IntelliClean : A Knowledge-Based Intelligent Data Cleaner

Mong Li Lee, Tok Wang Ling and Wai Lup Low
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
{leeml,lingtw,lowwailu}@comp.nus.edu.sg

ABSTRACT
Existing data cleaning methods work on the basis of computing the degree of similarity between nearby records in a sorted database. High recall is achieved by accepting records with low degrees of similarity as duplicates, at the cost of lower precision. High precision is achieved analogously at the cost of lower recall. This is the *recall-precision dilemma*. In this paper, we propose a generic knowledge-based framework for effective data cleaning that implements existing cleaning strategies and more. We develop a new method to compute transitive closure under uncertainty which handles the merging of groups of inexact duplicate records. Experimental results show that this framework can identify duplicates and anomalies with high recall and precision.

1. INTRODUCTION
The amount of data organizations are handling today is increasing at an explosive rate. The task of keeping the data in these large databases consistent and correct is insurmountable. There are many causes to dirty data: misuse of abbreviations, data entry mistakes, control information hiding, missing fields, spelling, outdated codes etc. Due to the 'garbage in, garbage out' principle, dirty data will distort information obtained from it. Clean data is critical for many industries over a wide variety of applications [5].

Research has shown the importance of domain knowledge in data cleaning. However, there has been little work on knowledge management issues for data cleaning. What type of knowledge is suitable? How can we represent the domain knowledge in a form that we can use for data cleaning? How do we manage the knowledge? Data cleaners using existing methods also face the *recall-precision dilemma*. Higher recall is achieved by relaxing the criteria for records to be considered duplicates. This leads to lower precision as more records which are not duplicates are classified as such. Analogously, we can have higher precision, but at the cost of lower recall. This paper addresses the main shortcomings of existing methods.

ACM SIGKDD'2000 Boston, MA USA

The rest of the paper is organized as follows. Section 2 gives a conceptual model and benchmarking metrics for data cleaning. Section 3 surveys related works. Section 4 describes our proposed knowledge-based framework. Section 5 gives the performance results and we conclude in Section 6.

2. PRELIMINARIES
Figure 1 shows a high level conceptual data cleaning model. We start with a "dirty" dataset with a variety of errors and anomalies. Cleaning strategies are applied to the dataset with the objective of obtaining consistent and correct data as the output. The effectiveness of the cleaning strategies will thus be the degree by which data quality is improved through cleaning. We define two metrics that benchmark the effectiveness of data cleaning strategies :

1. **Recall.** This is also known as *percentage hits*. It is defined as the percentage of duplicate records being correctly identified. Suppose we have 7 records $A_1, A_2, A_3, B_1, B_2, B_3, C_1$, with $\{A_1, A_2, A_3\}$ and $\{B_1, B_2, B_3\}$ being different representations of records A and B respectively. A cleaning process which identifies $\{A_1,A_2,C_1\}$ and $\{B_1,B_2\}$ as duplicates would have a recall of $\frac{4}{6} \times 100\% = 66.7\%$.

2. **False-Positive Error.** This is the antithesis of the precision measure and sometimes referred to as *false merges*. It is defined as the percentage of records wrongly identified as duplicates, i.e.

   $$\frac{\text{no. of wrongly identified duplicates}}{\text{total no. of identified duplicates}} \times 100\%$$

   In our example, the process incorrectly identified C_1 as a duplicate record and will have a false-positive error of $\frac{1}{5} \times 100\% = 20\%$.

3. RELATED WORKS
Pre-processing dirty data prior to the data cleaning process leads to more consistent data and better de-duplication. [6] propose that record equivalence can be determined by viewing their similarity at three levels: token, field and record levels. External source files are used to remove typographical errors and inconsistent use of abbreviations.

