IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4716

Using Neural Networks for Addressing Data Quality During the Software Maintenance Process

Abdullah A. Al-Namlah
Florida Institute of Technology
Department of Computer Sciences
150 West University Boulevard
Melbourne, FL 32901-6975
alnamlah@fit.edu

Shirley Ann Becker
Professor of Computer Information Systems
College of Business Administration
Northern Arizona University
Flagstaff, Arizona 86011
annie.becker@nau.edu

ABSTRACT

The high cost of software maintenance continues to be a great concern for many organizations due to poor data quality that plagues most legacy database systems. It is proposed in this paper that neural net technology be used to accommodate changes in user requirements when data quality is an issue. Neural nets can be trained to identify semantically equivalent data such that source code modifications do not have to be made. A case study is used to illustrate the use of neural nets to replace source code in identifying duplicate data within and across databases even when data is incorrect or incomplete.

INTRODUCTION

Software maintenance continues to challenge both practitioners and researchers alike because of its costs and resource requirements when modifications and enhancements are made to a software system (O'Neal and Carver, 2001; Zhou, et al., 1999). Most software systems are not only large, in terms of source code, but lack proper documentation as to how they are integrated with other system components (Kajko-Mattsson, et al., 2001). As a result, software maintenance is often considered the most difficult aspect of software production because even simple changes can be costly to an organization (Shirabad et al., 2001). Perhaps more importantly, changes to the system may result in corruption of other system components because of a lack of understanding of the overall system architecture. Today, it is estimated that sixty to eighty percent of the total software budget is used to support software maintenance (Stark and Oman, 1997).

Much of the current research on software maintenance focuses on managing functional changes to the source code and other software artifacts. Little has been done in terms of addressing software maintenance from a database perspective, though legacy data continues to be a major part of the maintenance issue. Existing legacy database systems, often required as an integral component of software systems, tend to be filled with incorrect, incomplete, or incompatible data (English, 1999). Legacy databases can add a layer of complexity when trying to modify existing components, as the results may be unpredictable due to poor data quality. Perhaps the most pertinent example of poor data quality was the Y2K problem, which cost organizations trillions of dollars. What is needed then is a means of addressing the data quality issue in order to maintain high-quality software applications.

With the introduction of new technologies, such as provided by the Internet, software systems are being modified to integrate data across new and legacy database systems. Unfortunately, data may not in be the same data format, differ in data type and length, and may be cryptic, incorrect or incomplete. As a result, it is difficult to determine whether data is semantically the same when integrating data across software applications. Though there has been much work done on data cleaning, in order to address the data quality issue, it is often prohibitively expensive. The end result is that data quality compounds the sky-rocketing cost and resource consumption associated with software maintenance. What is needed is a means of determining whether data is semantically equivalent when data quality is an issue during the software maintenance process.

In addition to dealing with the data quality issue, organizations are also struggling with source code updates while adapting systems to accommodate changing user requirements (Bohner, 1996; Lehman, 1994). We propose a technique for minimizing the cost of software maintenance associated with poor data quality while addressing the source code update problem. This innovative approach incorporates the handling of syntactic differences among data from multiple sources in the source code. Neural net technology is used to draw conclusions about the semantic equivalence of data across multiple databases based on a learning algorithm. Source code is modified to include probabilities of data equivalence even when data is syntactically incorrect or incomplete.

THE SOURCE CODE ISSUE

A difficult software maintenance problem resulting from merging or integrating heterogeneous databases is the discovery of records that semantically represent the same object but are syntactically different. Because of this complexity, semantic rules are often applied to the data to uncover data duplication. These records would often require further manual inspection to determine whether they are the same even when data is syntactically different. A semantic rule, for example, might require an inspection of primary and foreign key data to determine semantic equivalence even when part of the data is missing, incorrect, or cryptic. This data discrepancy may have happened when edit checks and data integrity constraints were missing from the original design.

In the business community, solving the data duplication problem by matching data based on semantic equivalence is called the "merge/purge problem". The merge/purge problem refers to the process of identifying duplicate records that semantically refer to the same entity and then merging them to constitute one syntactic and semantic entity (Hernandez & Stolfo, 1995). Data integration does not require that data are merged into one entity, but that data can be linked from one source to another via semantic equivalence. This is often achieved via a primary key and foreign key relationship within and across database systems.

