Schema Mapping Using Hybrid Ripple-Down Rules

Sarawat Anam[1,2], Yang Sok Kim[1], Byeong Ho Kang[1] and Qing Liu[2]

[1]{Sarawat.Anam, YangSok.Kim, Byeong.Kang}@utas.edu.au
School of Engineering and ICT, University of Tasmania
Sandy Bay, Hobart, Tasmania, Australia

[2]Q.Liu@csiro.au
Autonomous Systems, Digital Productivity and Service Flagship,
CSIRO Computational Informatics, Hobart, Tasmania, Australia

Abstract

Schema mapping is essential to manage schema heterogeneity among different sources. Schema mapping can be conducted by using machine learning algorithms or by knowledge engineering approaches. These two approaches have advantages and disadvantages. The machine learning approaches can learn their model using the data, but they are static, so they cannot be modified to reflect the domain data changes. Inversely, the knowledge engineering approaches need domain experts, but they can be modified by reflecting the domain data changes. In order to exploit the advantages of both approaches and reduce the limitations, we propose a hybrid approach, called Hybrid-RDR, which combines a machine learning algorithm with ripple-down rules (RDR), an incremental knowledge engineering approach. A model is constructed by a decision tree algorithm and then it is extended by adding rules incrementally. This approach achieves higher performance in terms of precision, recall and F-measure compared to the machine learning algorithm. This significantly reduces the effort for classifying the related schemas one by one by manually creating rules and it is possible to modify the knowledge base by adding rules without creating model again if decision tree gives wrong classifications whenever the schema data changes over time.

Keywords: Machine learning algorithm, Knowledge engineering approach, schema mapping, incremental learning.

1 Introduction

Schema mapping is a set of logical specifications that express correspondences between semantically related schemas of different datasets through the application of a matching algorithm (Shvaiko and Euzenat, 2005). Schema mapping is used in many application domains such as data integration, data exchange, data warehousing and schema evolution (Glavic et al., 2010). Schema matching can be done in the element level and structure level. While the element level matching considers only names of the schemas, the structure level matching uses the result of element level matching for matching the full graph. In this research, we only focus on the element level matching since our major aim is to evaluate whether our proposed approach can be applied to the schema mapping problem and our approach can easily be extended by considering the structure information.

In our research, the schema mapping process is defined as:

$$M = c(S_i, T_j, V_{i,j}),$$

where S_i is a source schema, T_j is a target schema, $V_{i,j}$ is an attribute value vector ($V_{i,j} = \{v_1, v_2, \dots, v_n\}$) and M is a Boolean output (if S_i is matched with T_j, return true; otherwise return false). The attribute values are derived by applying different similarity functions to S_i and T_j, and to the values of S_i and T_j after text pre-processing (e.g., synonym, tokenisation, and reverse abbreviation). Therefore, the schema mapping problem is to find the classification function (c) that accurately predicts a real relation between two schema elements.

Many schema mapping systems (Do and Rahm, 2002a, Aumueller et al., 2005, Doan et al., 2002, Marie and Gal, 2008) have been developed by employing machine learning algorithms and/or knowledge engineering approaches. A machine learning algorithm needs training data set for building models, but usually it is very difficult to obtain fine training datasets. In addition, it is very difficult to change the model by human knowledge. A knowledge engineering approach encodes human knowledge directly, such that the knowledge base can be constructed with limited data, but it requires time-consuming knowledge acquisition.

Schemas can be created over time incessantly, and thus knowledge also changes over time. Machine learning and knowledge engineering approaches manage this phenomenon differently. Machine learning algorithms usually reconstruct their model after collecting sufficient data, while knowledge engineering approaches acquire new knowledge if necessary.

In this research, we propose a hybrid approach that constructs classification model using a machine learning algorithm and maintains new knowledge using a knowledge engineering approach, called ripple-down rules. When new data are available, the classification model may suggest wrong classification for some cases. In this case, it is necessary to add a rule, called *censor rule*, which stops the wrong classification and to add a

rule, called *alternative rule*, which correctly classifies the problem case. When a case is stopped by a censor rule but have no alternative rule that correctly classifies it, the case has no conclusion. In this case, a new rule that classifies the case correctly should be as a child rule of the root node.

In order to simulate the changes of the data, we assume that the data for schema mapping is partially available at the beginning and new data will be added onward. While the hybrid approach improves the performance of schema mapping by adding rules incrementally for correcting schema mapping errors of the current knowledge base, the decision tree improves its performance by learning a new model by including newly available data. Our experimental results show that the Hybrid-RDR approach produces slightly higher performance compared to the decision tree.

2 Related Works

Machine learning techniques have been used in the context of schema matching. Learning Source Descriptions (LSD) (Doan et al., 2001) is a schema mapping system and the extension of LSD is GLUE (Doan et al., 2002) which creates ontology mapping. Both systems use machine learning techniques like Multi-strategy learning approach as base learner, Naïve Bayes for classifying text, and Meta learner for finding matching among a set of instances. Embley et al. (2004) develop an approach based on learning rules of decision trees for discovering hidden mapping among entities. In this approach, the rules are used for matching terms in WordNet. However, the decision trees are not used for choosing the best match algorithms. Duchateau et al. (2008) present an approach for schema matching which uses a decision tree to combine the best suitable match algorithms. The approach inputs a set of schemas and a decision tree which is composed of match algorithms, and outputs a list of mappings which are validated by experts to find out whether the matching is correct or not. The feedback is used to feed into another decision tree for learning. The machine learning techniques generally require refined training dataset that should be prepared by largely in manual and the techniques cannot easily change its model without sufficient data. In the rule based approach, well defined training dataset is not necessary. In the approach, schema matching is started for a small amount of schema data by adding rules.