The most reliable way to detect inexact duplicates is to compare every record with every other record. The *Sorted*

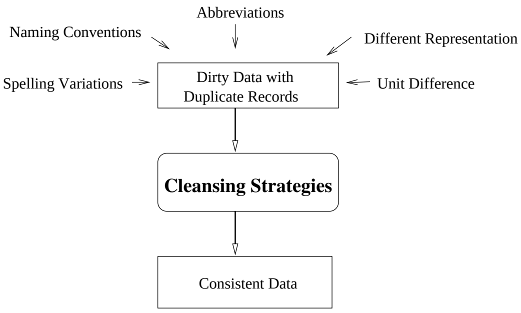

Figure 1: An Overall Conceptual Model

Neighbourhood Method (SNM) [4] reduces this $O(N^2)$ complexity by sorting the database on an application-specific key and making pairwise comparisons of nearby records by sliding a window of fixed size over the sorted database. If the window size is w, then every new record entering the window is matched with the previous $w-1$ records. The first record then moves out of the window. This method requires only wN comparisons, but its effectiveness depends heavily on the ability of the chosen key to bring the inexact duplicates close together. The *Duplicate Elimination SNM (DE-SNM)* [4] improves on the SNM by processing records with exact duplicate keys first. But the drawback of the SNM still persists. The clustering method avoids sorting the database by partitioning the database into independent clusters of approximately duplicate records [4]. However, this will result in many singleton clusters as the proportion of duplicates in a database is typically small.

Multi-pass algorithms assumes that no single key is unique enough to bring all inexact duplicates together and employs the SNM cycle several times, each time using a different key to sort the database to reduce the chances of missed duplicates. These algorithms also compute the transitive closure over the identified duplicates [4, 8], that is, if A is equivalent to B, and B is equivalent to C, then A is also equivalent to C under the assumption of transitivity. In this way, A and C can be detected as duplicates without being in the same window during any of the SNM runs. While this can increase the recall, it is also likely to lower the precision.

The Incremental Merge/Purge Procedure [9] use "representative records" to avoid re-processing data when increments of data need to be merged with previously processed data. The main difficulty with this method is the choice of representative records which characterize the database. [8] gives a domain-independent technique to detect approximate duplicate records which is applicable to textual databases.

4. A KNOWLEDGE-BASED FRAMEWORK

Although domain knowledge has been identified as one of the main ingredients for successful data cleaning [7], the issue of knowledge representation to support data cleaning strategies has not been well dealt with. In this section, we present a knowledge-based framework for intelligent data cleaning. This framework provides a systematic approach for representation standardization, duplicate elimination, anomaly detection and removal in dirty databases. The framework (see Figure 2) consists of three stages:

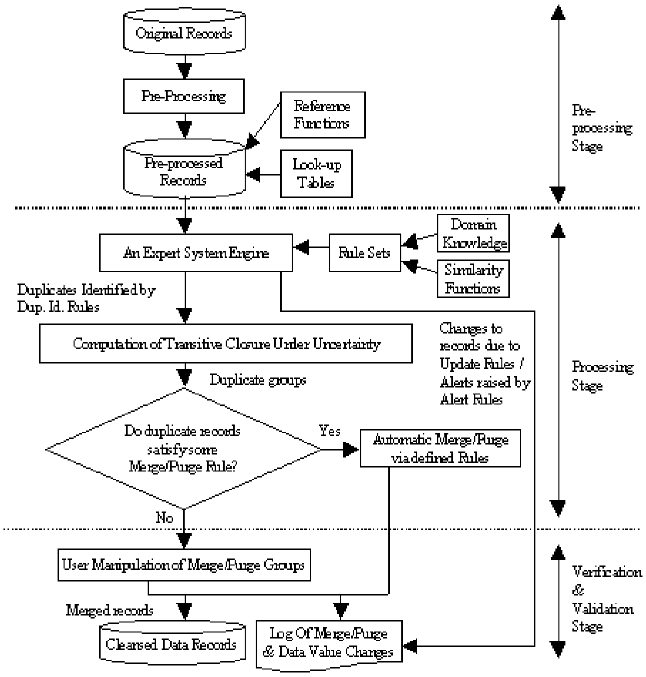

Figure 2: A Knowledge-Based Cleaning Framework

1. Pre-processing Stage.

   Data records are first conditioned and scrubbed of any anomalies we can detect at this stage. Data type checks and format standardization can be performed (e.g. 01/1/2000, 1st January 2000 can be standardized to one format). Inconsistent abbreviations used in the data can be resolved at this stage (e.g. Occurrences of '1' and 'A' in the sex field will be replaced by 'Male', and occurrences of '2' and 'B' will be replaced by 'Female'). Such record scrubbing can be done via reference functions and look-up tables. For example, a function can take in a date and output it in a specified format for the purpose of standardizing date representations. A two-column look-up table might contain an abbreviation and its standardized equivalent. The output of this stage will be a set of conditioned records which will be input to the processing stage.

2. Processing Stage.

   Conditioned records are next fed into an expert system engine together with a set of rules. Each rule will fall into one of the following categories :

   - Duplicate Identification Rules. These rules specify the conditions for two records to be classified as duplicates.
   - Merge/Purge Rules. These rules specify how duplicate records are handled. A simple rule might specify that only the record with the least number of empty fields is to be kept in a group of duplicate records, and the rest deleted. More complex actions can be specified in a similar fashion.
   - Update Rules. These rules specify the way data is to be updated in a particular situation. For example, if the quantity field of a book order record

is empty and the buyer has no other purchases, a rule specifies that the quantity field should be updated with 1. If the buyer has other purchases, the quantity should be the minimum of all purchases belonging to the buyer.

- Alert Rules. A user might want an alert to be raised when certain events occur. For example, data that shows a terminated employee is still receiving payments might be correct in the case of severance payments or might indicate erroneous data. These rules can also be used to alert users of constraint violations, such as functional dependencies and other integrity constraint violations.

The rules are fired in an opportunistic manner when pre-processed records are fed into the expert system engine, implemented using the Java Expert System Shell (JESS) [2]. The pattern matching and rule activation mechanisms of JESS make use of the Rete algorithm [1]. To avoid pairwise comparison for each and every record in the database (which may be infeasible in large databases), we employ the SNM. Hence, the rules will only act on one window of records in the expert system engine at any point in time. After the duplicate record groups are identified, we consider using a new method to compute the transitive closure under uncertainty for these groups to increase the recall. The merge/purge rules will act on the duplicate record groups that satisfy their conditions.

3. Validation and Verification Stage.

   Human intervention is required to manipulate the duplicate record groups for which merge/purge rules are not defined. The log report provides an audit trail for all actions and the reasons for actions made during the pre-processing and processing stages. This log can be inspected for consistency and accuracy and if necessary, wrong merges and incorrect updates can be undone. It can also be used for validating the rule base. If a rule consistently classifies the wrong records as duplicates, or updates values incorrectly, then it should probably be taken out or have its parameters changed.

4.1 The Rete Algorithm

The Rete Algorithm [1] is an efficient method for comparing a large collection of patterns to a large collection of objects. A basic production system checks each if-then statement to see which ones should be executed based on the facts in the database, looping back to the first rule when it has finished. The computational complexity is of the order $O(RF^P)$, where R is the number of rules, P is the average number of patterns in the condition part of the rules, and F is the number of facts in the knowledge base. This escalates dramatically as the number of patterns per rule increases. The Rete Algorithm avoids unnecessary recomputation by remembering what has already been matched from cycle to cycle and then computing only the changes necessary for the newly added or newly removed facts [3]. As a result, the computational complexity per iteration drops to $O(RFP)$, or linear in the size of the fact base [2]. Saving the state of the system may consume considerable memory, but this trade-off for speed is often worthwhile.

In the expert system engine of the proposed framework, data records are the facts. Rules are activated when records match the "patterns" specified in the rules. The actions in the consequent part of the rule will be executed.