In either case, a major challenge is determining appropriate business rules that identify semantic similarities between records. These rules may be found in the software documentation (specification an design artifacts), or they may have to be extracted from the physical design by reengineering relationships among data within and across database systems. For example, we may have a rule that specifies two customer records representing the same object if their social security

numbers match. This rule is typically part of the source code when manipulating the database components of the software system. The problem is that too often the same data may not be a 100% match because of poor data quality within and across databases. Because the source code would require an "all or nothing" data match, expensive data cleaning may be needed to resolve data quality problems.

Illustration

A data integration problem is presented for a fictitious business that has developed a web-based system inclusive of a database to process online customer orders. This new system is integrated with a legacy database, which contains data about existing customers and their credit ratings. This information is used to determine whether an online order should be processed given the customer's credit status with the organization.

Let's say that we are given a customer table to handle new online customers:

CustomerOnline (CustomerID, LastName, FirstName, MiddleInitial, EmailAddress, AreaCode, PhoneNumber, ExistingCust#).

There is an existing legacy database with customer mail order data used before the inception of the online system:

CustomerMailOrder (Customer#, CreditRating, RatingDate).

Data is integrated via ExistingCust# (foreign key in CustomerOnline) and Customer# (primary key in CustomerMailOrder) because of their matching datatypes representing semantic equivalence. The Customer# is almost always unique but there are duplicate numbers for different customers and partial data due to missing integrity constraints. Given this scenario, it would be difficult to ensure that a new customer record is not inserted into the CustomerOnline table without a link to the CustomerMailOrder table when there is a semantically matching record. The psuedocode, shown below, would code this relationship into an online application:

```
If CustomerID= Customer#
\* record exists in CustomerMailOrder *\
Insert CustomerOnline record with foreign key ExistingCust# =
SocialSecurity#
Else

Insert new customer record into CustomerOnline with ExistingCust#
set to NULL
End If.
```

Notice that the problem with this source code is that it only matches correct and complete data that hasn't been corrupted by long-term use.

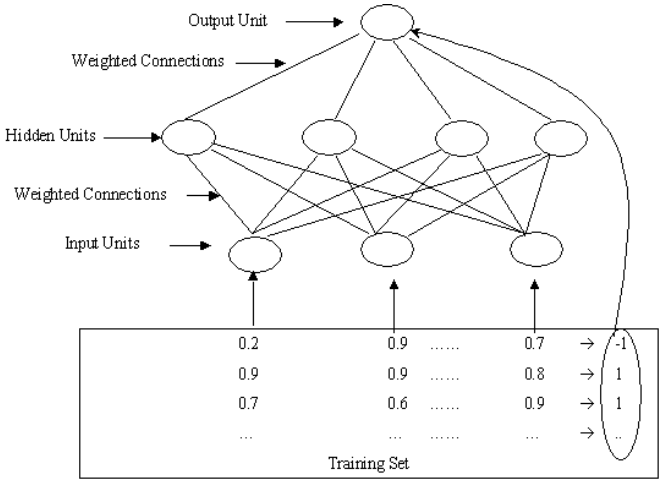
It is proposed that neural net technology be used to improve data retrieval even when data quality is poor. The use of neural net technology in this application adds the flexibility of retrieving records that are semantically the same even though the physical data is not a 100% match. Inflexible, hard-coded rules can be replaced by a set of training examples that minimize the effect of the changes on the source code as well as data quality in existing database systems.

This paper proposes the use of a backpropagation neural net to replace source code for identifying data matches in databases. The use of a backpropagation neural net provides a learning process mechanism for uncovering semantic equivalence among data. A main advantage to this approach is that source code does not have to change whenever a rule for discovering data matches changes.

DISCOVERING DUPLICATE DATA USING BACKPROPAGATION

Backpropagation neural nets have proven to be very effective for a broad range of problems, but are especially useful when there is a large data pool to use during the learning process. Because backpropagation is very useful in recognizing complex patterns in existing data, it is suitable for identifying data duplicates in existing database systems. It has been shown that a neural net can recognize most test patterns (specifying data duplication) after it has been trained to do so (Al-Namlah, et al., 2002).