Some systems have already used rule based techniques for schema matching. Among them, COMA/COMA++ (Do and Rahm, 2002b, Aumueller et al., 2005) are generic schema and ontology matching systems where simple, hybrid and reuse oriented matchers are used. In the systems, schemas are internally encoded as DAGs (Directed Acyclic Graphs) and are analysed using string matching algorithms. Different aggregation functions such as average, minimum, maximum and weighted sum, and rule based techniques are used in the systems for obtaining combined match results. However, in COMA/COMA++, determining best combination of matcher is not easy. YAM (Duchateau et al., 2009) is a machine learning based schema matching factory. In the learning phase, YAM considers users' requirement such as a preference for recall or precision, provided expert correspondences. It uses a Knowledge Base (KB) that consists of a set of classifier, a set of similarity measures, and pairs of schemas which have already been matched. In the matching phase, the KB is used to match unknown schemas. However, in the system, appropriate classifier is selected by users or to use a default classifier learned over a huge mapping knowledge base. In this research, we use the hybrid approach combining decision tree and rule based technique. In our system, the KB is empty at the beginning, and the first rule is added in the KB by classifying a dataset using decision tree learning model. Then rules are added incrementally in order to solve schema matching problems such as un-classifications and wrong classifications.

Traditional rule-based systems require time-consuming knowledge acquisition as in those systems a highly trained specialist, the knowledge engineer, and the time-poor domain expert are necessary in order to analyze domain (Richards, 2009). In order to solve the problem of time consuming knowledge acquisition, we adopt RDR (Ripple Down Rule) (Compton et al., 1991), a well-known incremental knowledge acquisition method. RDR has been successfully applied in many practical knowledge-based system developments. There are several versions of RDR methods, including Single Classification RDR (SCRDR), Multiple Classifications RDR (MCRDR), and Nested RDR. MCRDR is used in order to solve problems in some domains, e.g., pathology, text/web document classification, help desk information retrieval and medication review (Richards, 2009). Since our research aims to find matching relationship of schema (matched or not-matched), SCRDR is chosen for our research.

The success of RDR does not depend on representational differences; rather it largely depends on its distinctive operational semantics on standard production rules (SPR). SPR has the form $p \rightarrow a$, which is interpreted as "**if** a case satisfies condition p **then** do action a". RDR systems in general process cases sequentially and whenever the current knowledge base suggests wrong conclusions, new rules are added. Whenever a new rule is created, it is necessary to validate the rule normally by checking whether or not the future cases are given the correct classifications. If any case is wrongly classified by a rule, then RDR systems acquire exception rules for this particular rule. In this case, the expert directly refines the new rule adding conditions until all incorrect cases are removed. However, it is not easy to construct this kind of rule with resource constraints such as limited time and information. We use CPR (censored production rules) based RDR (Kim et al., 2012), to be used for acquiring exceptions when a new rule is created using censor conditions. CPR has the form $p \rightarrow a \neg c$, which is interpreted as "**if** a case satisfies condition p **then** do action a **unless** the case does not satisfy the censor conditions c." This approach also can provide multiple cornerstone cases that satisfy the main condition clause (positive cornerstone cases) as well as the censored condition clause (negative cornerstone cases). The approach is useful when we have a large number of validation cases at hand.

3 Method

In this section, we describe our proposed Hybrid-RDR approach used for schema mapping.

Hybrid-RDR

Hybrid-RDR approach combines a decision tree algorithm, J48 with CPR based RDR. The process of Hybrid-RDR is given in **Fig.1**. The process is described below:

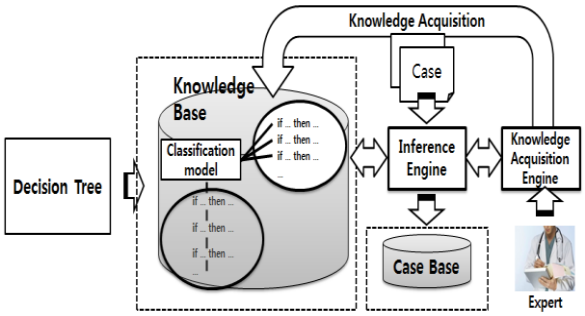

Fig.1 Hybrid-RDR

Any machine learning classification algorithms such as Naïve Bayes, and decision tree, can be used for our hybrid approach. Among them, we choose the decision tree algorithm because its model can be understood by the human expert. A decision tree is a tree whose internal nodes represent the attributes, and the edges represent the conditions on the result of the attributes. All the leaf nodes represent classifications which are either *true* or *false*, indicating whether there is a match or not. We use J48 decision tree, a Java implementation of C4.5 in WEKA (Quinlan, 1993), for classifying schemas. Decision tree inputs a collection of cases (training set) where each case contains a set of features obtained from input schemas and from the application of string similarity metrics and text processing techniques to the input schemas, and also from manually providing class level (true or false). Then it builds up a classification model. The accuracy of the model is evaluated by using a test dataset.

For the knowledge engineering, we chose RDR approach since it overcomes "knowledge acquisition bottleneck" problem of the conventional knowledge engineering approaches by employing error-driven knowledge maintenance strategy, where all rules have clear relationship and they are added as either exception or alternative of the existing rules (Compton et al., 1991). RDR has been successfully applied in many application domains (Richards, 2009). The structure of Knowledge Base (KB) is designed as an n-tree. Each node of the tree is a rule and each rule consists of IF [conditions] THEN [conclusion] UNLESS [censor-condition]. The KB structure and the inference process are shown in **Fig.2**.

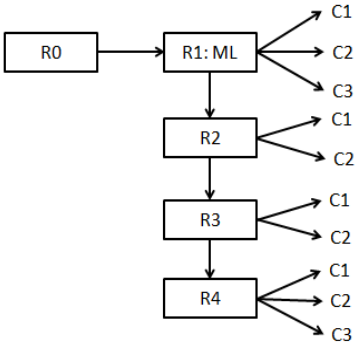

Fig.2 Hybrid-RDR Knowledge Base

In **Fig.2,** at the beginning when KB is empty, we define R0 (rule 0), which is always true. We denote the first level rules by R1, R2, R3 and R4 and the censored rules by C1, C2, and C3. In the approach, the first rule is added in the KB by classifying a dataset using decision tree classification model, ML (machine Learning). Then other rules are added incrementally when schema data changes over time.

