31.2%]

3. netscape (netscape 4.7, netscape 4.x, netscape 6, netscape ldap) ∈ application ⇒ tcp/ip (tcp ip, tcpip) ∈ application [3.3%, 34.5%]

Figure 6. Sample soft association rules from USENET resume postings ((Implementation I, T = 0.7)

1. bob ruddick ∈ author ⇒ blanche l. sims (blanche sims) ∈ author [0.3%, 30.0%]

2. david gerrold (d. gerrold) ∈ author ⇒ american science fiction and fantasy science fiction science fiction ; fantasy (american science fiction and fantasy science fiction, general science fiction science fiction ; fantasy, english science fiction and fantasy science fiction science fictinon ; fantasy, etc..) ∈ subjects [0.2%, 50.0%]

3. isaac asimov (isaac amimov, isaac asimoy) ∈ author **and** robot dreams ∈ related books ⇒ robot visions ∈ related books [0.1%, 100.0%]

Figure 7. Sample soft association rules from SF book descriptions (Implementation I, T = 0.7)

discovers more general rules including frequent clusters of similar items that would be overlooked by the traditional algorithm because of the low support values for individual items. Other examples show that soft association rules are able to capture patterns based on groups of similar items such as the different versions of Visual Basic, typos in the author slot, etc..

We presented items in soft association rules by specifying the item itself followed by its corresponding similar items in parentheses. For other similarity metrics, alternative approaches to representing "softness," such as ranges on numeric values, may be appropriate. In Implementation II, items are represented by the bag intersection of its similar items in the cluster as in (Nahm & Mooney, 2001). Sample rules from this implementation are shown in Fig. 8. Numbers in parentheses stand for the number of occurrences for each word, or bag counters.

3.3.2. Experimental Methodology

We measured the ability of both hard and soft association rules mined from the same training data with the same minimum confidence and support parameters to make accurate predictions on the same *disjoint* set of test data. To obtain statistically reliable estimates of accuracy, we employed ten-fold cross-validation which averages performance over

1. {mike(1), resnick(1)} ∈ author **and** ⇒ {american(1), fantasy(1), fiction(4), science(3)} ∈ subject [0.2%, 35.0%]

2. {gery(1), greer(1)} ∈ author **and** {accidentally(1), action(1), aliens(1), book(2), boys(1), carried(1), copyright(1), distinct(1), escape(1), horn(2), humorous(1), personalities(1), point(1), reserved(1), rights(1), slight(1), space(1), story(1), strong(1), trouble(1), worth(1)} ∈ review ⇒ {spaceship(1)} ∈ title [0.1%, 100.0%]

3. {asimov(1), janet(1)} ∈ author **and** {fiction(2), robots(1), science(1)} ∈ subject ⇒ {asimov(1), isaac(1)} ∈ author [0.1%, 100.0%]

Figure 8. Sample soft association rules from SF book descriptions (Implementation II, T = 0.7)

Input: $\mathcal{D}_{test}$ (test database), $Rules$ (association rule set)
Output: ($Precision$, $Recall$)
Function ComputeAccuracy ($\mathcal{D}_{test}$, $Rules$)
$fired := matched := item := predicted := 0$.
for each record $R \in \mathcal{D}_{test}$ **do**
  for each r ($A \Rightarrow c$) $\in Rules$ **do**
    if ((r is hard and $A \subseteq R$) **or**
      (r is soft and $A \subseteq_{soft} R$))
    then if r is hard **then** $A' := A$.
        else $A' := X$ s.t. $X \subseteq R$ and $X \sim A$.
      $fired := fired + 1$.
      if $c \in_{soft} R - A'$
      then $matched := matched + 1$.
  for each $c' \in R$ **do**
    $item := item + 1$.
    if there exists a r ($A \Rightarrow c$) $\in Rules$ s.t.
      $c \sim c'$ **and** ((r is hard and $A \subseteq R - \{c'\}$) **or**
        (r is soft and $A \subseteq_{soft} R - \{c'\}$))
    then $predicted := predicted + 1$.
return ($matched/fired, predicted/item$).

Figure 9. Method for evaluating precision/recall

10 trials of training on 90% of the data and testing on 10% (Mitchell, 1997).

To determine the accuracy of a set of association rules, we measured precision and recall with respect to predicting the presence of items in a record from other items in that record. *Precision* is the percentage of predicted items that are actually present and *recall* is the percentage of actual items that are correctly predicted. We also report *F-measure* which is the harmonic mean of recall and precision. A prediction is judged to be correct iff there is an item in the record that is at least similar to the predicted item (i.e. $similarity(x, y) \geq T$). The pseudocode for the evaluation method is presented in Fig. 9.