Figure 1: Neural Net Architecture



The neural net, as shown in Figure 1, is made up of number of highly interconnected input, hidden, and output processing units (Fausett, 1994). The weighted connections between the units are used to store the acquired knowledge. The knowledge is obtained through a training process that uses a data file for training the net.

The Learning Process

Data duplication within and across databases can be discovered using backpropagation via a simple algorithm that selects a set of records, based on some condition, in order to determine data equivalence. Then, percentages of data matching are used as a learning mechanism for the backpropagation algorithm. In this way, the algorithm can be trained to discover duplicate data.

A three-step approach was proposed by Al-Namlah and Becker (2002) in order to use backpropagation to provide feedback on the quality of a data source. This approach was applied in a limited fashion in order to specifically train the backpropagation neural net on what constitutes duplicate records in a database system. The output of the neural net was used to draw conclusions based on its statistical input. In this research, the input to the net was based on existing data quality, as defined by pattern matching in the existing data. This early work is used as a basis for finding duplicate data in order to address the software maintenance problem.

Training Example

During the training phase of building the neural net, rules are identified from which learning can occur. The following example illustrates one of the rules that have been extracted from existing data in a_database. Rules are defined in terms of the percentages associated with data matching between data records.

Rule: Unique identifier data matches 100% and event-driven data has low match rate.

Records Retrieved:

FNAME	MI	LNAME	SSN	DOB	ADDRESS	GENDER	SAL	SUPERSSN	D#
Franklin	T	Wong	333445555	08-dec-45	638 Voss, Houston TX	M	40000	888665555	5
Frank	S	Wong	333445555	08-dec-45	638 Voxx, Houstojn, TX	M	18000	987654321	

Data Matching Percentages:

FNAME	MI	LNAME	SSN	DOB	ADDRESS	GENDER	SALARY	SUPERSSN	D#
63	0	**100**	**100**	**100**	76	100	60	25	0

The data matching percentages, for these two records, are presented above. In this case, the *social security number (SSN), date of birth (DOB), gender* and *last name (LNAME)* are 100% matches. Because these data fields are typically used in databases as part of the unique identifier (or alternate identifiers), data duplication appears to hold. Further analysis of data matching shows that the event-driven data, *salary (SAL), supervisor (SUPERSSN)* and *department* (*D#)* data, differ significantly. However, these low data matching percentagess are considered a reflection of event-driven data. For this scenario, the net would be trained to detect record duplication.

Rule Implementation Without Neural Net Technology

In the example above, a rule has been specified as such, "if the LNAME, SSN and DOB in two data records are exactly the same, then a match occurs." This rule can be coded as (pseudocode):

```
If   (S.LNAME = T.LNAME) and
     (S.SSN = T.SSN) and
      (S.DOB = T.DOB) then
       S and T are semantically equivalent
Else
     S and T are not semantically equivalent
End If.
Where S the source record and T is the target record.
```

When data quality is poor, the rule can be modified to add probability of semantic equivalence. For example, the rule may specify that the DOB in two records must be a 90% or greater match in order to be considered semantically equivalent. The code is updated as follows:

```
If  (S.LNAME = T.LNAME) and
    (S.SSN = T.SSN) and
     Match (S.DOB = T.DOB) ³ 0.9 then
     S and T are semantically equivalent
Else
    S and T are not semantically equivalent
End If.
```

Note: Match *is a function (not shown) determining semantic equivalence between two records.*

Although this user requirement appears to require a trivial change in the source code, there are maintenance costs associated with it. The new function will need to be tested, recompiled and executed. Software artifacts will have to be updated and validated. These artifacts may include system specifications, user requirements, source code, user manuals, configuration management systems, and user interface designs. It is proposed that neural net technology replace the source code function called *Match* in order to minimize software maintenance.

Rule Implementation With Neural Net Technology

Using the backpropagation algorithm, this modified rule can be readily supported via neural net technology by adding another example to the training set that shows a 90% percent match when two DOB data values are equivalent. The new training pattern will eliminate the strict condition that required a 100% match for the DOB field. The neural net will continue to adapt to new user requirements by learning from training examples and changing the weights accordingly.

In, general, the only maintenance needed to satisfy new user requirements is adding one or more training pairs to the training set. It is true that weighted connections will be changed when adding a training pair(s) to the training set; however, the weighted connections are changed automatically without manual intervention. It is important to note that using neural net technology requires no changes to software artifacts inclusive of source code.