The inference process is based on searching the KB represented as a decision list with each decision possibly refined again by another decision list. Once a rule is satisfied by any case, the process evaluates whether or not the censor conditions are matched to the given case. If any censor rule is not satisfied, then the process stops with one path and one conclusion. However, if any censor rule is satisfied, other rules below the rule that was satisfied at the top level is evaluated. The process stops when none of the rules can be satisfied by the case in hand.

Knowledge acquisition is a process which transfers knowledge from human experts to knowledge based systems. Knowledge acquisition process can be divided into three parts. Firstly, a correct classification should be decided by the expert. Secondly, new rules' locations should be specified by the system. Thirdly, new rule's condition should be decided by the expert. If the current knowledge base suggests wrong classification, it is necessary to add a censor rule that has NULL as classification. If the current knowledge base suggests no classification for any case, a new rule should be added as an alternative rule, which is added as a child rule of the root node of the knowledge base. The cases used for creating rules are called cornerstone cases and they are used in consequent knowledge acquisition process (Compton and Jansen, 1990).

The advantage of Hybrid-RDR approach is that only one classification model is created by decision tree for a small amount of schema data and knowledge base is then built incrementally by adding rules to solve schema matching problems. The process helps to reduce time in two ways. Firstly, it does not create classification model when schema data changes over time. Secondly, it does not classify all the related schemas one by one by manually creating rules. The Hybrid-RDR approach is useful where there are large numbers of validation cases are at hand.

4 Experimental Design

4.1 Datasets

Four XDR schemas of purchase order domain, such as EXCEL, CIDX, NORIS, and PARAGON, obtained from www.biztalk.org are used for this evaluation study. We denote the schema datasets EXCEL, CIDX, NORIS and PARAGON by E, C, N, and P respectively. These schema datasets are used for some schema matcher evaluation (Do and Rahm, 2002b). These schema datasets contain different types of features such as identical words, combined words, abbreviated words and synonym words. Each schema dataset contains 35 (E), 30 (C), 46 (N), and 59 (P) schema names.

4.2 Experimental Procedure

In this research, we experiment ten matching tasks one by one. For this, we take all the combinations of six schema datasets such as E-C (first combination is to deal with two datasets, EXCEL and CIDX), E-N, E-P, C-N, C-P and N-P. Then we take the Cartesian product of the six schema datasets separately. The sizes of Cartesian product of the datasets are 1050 (E-C), 1610 (E-N), 2065 (E-P), 1380 (C-N), 1770 (C-P) and 2714 (N-P) entity pairs. We combine all the entity pairs and get total 10589 entity pairs. We randomly divide the entity pairs into ten for creating ten datasets (D1 to D10) where datasets D1 to D8 and D10 contain 1058 entity pairs and D9 dataset contains 1067 entity pairs. These ten datasets are used for ten matching tasks. In order to use the datasets for classification and to give proper knowledge to the users for creating rules, we construct attributes as follows:

Attributes Construction. In order to give proper knowledge to the users, attributes are constructed in three steps:

- The input schema names (source and target);
- Application of text processing approaches such as tokenization, abbreviations and acronyms expansion, and synonym lookup on the input schemas. In tokenization and word separation, schema names containing multiple words are split into lists of words by a customizable tokenizer using punctuation, uppercase, special symbols, whitespace and digits. For instance, "contactEmail" is split into "contact" and "Email". Abbreviations and acronyms are expanded by using external resources such as a dictionary and/or a thesaurus. For instance, "tel" is expanded into its original form "telephone". For this, we use the abbreviation file created for COMA (Do and Rahm, 2002b). Synonym processing is applied to use semantically identical schema names to measure similarity (e.g., 'Invoice' is semantically same as 'Bill' in purchase order domain). We use the synonym file created for COMA (Do and Rahm, 2002b).
- Application of the string similarity metrics on the features of the attributes computed from step 1 and 2, which creates another attributes. We use string similarity metrics developed by two open source projects. For Levenshtein, JaroWinkler, Jaro Measure, TFIDF and Jaccard, we use open source library SecondString[1] and for Monge-Elkan, Smith-Waterman, Needleman-Wunsch, Q-gram and Cosine we use SimMetric open source library[2]. Similarity values are normalized, such that the value within from 0 to 1, where 0 means strong dissimilarity and 1 means strong similarity. The threshold values for deciding schema matching (true/false) are increased with 0.1 from 0 to 1. We also provide class level (true or false) manually which creates another attribute. In such a way we get 73 attributes by using schema information of two datasets (one matching task). Computed attributes represent knowledge about a relation between attributes, operator or process patterns. After preparing the attributes and the schema data under the attributes, all these are fed in to the dynamic decision tree algorithm and the Hybrid-RDR. The dynamic decision tree algorithm learns a new model by including newly available data. The evaluation approach is shown in **Fig.3**.

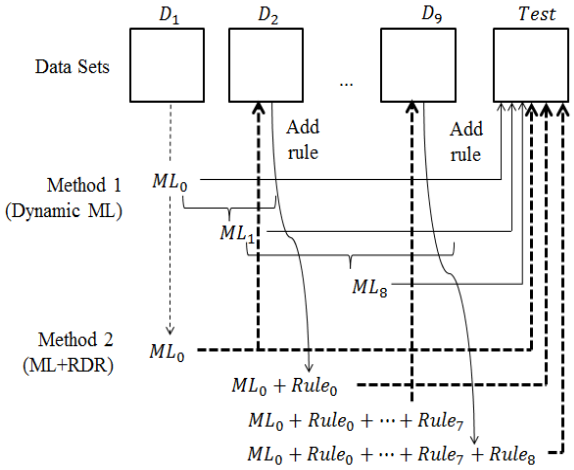

Fig.3 Dynamic ML and hybrid-RDR

In the evaluation approach, we randomly select datasets for training and testing. For example, we select D1 for training and D10 for testing.