4.2 Rule Representation and Effectiveness

Rules, which form the knowledge-base of the framework, are written in the Java Expert System Shell (JESS) [2] language. A rule will generally be of the form:

```
if <condition> then <action>
```

The action part of the rule will be activated or *fired* when the conditions are satisfied. We note here that complex predicates and external function references may be contained in both the condition and action parts of the rule. The rules are derived naturally from the business domain. The business analyst with subject matter knowledge is able to fully understand the governing business logic and can develop the appropriate conditions and actions.

The complexity of the rule language is an important issue in such applications. Although the declarative style of the JESS language makes it intuitive, some knowledge of the working of the Rete Algorithm is required for optimizing rule efficiency. Note that rules can be re-used without modification by different applications if they possess similar business rules. This makes domain-specific rule packages feasible. Given the structure of the rule language, simple rules may be generated automatically when supplied with necessary parameters though rule optimization for more complex rules might require hand-coding.

Well-developed rules are effective in identifying true duplicates but are strict enough to keep out similar records which are not duplicates. Higher recall can then be achieved with more rules. Hence, an increase in the number of well-developed rules can achieve higher recall without sacrificing precision, thereby resolving the recall-precision dilemma.

4.3 Loss of Precision from Computation of Transitive Closure

The rationale for computing the transitive closure after running multi-pass data cleaning algorithms has been discussed in Section 3. However, we note that this procedure can raise the false-positive error as incorrect pairs are merged due to the fact that we are dealing with *inexact equivalence*. Suppose we have two groups of very similar "duplicate" words: (FATHER,MATHER) and (MATHER,MOTHER). The transitive closure step will then merge the two groups into a single group of (FATHER,MATHER,MOTHER), which is obviously wrong since "FATHER" and "MOTHER" are two different words. This problem will magnify as more groups are merged into a single group due to different common records linking various pairs. In this case, the first pair will probably be very different from the last pair.

We can reduce the number of wrongly merged duplicate groups by modifying the structure of Duplicate Identification Rules to the form:

```
if <condition> then
<records are duplicates with certainty factor cf>
```

where $0 < cf \leq 1$. cf represents our confidence in the rule's effectiveness in identifying true duplicates. We can assign high certainty factors to rules we are confident of being able to identify true duplicates. During the computation of the transitive closure, we compare the product of the certainty factors of the merge groups against a user-defined threshold. This threshold represents how confident we want the merges to be. Any merges that will result in a certainty factor less than the threshold will not be carried out. This is because as more groups are merged during the transitive closure step, we are less confident that all the records in the resultant group are representations of the same real-world entity. We present 2 examples here for clarity. CF and TH represent the certainty factor and threshold respectively.

Example 1. Merge Records (A B) with CF=0.8, TH=0.7. Records A and B will be merged as $CF > TH$.

Example 2. Merge Records (A B) with CF=0.9, (B C) with CF=0.85, (C D) with CF=0.8, TH=0.5.
The groups (A B) and (B C) will be considered first, as these groups have higher CFs. They will be merged to form (A B C) with $CF = 0.765$ (since $0.9 \times 0.85 = 0.765$). Then, this group will be merged with (C D) to form (A B C D) with CF=0.612 (since $0.765 \times 0.8 = 0.612$), and the CF is still greater than TH. However, if $TH = 0.7$, (A B C) and (C D) will remain separate, as the resultant CF of the merged group (0.612) will be less than TH.

5. PERFORMANCE STUDY

Based on the ideas and techniques presented in this paper, we implemented the IntelliClean system using Java 1.2 on a Pentium 200 MMX system with 64 MB RAM running Windows NT 4.0. Calls to the Oracle 8 database system are made using JDBC (Java Database Connectivity). The database server is a SUN Enterprise 450 server located in a remote machine connected via a local area network.