CASE STUDY

In this case study, a neural net was built using a backpropagation algorithm and trained to recognize duplicate records. The neural net consisted of 10 input units, one hidden layer with four units and one output unit. The Nguyen-Widrow method (Nguyen and Widrow, 1990) was used to initialize the weights.

Training Set Data

Table 1 shows the training set data with their targets.

Each row in the *Training Patterns* table contains a set of percentages, representing the extent of data equivalence, when two database records are compared. Each column in the table contains a percentage that represents syntactic equivalence when two data values in the records are compared. For example, the "63 %" value in the Pattern Number 1 row represents the extent to which two FNAME data values match. Based on the overall percentage of data matching between records, a data duplication value is assigned to each Pattern Number row. These values, listed in the *Target* column, specify data duplication (value of "+1") or non-duplication (value of "–1"). This target data is used to train the net to determine whether two records are semantically the same.

Notice that there is an implicit rule, "LNAME, SSN, and DOB must match 100%", for semantic equivalence between two records to hold. This rule is implemented within the training set data such that if all three field values are "1" (100% match) then the target value is set to "+1".

Table 2 provides information about five patterns, each of which represents data matching between two records. The neural net was used to determine whether the records were semantically the same given the percentages in the testing pattern columns. Notice that Pattern Number 1, for example, is assigned a 100% ("+1") probability of being duplicate records; whereas, Pattern Number 4 is assigned a 100% probability ("-1") of not being duplicate records. By reviewing all five patterns, it is seen that the neural net has learned the rule (100% match of the three fields LNAME, SSN and DOB is required to conclude data duplication).

Next, we illustrate how neural net technology can be used to accommodate changes in user requirements. Let's say that user requirements changed such that DOB no longer must be a 100% data match.

Table 1: Training Patterns Used to Train the Neural Net

Pattern Number	Training Pattern										Target
	FNAME	MI	LNAME	SSN	DOB	ADDRESS	GENDER	SAL	SUPERSSN	D#	
1	0.63	0	**1**	**1**	**1**	0.76	1	0.6	0.25	0	+1
2	0.8	0	**1**	**1**	**1**	0.77	1	0.4	0.33	0	+1
3	0.77	0	**1**	**1**	**1**	0.78	1	0.55	0.44	1	+1
4	0.66	0	**1**	**1**	**0.5**	1	0	0.66	1	0	- 1
5	0.54	0	**1**	**1**	**0.4**	1	0	0.77	1	0	- 1
6	0.88	1	**1**	**1**	**0.3**	1	1	0.88	1	1	- 1
7	0.77	0	**1**	**1**	**1**	0.66	1	0.55	0.44	0	+1
8	0.44	1	**1**	**1**	**1**	0.33	0	0.2	0.11	0	+1
9	0.22	0	**1**	**1**	**1**	0.16	1	0.33	0.16	1	+1
10	0.8	1	**1**	**1**	**0.6**	1	0	0.57	1	0	- 1
11	0.7	0	**1**	**1**	**0.55**	0.8	1	0.6	0.7	1	- 1
12	0.98	1	**1**	**1**	**0.7**	0.7	1	0.9	1	0	- 1

Table 2: Testing Set Results

Pattern Number	Testing Pattern										Result
	FNAME	MI	LNAME	SSN	DOB	ADDRESS	GENDER	SAL	SUPERSSN	D#	
1	0.63	0	1	1	1	0.76	1	0.6	0.25	0	+1.00
2	0.77	0	1	1	1	0.9	0.7	0.7	0.25	0	+ 0.99
3	0	1	0.1	0.1	0.1	0.6	0.22	1	1	0.11	-1.00
4	0	1	1	1	0.1	0.6	0.22	1	1	0.11	- 1.00
5	0	1	1	1	0.9	0.6	0.22	1	1	0.11	-0.99

Table 3: New Testing Set Results

Pattern Number	Testing Pattern										Result
	FNAME	MI	LNAME	SSN	DOB	ADDRESS	GENDER	SAL	SUPERSSN	D#	
1	0.63	0	1	1	1	0.76	1	0.6	0.25	0	+1.00
2	0.77	0	1	1	1	0.9	0.7	0.7	0.25	0	+ 0.99
3	0	1	0.1	0.1	0.1	0.6	0.22	1	1	0.11	-1.00
4	0	1	1	1	0.1	0.6	0.22	1	1	0.11	- 1.00
5	0	1	1	1	0.9	0.6	0.22	1	1	0.11	+ 0.96