Dynamic ML. In the dynamic machine learning approach, we create decision tree model, ML_0 for D1 and test D10. Then we incrementally add other datasets like D1+D2, D1+D2+D3 for creating decision tree models, ML_1, ML_2 and test D10. In this way, we add all nine datasets for creating decision tree model, ML_8 and test D10.

Hybrid-RDR. In Hybrid-RDR approach, we create decision tree model, ML_0 for D1 and test D10. We also test D2 and find some un-classified cases and wrong classified cases. Then we refine the decision tree rule by adding censor rules, $Rule_0$ and again classify the cases by adding alternative rules, $Rule_0$. The censor rules are added as censor nodes of decision tree in the KB and alternative rules are added as parent rules in the KB. The ML_0+$Rule_0$ is then used for testing D10 and also for testing D3. We add rules, $Rule_1$ again for the wrong

[1] http://secondstring.sourceforge.net
[2] http://sourceforge.net/projects/simmetrics

classified cases of D3, and ML_0+$Rule_0$+$Rule_1$ is used for testing D10. In such a way, we incrementally add rules for all nine datasets, ML_0+$Rule_0$+$Rule_1$+…+$Rule_8$ and test D10. The detail rule creation process for schema mapping is described in the following:

Schema Mapping by Hybrid-RDR. A simple GUI (Graphical User Interface) is created which can select any datasets from repository. The attributes that are created by the above steps of *Attribute construction* are represented in a "**Case Browser"** to provide sufficient knowledge to the users (**Fig. 4**). The system works in two phases: Training phase and classification phase. In the training phase, "**Training by DT**" of **Fig. 4** is used. We use the button in order to train one dataset using decision tree, J48. The attributes which are created by the above steps of *Attribute construction*, are used as training sample to build a model. The purpose of building a model is to classify whether a given entity pair of schema names is matched or not based on their feature similarity measure. For all machine learning techniques, we consider 10-fold cross validation. 10-fold cross validation means that the data is split into 10 groups where nine groups are considered for training and the remaining one group is considered for testing. This process is repeated for all 10 groups. In the classification phase, "**Classify**" button of **Fig. 4** is used. For matching entity pair of schema names using the algorithm, we provide the attributes created from another datasets. Finally, we get the matching results as true positive (if reported match by expert is true and predicted match by algorithm is true), false positive (if reported match is false and predicted match is true), true negative (if reported match is false and predicted match is false) and false negative (if reported match is true and predicted match is false) which are displayed in **Fig. 4**.

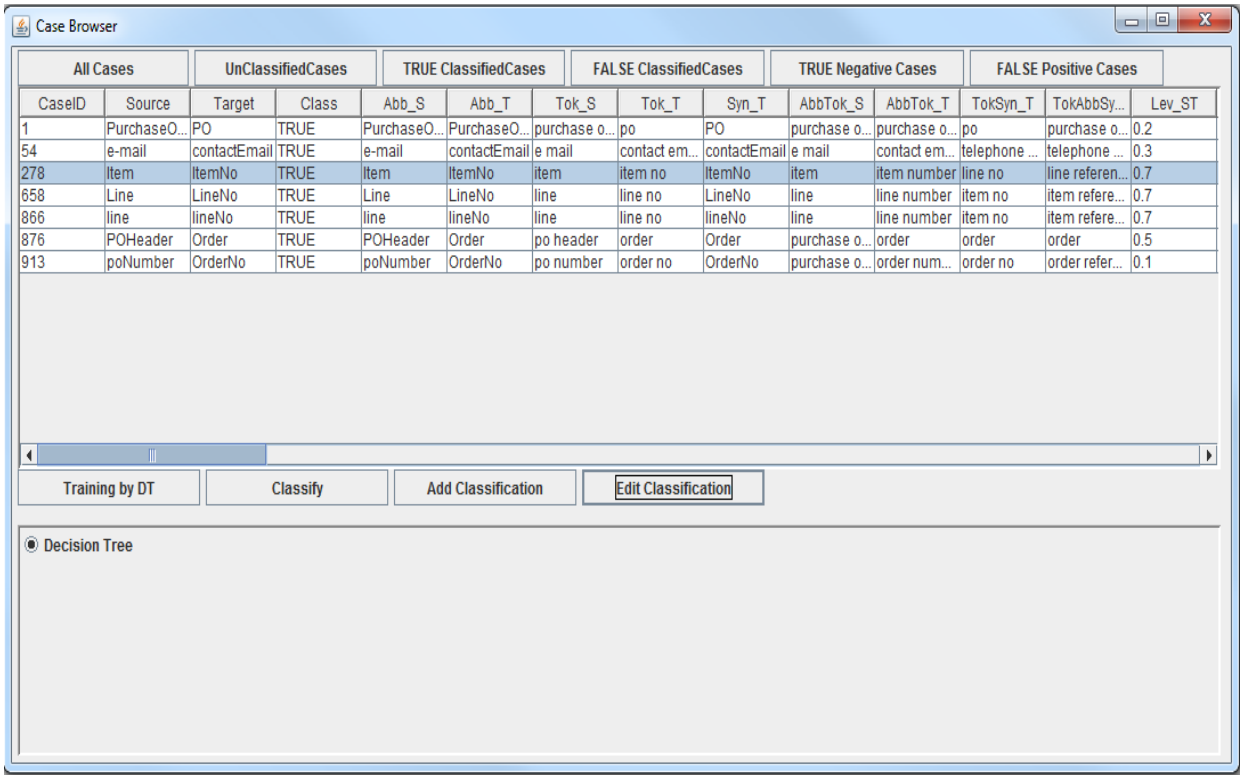

Fig. 4. GUI represents 73 attributes with schema names (all the attributes are not visible)

In order to solve the problem of false negative and false positive (wrong classifications), we use "**Edit Classification**" of **Fig. 4**. "**Edit Classification**" button helps to refine the wrong classified cases by adding new conditions until all incorrect cases are removed or creating another new rule using knowledge Acquisition GUI. Classification for the censor rule is always "**NULL**". For editing classification, the Knowledge Acquisition GUI is displayed **in Fig. 5**.