3.3.3. Results and Discussion

The experimental results obtained for the resume postings using Implementation I are summarized in Table 3. We used a similarity threshold of 0.7 for every field. Differences for hard and soft rules were evaluated by a two-tailed,

Minconf		Minsup (%)		
(%)	Rule	5	10	15
50	Soft	90.86/3.17	86.95/3.14	84.55/3.13
	Hard	62.19/3.01	60.41/2.76	60.32/2.31
60	Soft	90.79/3.18	87.71/3.13	85.64/3.13
	Hard	66.64/2.89	64.47/2.50	62.16/2.09
70	Soft	91.34/3.18	89.45/3.13	85.76/3.08
	Hard	71.51/2.61	69.75/1.92	74.50/1.43
80	Soft	92.14/3.15	88.37/3.11	84.13/2.82
	Hard	78.84/2.25	79.05/1.46	80.60/0.69

Table 3. Accuracies of soft vs. hard rules on USENET resume postings (precision/recall)

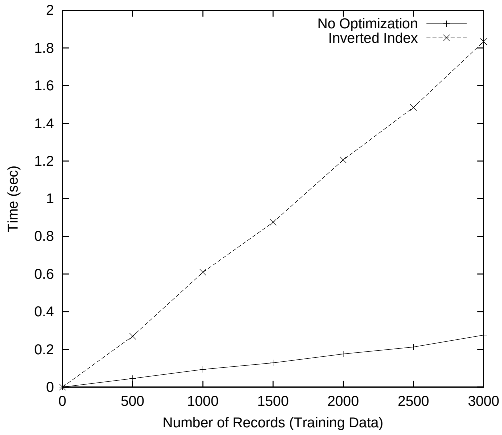

Figure 10. Running time for similarity computations (Implementation II)

paired t-test to determine if they were statistically significant ($p < 0.05$). Overall, the results clearly show that soft rules are generally better than hard rules at discovering reliable regularities in "dirty" data.

3.3.4. Performance Results

Finally, we present results showing the efficiency gained by using the optimization methods presented previously for quickly finding similar string-valued items. The 3,000 SF book descriptions were used in this experiment. Fig. 10 shows the CPU time needed for each item in the similarity computation step. The "Inverted Index" method employs the inverted index to efficiently retrieve documents with shared terms. Performance gains for the edit distance implementation can be achieved similarly by using the trigram index (Nahm & Mooney, 2002). Both results demonstrate that with good heuristics and an efficient indexing method, our approach is scalable to larger datasets by reducing the number of explicit similarity comparisons between pairs of items.

4. Related Work

In previous work, the problem of identifying duplicate records in databases was studied as record linkage (Winkler, 1999), the merge/purge problem (Hernández & Stolfo, 1995), duplicate detection (Monge & Elkan, 1997), hardening soft databases (Cohen et al., 2000), and reference matching (McCallum et al., 2000). In all of these approaches fixed-cost similarity metrics were used to compare database records. The only previous work on adaptive duplicate detection that we know of is the approach described in (Cohen & Richman, 2001), which learns how to combine multiple similarity metrics to identify duplicates, but does not adaptively tune the underlying field-similarity metrics themselves.

Association rule mining has been applied directly to textual data (Feldman & Hirsh, 1996; Ghani et al., 2000); however, the heterogeneity of items in textual databases has not been adequately addressed. Compared to an inductive method for learning soft-matching prediction rules (Nahm & Mooney, 2001), SoftApriori finds *all* association rules with a given soft-support and soft-confidence, and therefore typically discovers a larger set of regularities.

5. Future Work

Extending the metric learning approach to token-based distance metrics, such as Jaccard similarity or vector-space cosine distance, is a promising avenue for research. Previous work on semi-supervised clustering (Cohn et al., 2000) has shown the usefulness of a similar approach: learning weights of individual words when calculating distance between documents using Kullback-Leibler divergence.

Another area for future work lies in generalizing edit distance to include macro-operators for inserting and deleting common substrings, e.g. deleting "Street" in address fields. The string distance model with gaps would be particularly useful for this task, since it would allow discovering useful deletion sequences by counting the frequencies of common gaps.

One possible extension to SoftApriori is to incorporate semantic information in the similarity metric using either lexical knowledge-bases such as WordNet (Fellbaum, 1998) or utilizing statistical measures of semantic similarity in LSI (Deerwester et al., 1990). The limitation of the current definitions for soft-support and soft-confidence is that they do not reflect the different *original support* values of individual items nor different degrees of similarities between items. One solution to this problem is to adapt the approach to use a real-valued similarity measure rather than a binary one.

Currently, the similarity function as well as the threshold values for determining similarity for SoftApriori are pre-determined and fixed throughout the mining phase. Since the similarity of two textual items can vary depending on the specific domain, automatic learning of these functions should be explored. We plan to explore replacing the fixed similarity function currently used in SoftApri-

ORI by learned similarity metrics generated by MARLIN.