The new rule that is implemented by the neural net is: "LNAME and SSN must match 100% and DOB must match at least by 90%" for semantic equivalence between two records to hold. The training set, used by the neural net, is updated in order to apply this revised rule when identifying semantically equivalent records. Table 1 would be updated with a new Pattern Number row as follows:

FNAME	0
MI	1
LNAME	1
SSN	1
DOB	0.9
ADDRESS	0.4
GENDER	0.29
SAL	1
SUPERSSN	1
D#	0.14
TARGET	+1

Table 3 shows the results of training the neural net using the new rule. Notice that the *Result* data value, for Pattern Number 5, is "+0.96", which means that the two records are found to be duplicates. The neural net has been trained to uncover data matches even when the DOB data isn't a 100% match. There are no changes to the source code required to accommodate this change in user requirements.

FUTURE RESEARCH

The results of the initial work show the use of backpropagation as a viable means of reducing software maintenance when data quality is an issue. The approach needs to be applied to a large software system in order to determine the cost savings and other intangible benefits. There are costs associated with training the neural net when user requirements change, however, they appear to be minimal as a result of automated support. Future research is needed to evaluate the cost of training the net versus writing source code to handle data quality issues in legacy database systems.

REFERENCES

Al-Namlah A. & Becker S. (2002). "Employing Neural Networks to Assess Data Quality", **Proceedings of the 2002 IRMA Conference,** Vol. 1, pages 28-31, Seattle, WA, May 19-22, 2002.

Al-Namlah A., Becker S., and Koksal S., (2002). "Eliminating Data Duplication Using A Backpropagation Neural Net," **Proceedings of the 2nd International Conference on Neural, Parallel, and Scientific Computations,** Vol. 2, pages 37-41, Atlanta, GA, August 07-10, 2002.

Bohner S., (1996). "Impact Analysis in the Software Change Process: A year 2000 Perspective," **Proceedings IEEE International Conference on Software Maintenance (ICSM'96),** Monterey, California, pages 42-51 November1996.

English Larry P. (1999). **Improving Data Warehouse and Business Information Quality, Methods for Reducing Costs and Increasing Profits,** John Wiley & Sons, Inc., New York, NY.

Fausett L. (1994). **Fundamentals of Neural Networks, Architectures, Algorithms, and Applications,** Prentice-Hall, Inc., Upper Saddle River, NJ.

Hernandez M A. and Stolfo, S., (1995) "Merge/Purge Problem for large databases," **Proceedings of the ACM SIGMOD International Conference on Management of Data,** pages 127-138 May 1995.

Kajko-Mattsson M., Forssander S., and Ollsson U., (2001). "Corrective Maintenance Maturity Model (CM^3): Maintainer's Education and Training," **Proceedings of the 23rd International Conference on Software Engineering,** Toronto, Ontario, Canada, 2001.

Lehman M., (1994). "Software Evolution," **Encyclopedia of Software Engineering**, 1994, pages 1202-1208.

Nguyen D., & Widrow B., (1990). "Improving the Learning Speed of Two-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights", **International Joint Conference on Neural Networks**, San Diego, CA, III: 21-26.

O'Neal J. S., Carver L.C., (2001). "Analyzing the Impact of Changing Requirements", **Proceedings IEEE International Conference on Software Maintenance (ICSM 2001)**, Florence, Italy, pages 22-31 November 2001.

Shirabad J. S., Lethbridge T.C. & Matwin S., (2001). "Supporting Software Maintenance by Mining Software Update Records", **Proceedings IEEE International Conference on Software Maintenance (ICSM 2001)**, Florence, Italy, pages 22-31 November 2001.

Stark, G.; and Oman, Paul W. (1997) 'Software maintenance management strategies: observations from the field', **Journal of Software Maintenance**, 9(6), p. 365-378.

Zhou S., Zedan H., Cau A., (2001). "A Framework For Analyzing The Effect Of 'Change' In Legacy Code", **Proceedings IEEE International Conference on Software Maintenance (ICSM'99)**, Oxford, England, pages 411-420 August 1999.