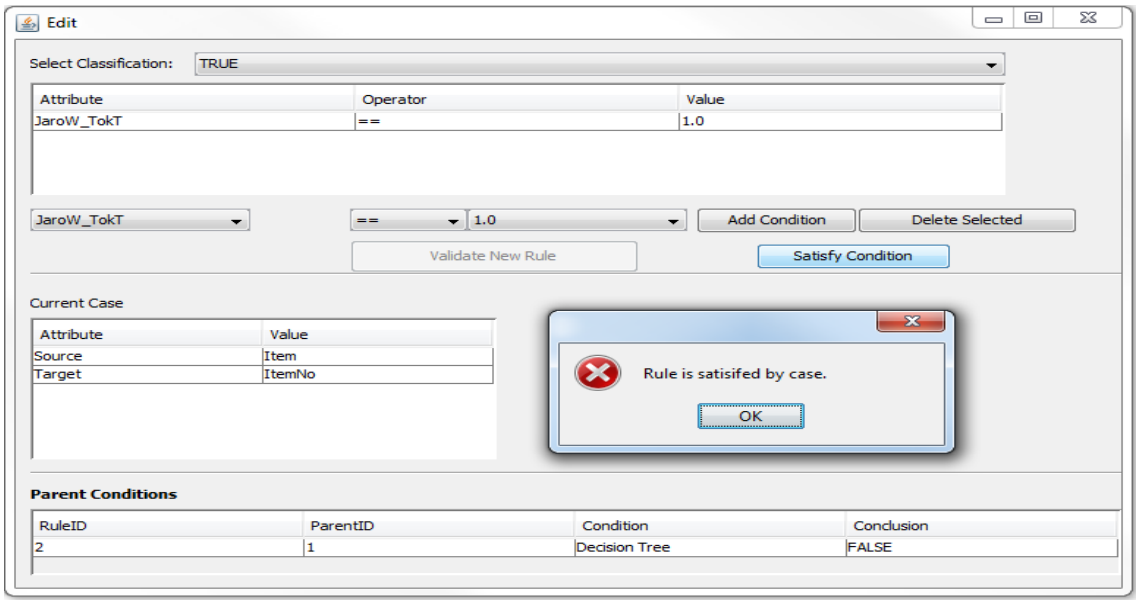

Fig. 5. Knowledge Acquisition GUI for editing rules

In **Fig.5,** parent condition is decision tree which gives the wrong classification for the current case. In order to edit the parent rule, it is not necessary to select the classification as classification for the censor rules is always "NULL". First, the rule conditions are added. For each condition in the rule, the *attribute*, *operator*, and *value* are selected from the drop down boxes, which list all the attributes, operators and values respectively. After selecting condition**, "Add Condition"** adds condition**.** It is possible to add more than one condition and delete condition using **"Delete Selected"** button if users think that the added condition is not suitable. **"Satisfy Condition"** button helps to look at whether the rule is satisfied by the selected case or not. If rule is satisfied, the "**Validate New Rule**" becomes active and this helps to validate the rule on the un-classified and wrong classified cases of the dataset (**Fig. 6**).

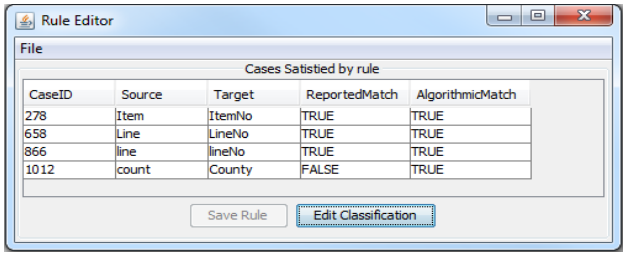

Fig. 6. Rule validation

In **Fig. 6,** *Reported Match* shows the manual matching results and *Algorithmic Match* shows the results calculated from rules. The "**Save Rule**" button helps to save rule in the rule database (KB) and case in the case database. "**Edit Classification**" button helps to refine the wrong classified cases by adding new conditions until all wrong cases are removed or creating another new rule using knowledge Acquisition GUI. Classification for the censor rule is always "**NULL**". The refined cases and the deleted wrong classified cases from the satisfied cases list are shown in **Fig. 7**.

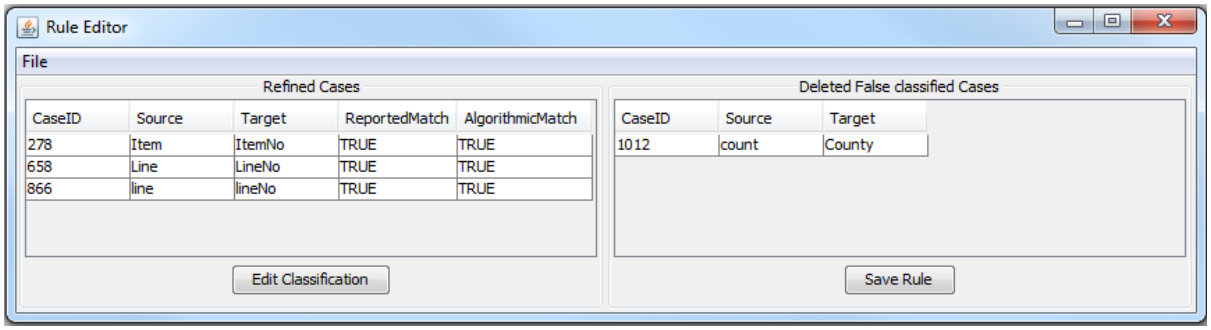

Fig. 7. GUI for refined cases and the deleted cases

In **Fig. 7,** the "**Save Rule**" button saves the censor rule in the rule database (KB) as censor node and the deleted cases in the case database as NULL classification. If there are more wrong classified cases, the rule can be refined by adding other censor rules. Then to classify the "NULL' classified cases, the alternative rules are created by "**Add Classification**" button of **Fig. 4,** are used. For adding classification, the Knowledge Acquisition GUI is like **Fig. 5**. In this case, first the classification of the rule is selected. This can be done using the drop down box at

the top, which lists TRUE or FALSE classifications for this domain. Having selected the classification, the conditions for creating rule are added. Then it is checked whether the rule is satisfied by the current case or not. If the rule is satisfied, then it is validated to determine whether the conclusion provided by the rule is matched with the reported match. The alternative rule is saved in the KB as parent rule. If any case is wrongly classified by the current rule, then the classification is edited.