6. Conclusion

Data mining methods generally suffer from noisy databases with non-standardized variations especially for text-valued fields. In this paper, we presented two different methods to deal with this problem. First, an adaptive approach called MARLIN has been proposed that learns to identify duplicate records in a specific domain. Our approach uses learning at two levels: training similarity metrics for field-level duplicates and combining multiple similarity metrics for each field to learn a record-level function. Second, the SOFTAPRIORI algorithm to discover "soft matching" rules was developed and evaluated using two different similarity metrics. Experimental results in several domains demonstrate that MARLIN detects duplicates more accurately than competing static approaches, while SOFTAPRIORI allows the discovery of interesting rules that more accurately capture regularities not discovered by traditional methods.

Acknowledgements

We would like to thank William Cohen, Nick Kushmerick, Sheila Tejada and Mauricio Hernández for providing us the duplicate detection datasets. This research was supported by the National Science Foundation under grant IIS-0117308.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *Proceedings of the 20th International Conference on Very Large Databases (VLDB-94)* (pp. 487–499). Santiago, Chile.

Bilenko, M., & Mooney, R. J. (2002). Learning to combine trained distance metrics for duplicate detection in databases. Submitted to CIKM-2002.

Borgelt, C. (2000). Apriori version 2.6. http://fuzzy.cs.UniMagdeburg.de/˜borgelt/.

Cohen, W., & Richman, J. (2001). Learning to match and cluster entity names. *ACM SIGIR-2001 Workshop on Mathematical/Formal Methods in Information Retrieval*. New Orleans, LA.

Cohen, W. W. (1998). Providing database-like access to the web using queries based on textual similarity. *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD-98)* (pp. 558–560). Seattle, WA.

Cohen, W. W., Kautz, H., & McAllester, D. (2000). Hardening soft information sources. *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000)* (pp. 255–259). Boston, MA.

Cohn, D., Caruana, R., & McCallum, A. (2000). Semi-supervised clustering with user feedback. Unpublished manuscript. Available at `http://www-2.cs.cmu.edu/˜mccallum/`.

Deerwester, S. C., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*, 391–407.

Feldman, R., & Hirsh, H. (1996). Mining associations in text in the presence of background knowledge. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)* (pp. 343–346). Portland, OR.

Fellbaum, C. D. (1998). *WordNet: An electronic lexical database*. Cambridge, MA: MIT Press.

Ghani, R., Jones, R., Mladenić, D., Nigam, K., & Slattery, S. (2000). Data mining on symbolic knowledge extracted from the web. *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining* (pp. 29–36). Boston, MA.

Gusfield, D. (1997). *Algorithms on strings, trees and sequences*. New York: Cambridge University Press.

Hernández, M. A., & Stolfo, S. J. (1995). The merge/purge problem for large databases. *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD-95)* (pp. 127–138). San Jose, CA.

Kim, J. Y., & Shawe-Taylor, J. (1994). Fast string matching using an n-gram algorithm. *Software - Practice and Experience*, *24*, 79–83.

Levenshtein, V. I. (1966). Binary codes capable of correcting insertions and reversals. *Soviet Physics Doklady*, *10*, 707–710.

McCallum, A. K., Nigam, K., & Ungar, L. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000)* (pp. 169–178). Boston, MA.

Mitchell, T. (1997). *Machine learning*. New York, NY: McGraw-Hill.

Monge, A. E., & Elkan, C. P. (1997). An efficient domain-independent algorithm for detecting approximately duplicate database records. *Proceedings of the SIGMOD 1997 Workshop on Research Issues on Data Mining and Knowledge Discovery* (pp. 23–29). Tuscon, AZ.

Nahm, U. Y., & Mooney, R. J. (2000). Using information extraction to aid the discovery of prediction rules from texts. *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining* (pp. 51–58). Boston, MA.

Nahm, U. Y., & Mooney, R. J. (2001). Mining soft-matching rules from textual data. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)* (pp. 979–984). Seattle, WA.

Nahm, U. Y., & Mooney, R. J. (2002). Mining soft-matching association rules. Submitted to CIKM-2002.

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, *48*, 443–453.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*, 257–286.

Ristad, E. S., & Yianilos, P. N. (1998). Learning string edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*.

Salton, G. (1989). *Automatic text processing: The transformation, analysis and retrieval of information by computer*. Addison-Wesley.

Vapnik, V. N. (1995). *The nature of statistical learning theory*. Berlin: Springer-Verlag.

Winkler, W. E. (1999). *The state of record linkage and current research problems* (Technical Report). Statistical Research Division, U.S. Bureau of the Census, Wachington, DC.

Witten, I. H., & Frank, E. (1999). *Data mining: Practical machine learning tools and techniques with java implementations*. San Francisco: Morgan Kaufmann.