We tested IntelliClean using two real world datasets : a Company dataset and a Patient dataset. The former requires complex matching logic and data manipulation, while the latter is a much larger dataset containing many errors.

5.1 Cleaning the Company dataset

The company dataset has 856 records. Each record has 7 fields: Company Code, Company Name, First Address, Second Address, Currency Used, Telephone Number, and Fax Number. Human inspection reveals 60 duplicate records. Problems encountered include large number of empty fields, incomplete data, typographical errors, different representations for names and addresses, misuse of data fields and very similar representations for different entities.

1. Pre-processing. The two address fields are joined into a single field and abbreviations are standardized using a look-up table.

2. Processing. Runs of SNM are conducted with varying window sizes and number of duplicate identification rules. The system detected 80% of the duplicates with 7 duplicate identification rules and kept the false-positive error at less than 8%. Figure 3 shows the effect of the number of rules on recall, precision and runtime.

3. Validation and Verification. The results are verified and manually inspected using the log file generated.

The results show that recall increases with the number of rules, and more complex rules identified more true duplicates. The runtime of the system depends on the complexity of the rules rather than the number of rules. We postulate that an increase in the number of effective rules given additional business knowledge governing this dataset can lead to a further rise in recall while maintaining the false-positive error at low levels.

The window size does not have much effect on recall, unless the window size is comparable to the size of the dataset. This is because SNM's effectiveness depends on inexact duplicates being close to one another after sorting on a chosen key, and that their proximity depends only on the *first few critical characters* of the key. Consider a simple case of a n-character key string and that two records will never be in the same window during the SNM window scan if any of the first m characters are "far apart". If there is a single error in one of the keys, and the error is equally likely to be at any character of the string, then the probability of the character being one of the first few critical characters is m/n. In the case of a 100-character key and the critical characters are the first 3, the probability of two duplicates not being in a same window due to an error in the critical characters is 0.03. Hence, even if erroneous data is in the key, the records are still likely to appear in a same window during the SNM scanning process. In our example above, the probability that the 2 duplicate records being detected is 97%. This estimate will be even higher if we use multiple passes of the SNM with each pass employing a different key. Figure 4 shows the results of our experiments when we vary the window size and number of passes. Note that recall does not increase much as window size increases from 5 to 30.

5.2 Cleaning the Patient dataset

The patient dataset contains 22122 records. This database has 60 fields, including the NRIC Number [1], Sex, Date of Birth, Date of Screening, the Clinic Number and various fields containing codes for the results of a diabetes screen test.

Three rules were used for this dataset : 2 duplicate identification rules and 1 merge/purge rule:

1. Duplicate Id. Rule 1. Certainty Factor 1.
All 60 fields of the record match.

2. Duplicate Id. Rule 2. Certainty Factor 0.9.
Matching Date of Screening and matching NRIC Number after removing all non-numeric characters.

3. Merge/Purge Rule. For all duplicate record groups with certainty factor 1, keep one record from the group and delete the rest.

The Singapore National Registration Identification Card (NRIC) Number is the local equivalent of the Social Security Number of the United States.

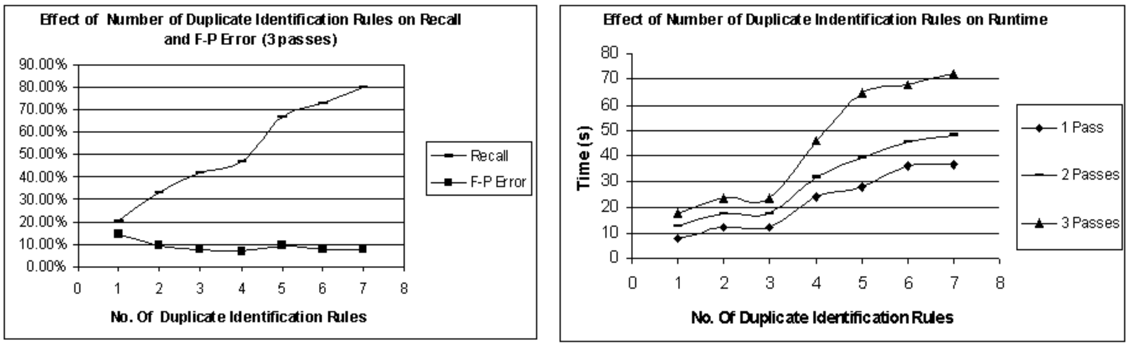

Figure 3: Recall, Precision and Runtime of Data Cleaning System on Company dataset (Window Size 10)