4.3 Evaluation Metrics

As this task is a classification task, we use the following conventional metrics: precision = $\frac{TP}{TP+FP}$, recall = $\frac{TP}{TP+FN}$ and F-measure = $\frac{2*precision*recall}{precision+recall}$, where TP is True Positive (hit), FP is False Positive (false alarm, Type I error) and FN is False Negative (miss, Type II error). For a specific threshold value, we calculate TP, FP and FN by comparing manually defined matches (R) with the predicted matches (P) returned by the matching algorithms according to Jimenez et al. (2009).

5 Evaluation Results

Performance of the Hybrid-RDR method and dynamic decision tree depend on the features of the datasets which are created using string similarity metrics and text processing techniques. The performance of Hybrid-RDR method also depends on the efficient rule creation. We compute performance in terms of precision, recall and F-measure. Precision estimates the reliability of the match predictions and recall specifies the share of real matches. During schema mapping, manually matching schemas of two heterogeneous data sources and false identified matches by algorithms are handled by humans. The burden of deleting false identified matches is much easier than creating manual matches among thousands of schemas (Stoilos et al., 2005). As for calculating recall value, manually identified matches are necessary, so recall value is very important. Only precision or recall cannot estimate the performance of match algorithms (Cheng et al., 2005). So it is necessary to calculate the overall performance or F-measure of rule based system and machine learning techniques using both precision and recall. For this, we determine the best performing classification system based on the optimized F-measure (Marie and Gal, 2008) for almost all experimental datasets. For all experiments using decision tree, we use WEKA (Hall et al., 2009) data mining and machine learning toolbox.

5.1 Schema Mapping Results

In the experiment, we randomly select datasets for training and testing. We do three experiments to get the performance of dynamic decision tree and Hybrid-RDR method. The performances (precision, recall and F-measure) of schema mapping using dynamic decision tree and Hybrid-RDR, and the rules used by Hybrid-RDR method are described in **Fig. 8**.

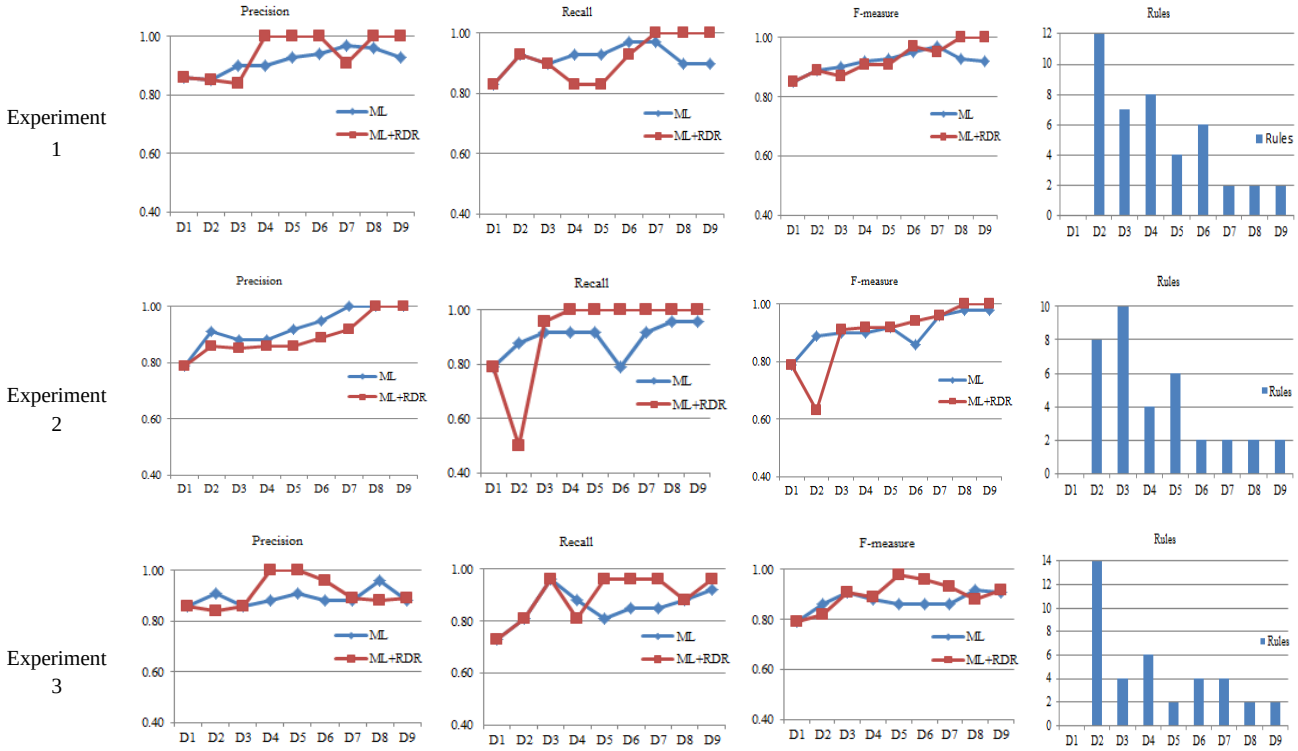

Fig. 8. Schema mapping results using dynamic decision tree and Hybrid-RDR

In **Fig. 8**, for all experiments, ML means the results that are produced by dynamic decision tree and ML+RDR means the results that are got by using Hybrid-RDR. In all experiments, we randomly select one dataset for training and other dataset for testing. In dynamic decision tree method, we create decision tree model, ML_0 for one dataset and use ML_0 for testing the test dataset. Then we select another dataset and add the previous dataset for which ML_0 has been created, with the current selected dataset, and create ML_1 and use ML_1 for testing the test dataset. In this way, we create ML for all the datasets except test dataset and use ML for testing the test dataset.

In Hybrid-RDR approach, we create decision tree model, ML_0 for one dataset and use ML_0 for testing the test dataset. We also select another dataset and use ML_0 for testing and find some un-classified cases and wrong classified cases. Then we refine the decision tree rule by adding censor rule, $Rule_0$ and again classify the cases by adding alternative rules, $Rule_0$. Total $Rule_0$ is 12, 12, 14 for experiment1, experiment2 and experiment3 respectively. The ML_0+$Rule_0$ is then used for testing the test dataset and also for testing another dataset. We add rules again for the wrong classified cases of another dataset, and total $Rule_1$ is 7, 10 and 4 for experiment1, experiment2 and experiment3 respectively. The ML_0+$Rule_0$+$Rule_1$ is used for testing the test dataset and also for testing another dataset. In such a way, we add rules incrementally for all nine datasets, ML_0+$Rule_0$+$Rule_1$+…+$Rule_8$ and use for testing the test dataset. In the table, we also see that the number of rules addition for wrong classifications decreases gradually.

The results indicate that using ML+RDR, the performance is higher than ML in almost all experiments in terms of precision, recall and F-measure. In experiment2, though the performance of ML is higher according to precision for almost all datasets except D1, D8 and D9, but recall and F-measure using ML are not higher than ML+RDR. The reason of high precision means less false positive values, and low recall means that the false negative numbers are high (Marie and Gal, 2008).

5.2 Prune Tree and Knowledge Base

As an example of prune tree for training one dataset and two datasets using J48 is given in **Fig. 9(a)** and **9(b)** respectively. It is found that the prune tree for training one dataset is different from the prune tree of training two datasets.

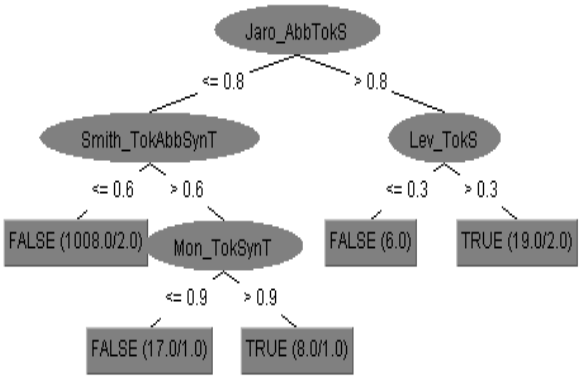

Fig. 9 (a). J48 Prune Tree for training one dataset

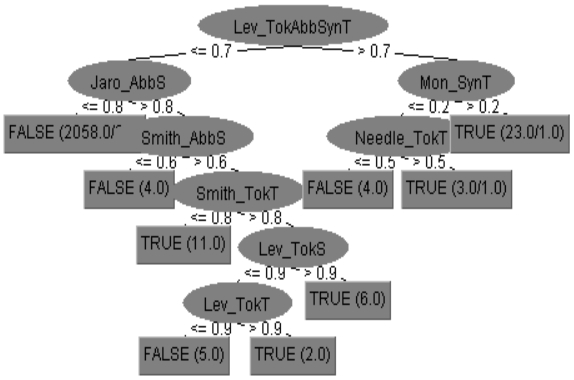

Fig. 9 (b). J48 Prune Tree for training two datasets

Example of Knowledge Base (KB) of Hybrid-RDR which is created for solving un-classifications and wrong classifications is given in **Table 1**.

RID	PID	Condition	Conclusion	CaseID
1	0	0	0	0
2	1	Decision Tree	TRUE/FALSE	ALL
3	2	Lev_AbbTokT==1.0	NULL	643
4	1	Lev_AbbTokT==1.0	TRUE	643
5	2	JaroW_ST == 0.9	NULL	272
6	1	JaroW_ST == 0.9	TRUE	272
7	2	Lev_ST <= 0.2	NULL	63
8	1	Lev_ST <= 0.2	FALSE	63
9	6	Lev_ST == 0.6 && JaroW_ST == 0.9	NULL	818
10	1	Lev_ST == 0.6 && JaroW_ST == 0.9	FALSE	818
11	2	Lev_TokSynT==1.0	NULL	978
12	1	Lev_TokSynT==1.0	TRUE	978

Table 1. Knowledge Base (KB) for creating rules using Hybrid-RDR

In **Table 1,** the attributes *RID*, *PID*, *Condition*, *Conclusion* and *CaseID* means rule id, parent rule id, condition for the rules, conclusion produced by rules and the classified case id respectively. In addition, *Lev*, *S*, *T*, *AbbTokT*, *TokSynT*, and *JaroW* means Levenshtein function, source schema, target schema, abbreviation and tokenization of target, tokenization and synonym of target, and JaroWinkler function respectively. The values 1.0, 0.9, 0.2, 0.6 are thresholds. Example of rule, JaroW_ST==0.9 means if the value of JaroWinkler function applied on source and target equals to the threshold value 0.9, then the conclusion is TRUE.

In the table, rule 1 is always true. We use rules 2 to 12 for classifying cases of datasets. We apply rule 2 to classify one dataset and test dataset. In order to solve un-classification and wrong classification of one dataset, we create rules 3 to 8 and apply rules 1 to 8 for classifying another dataset and test dataset. Then we add rules 9 to 12 for solving wrong classification of another dataset which incrementally build the knowledge base. In **Table 1,** we see that the same rules, for example 3 and 4 are used for making NULL and TRUE conclusion. The reason is that first we create censor rule, for example rule 3 for making wrong classification as NULL classification. Then we add

alternative rule, for example rule 4 for making the right classification.