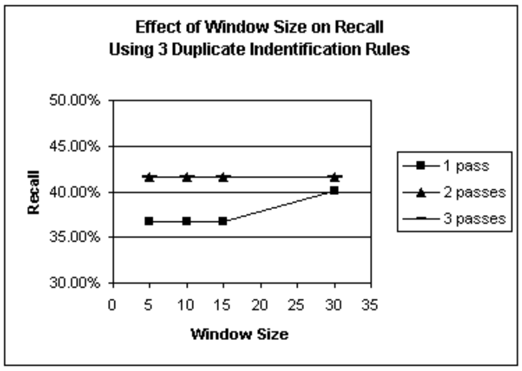

Figure 4: The Effect of Window Size on Recall

Dup. Id. Rule	Match Criteria	Runtime(s)	Duplicates Found
1	All 60 fields match	370.1	8 (4 pairs)
2	Matching Date of Screening and NRIC	464.7	34 (17 pairs)
1 + 2	Criteria of Rules 1 OR 2	503.2	34 (17 pairs)

Table 1: Results of Processing the Patient dataset

A total of 129 alerts, including check digit errors, format and domain value violations, were raised during the pre-processing stage which took 118.5 seconds. The results of the processing stage are shown in Table 1. Manual inspection of the database revealed that we achieved 100% accuracy and precision for this experiment. Although the records identified by Rule 1 form a subset of those identified by Rule 2, we still include this rule as it identifies *exact duplicates*, which the Merge/Purge Rule can handle automatically. The 8 records (4 pairs) that were exact duplicates were automatically processed in this manner. The rest were marked for manual processing.

6. CONCLUSION

In this paper, we have presented a generic knowledge-based framework that can effectively perform data cleaning on any textual database. This framework provides for efficient representation and easy management of knowledge for data cleaning in an expert system. The recall-precision dilemma can also be resolved by specifying effective rules. We have also outlined how the introduction of uncertainty into the computation of transitive closure can increase precision. Consisting of three stages, this framework can support a wide variety of cleaning strategies, of which existing methods form a subset. We are currently extending this framework to de-duplicate results returned by web search engines.

7. REFERENCES

[1] C. Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[2] E. J. Friedman-Hill. Jess, the java expert system shell, 1999. Available at URL `http://herzberg.ca.sandia.gov/jess/`.

[3] J. Giarratano and G. Riley. *Expert Systems : Principles and Programming (3rd Edition)*. PWS Publishing Company, Boston, 1998.

[4] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 127–138, May 1995.

[5] R. Kimball. Dealing with dirty data. *DBMS Online*, September 1996. Available at URL `http://www.dbmsmag.com/9609d14.htm`.

[6] M. L. Lee, H. Lu, T. W. Ling, and Y. T. Ko. Cleansing data for mining and warehousing. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications (DEXA)*, pages 751–760, 1999.

[7] A. Maydanchik. Challenges of efficient data cleansing. *DM Review*, September 1999. Available at URL `http://www.dmreview.com/editorial/dmreview/print_action.cfm?EdID=1403`.

[8] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the ACM-SIGMOD Workshop on Research Issues on Knowledge Discovery and Data Mining*. Tucson, AZ, 1997.

[9] M. J. Waller. A comparison of two incremental merge/purge strategies. *Master Thesis, University of Illinois*, 1998.