In **Fig. 9(a), 9(b)** and **Table 1,** we find that though the rules of training one dataset and two datasets are different using the dynamic decision tree, the rules of classifying one dataset and another dataset are not different using Hybrid-RDR, rather we add rules incrementally for solving wrong classifications. Therefore, the advantage of Hybrid-RDR compared to dynamic decision tree is that we do not need to create training model whenever the schema data changes over time.

6 Discussion

In order to solve the problem of a machine learning algorithm that needs training data set for building models, and a knowledge engineering approach that requires time-consuming knowledge acquisition when schema data changes over time, we propose a Hybrid-RDR. The advantage of Hybrid-RDR is that classification model is built by decision tree only for a small amount of schema data, and knowledge base is then built incrementally by adding rules to solve schema matching problems: un-classifications and wrong classifications. In this research, we only focus on element level schema matching using Hybrid-RDR in order to determine whether our proposed approach can be applied to the schema mapping problem. In future, we will perform structure level matching with our element level matching by Hybrid-RDR to improve the performance.

7 Conclusion and Future Works

In this research, we have proposed Hybrid-RDR approach by combining decision tree, J48 and CPR based RDR. We have computed attributes from the input schemas as well as from the application of text processing techniques and string similarity metrics on the schema names. In addition, we have designed a schema mapping tool and used the attributes in order to create rules using Hybrid-RDR. It can handle two problems of schema matching, un-classifications and wrong classifications using incremental knowledge acquisition techniques. We have also used the attributes to feed into a machine learning technique, dynamic decision tree and have compared the performance of Hybrid-RDR and dynamic decision tree for schema mapping. We have found that our Hybrid-RDR method shows slightly better performance than the dynamic decision tree. The main advantage of Hybrid-RDR compared to dynamic decision tree is that it is not necessary to create models whenever the schema data changes over time. The model which is created for one dataset, can be used for classifying another dataset, and rules can be added incrementally for solving wrong classifications. Later the same model and the added rules can be used for classifying another dataset. In this research, we have only considered element level matching, but accurate results of this element level matching should be a premise to work in the next step with structure level matching.

Acknowledgement

The Intelligent Sensing and Systems Laboratory and the Tasmanian node of the Australian Centre for Broadband Innovation are assisted by a grant from the Tasmanian Government which is administered by the Tasmanian Department of Economic Development, Tourism and the Arts.

References

Aumueller, D., Do, H-H., Massmann, S. and Rahm, E (2005): Schema and ontology matching with COMA++. In *Proceedings of the ACM SIGMOD international conference on Management of data*, ACM, 906-908.

Cheng, W., Lin, H. and Sun, Y. (2005): An efficient schema matching algorithm. In *Knowledge-Based Intelligent Information and Engineering Systems*, Springer, 972-978.

Compton, P., Edwards, G., Kang, B., Lazarus, L., Malor, R., Menzies, T., Preston, P., Srinivasan, A. and Sammut, C. (1991): Ripple down rules: possibilities and limitations. In *Proceedings of the Sixth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Calgary, Canada, University of Calgary, 6-1.

Compton, P. and Jansen, R. (1990): A philosophical basis for knowledge acquisition. In *Knowledge acquisition,* 2(3)**,** 241-258.

Do, H-H. and Rahm, E. (2002a): COMA: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*, Hong Kong, China.

Doan, A., Domingos, P. and Halevy, A.Y. (2001): Reconciling schemas of disparate data sources: a machine-learning approach. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data,* Santa Barbara, California, USA, ACM.

Doan, A., Madhavan, J., Domingos, P. and Halevy, A.(2002): Learning to map between ontologies on the semantic web. In *Proceedings of the 11th international conference on World Wide Web*, ACM, 662-673.

Duchateau, F., Bellahsene, Z. and Coletta, R. (2008): A flexible approach for planning schema matching algorithms. In *On the Move to Meaningful Internet Systems: OTM 2008.* Springer.

Duchateau, F., Coletta, R., Bellahsene, Z. and Miller, R.J. (2009): Yam: a schema matcher factory. In *Proceedings of the 18th ACM conference on Information and knowledge management*, ACM, 2079-2080.

Embley, D.W., Xu, L. and Ding, Y. (2004): Automatic direct and indirect schema mapping: experiences and lessons learned. In *ACM SIGMod Record,* 33(4)**,** 14-19.

Glavic, B., Alonso, G., Miller, R.J. and Haas, L.M. (2010): TRAMP: Understanding the behavior of schema mappings through provenance. In *Proceedings of the VLDB Endowment,* 3(1-2)**,** 1314-1325.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I.H. (2009): The WEKA data mining software: an update. In *ACM SIGKDD explorations newsletter*, 11(1)**,** 10-18.

Jimenez, S., Becerra, C., Gelbukh, A. and Gonzalez, F. (2009): Generalized monge-elkan method for approximate text string comparison. In *Computational Linguistics and Intelligent Text Processing*, Springer, 559-570.

Kang, B., Compton, P. and Preston, P. (1995): Multiple classification ripple down rules: Evaluation and possibilities. In *The 9th knowledge acquisition for knowledge based systems workshop*.

Kim, Y.S., Compton, P. & Kang, B.H. (2012): Ripple-down rules with censored production rules. In *Knowledge Management and Acquisition for Intelligent Systems*, Springer, 175-187.

Marie, A. & Gal, A. (2008): Boosting schema matchers. In *On the Move to Meaningful Internet Systems: OTM*, Springer, 283-300.

Quinlan, J.R. (1993): C4. 5: programs for machine learning. *Morgan kaufmann* (1).

Richards, D. (2009): Two decades of ripple down rules research. In *The Knowledge Engineering Review*, 24(2)**,** 159-184.

Shvaiko, P. and Euzenat, J. (2005): A survey of schema-based matching approaches. In *Journal on Data Semantics IV*, Springer, 146-171.

Stoilos, G., Stamou, G. and Kollias, S. (2005): A string metric for ontology alignment. In *Proceedings of the 4th international conference on The Semantic Web*, Galway, Ireland: Springer-Verlag.