A Practical Heterogeneous Classifier for Relational Databases

Geetha Manjunath*, M Narasimha Murty
Indian Institute of Science
{geetham,nnm}@csa.iisc.ernet.in

Dinkar Sitaram
*Hewlett Packard
dinakar.sitaram@hp.com

Abstract
Most enterprise data is distributed in multiple relational databases with expert-designed schema. Using traditional single-table machine learning techniques over such data not only incur a computational penalty for converting to a "flat" form (mega-join), even the human-specified semantic information present in the relations is lost. In this paper, we present a two-phase hierarchical meta-classification algorithm for relational databases with a semantic divide and conquer approach. We propose a recursive, prediction aggregation technique over heterogeneous classifiers applied on individual database tables. A preliminary evaluation on TPCH and UCI benchmarks shows reduced training time without any loss of prediction accuracy.

1 Introduction
Knowledge mining by using state-of-the-art machine learning techniques over real enterprise data would be very valuable for business intelligence applications. However, while enterprise data resides in multiple relational databases with expert-designed schema, most traditional classification algorithms still assume that the dataset is available in a single table, a flat representation of data attributes. Applying single-table data mining techniques to such distributed relational data not only incurs a computational penalty for converting to a "flat" form (mega-join), even the human-specified semantic information present in the relations/schema is lost. Furthermore, relational classification algorithms that use inductive logic programming or probabilistic relational models also cannot be directly applied to such databases as they need propositionalization and multiple database scans for predicate selection. There is a need for accurate classification algorithms that can work directly on relational data distributed across multiple database tables.

We propose a heterogeneous classifier for relational databases (with a novel recursive prediction-aggregation technique) that has the following benefits

- It works directly on existing databases without any data transformation or fusion of the database tables (saving time and space needed for SQL JOIN).
- It leverages application knowledge hidden in the database design and can be automated using standard models to represent semantics (such as RDF).

The structure of this document is as follows. Section 2 gives related work. Section 3 describes our proposed algorithm with a brief proof. Section 4 details the implementation and results. We conclude in section 5.

2 Related Work
Graph-NB [1] is also a modified Naive Bayes algorithm for classifying relational datasets. They use a relationship graph like ours but for feature elimination. However, their approach assumes mutual independence of all attributes and also requires the probability of table linkages, which makes it somewhat impractical. Yin, Han et al [2] propose Cross-Mine, an algorithm that uses a meta-learner to learn from the results of the classifiers applied on individual relational databases. We compare our algorithm with theirs to show better performance and accuracy without the second phase training.

Even [4] and [5] use a 2-step NB algorithm, but the first step extracts the relevant first-order features using ILP which requires transformation. Hongyu Guo, et al [3] propose use of multi-view methods under a concept learning formulation. While, use of SQL views to capture the dataset for first level classification is somewhat similar to ours, they use additional aggregation based features (COUNT, SUM, AVG) to summarize properties of a set of related objects. Our solution optionally requires only class priors (just COUNT) on the fused dataset.

3 Proposed Solution
We propose a two-phase classifier with a semantic divide and conquer approach. First, we learn multiple classifiers from individual database tables with propagated class labels. During classification, these table-level classifiers are used on an appropriate subset of test data (first phase). An aggregated likelihood is computed for every class by collating these inferred posterior probabilities in second phase. Expert database designers typically group sets of related attributes of an
Table 1: Research Employees

<table>
<thead>
<tr>
<th>ENo</th>
<th>Sex</th>
<th>Age</th>
<th>Divn</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>F</td>
<td>38</td>
<td>Mob</td>
<td>Y</td>
</tr>
<tr>
<td>E2</td>
<td>M</td>
<td>51</td>
<td>HCI</td>
<td>N</td>
</tr>
<tr>
<td>E3</td>
<td>M</td>
<td>42</td>
<td>SoA</td>
<td>Y</td>
</tr>
<tr>
<td>E4</td>
<td>F</td>
<td>35</td>
<td>Grid</td>
<td>N</td>
</tr>
<tr>
<td>E5</td>
<td>M</td>
<td>22</td>
<td>Mob</td>
<td>?</td>
</tr>
</tbody>
</table>

Table 2: Division (extended view)

<table>
<thead>
<tr>
<th>Divn</th>
<th>Rank</th>
<th>Size</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mob</td>
<td>2</td>
<td>&gt; 10</td>
<td>Y</td>
</tr>
<tr>
<td>HCI</td>
<td>2</td>
<td>&gt; 10</td>
<td>N</td>
</tr>
<tr>
<td>SoA</td>
<td>1</td>
<td>&lt; 10</td>
<td>Y</td>
</tr>
<tr>
<td>Grid</td>
<td>2</td>
<td>&lt; 10</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 3: Publications (extended view)

<table>
<thead>
<tr>
<th>PNo</th>
<th>Type</th>
<th>Tier</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Conf</td>
<td>1</td>
<td>E1</td>
</tr>
<tr>
<td>P2</td>
<td>Conf</td>
<td>2</td>
<td>E2</td>
</tr>
<tr>
<td>P3</td>
<td>Conf</td>
<td>3</td>
<td>E3</td>
</tr>
<tr>
<td>P4</td>
<td>Jour</td>
<td>1</td>
<td>E1</td>
</tr>
<tr>
<td>P5</td>
<td>Jour</td>
<td>1</td>
<td>E4</td>
</tr>
<tr>
<td>P6</td>
<td>Jour</td>
<td>2</td>
<td>E2</td>
</tr>
<tr>
<td>P7</td>
<td>Conf</td>
<td>2</td>
<td>E5</td>
</tr>
</tbody>
</table>

Figure 1. Example Relational Database

Figure 2. The Join Graph for the example

Entity into separate database tables and explicitly state the relationships between the tables (primary-foreign keys). We use this implicit knowledge of attribute dependencies to generate appropriate views of the tables for the first phase (using a Join Graph). The novel recursive aggregation technique we propose ensures that the node-level computation is just based on its own records and records from its immediately related tables. We describe these in the rest of the section.

Consider a sample application containing three database tables Researcher, Division and Publications shown in Figure 1. The classification under interest is that of predicting the readiness of the researcher for a promotion. We need to find the class label of the research employee E5 (fill missing values in the DB) using all the relevant tables but without using additional space for table fusion. Actually, the class label of interest (target attribute, status) is available only in one table. The dotted columns in tables 2 and 3 show the propagated class label during data preparation (forming the extended views of the tables described in step 5 of §3.1.2).

We introduce a concept of Join Graph to represent the semantic dependency between database tables. The edges in the Join graph represent (a) primary-key to foreign-key relationship (b) foreign-key to primary-key relationship and (c) Additional expert-specified attribute relationship. The Join Graph corresponding to the example tables in Figure 1 is shown in Figure 2.

**Definition:** A Join Graph is an undirected, labeled graph (V, E, W), where V is a set of vertices, one vertex per table in the database, E is a set of labeled edges and W is the set of labels (attribute pairs). An edge (V_i, V_j) with label < l_i, l_j > ∈ W exists if Table V_i can be directly operated with Table V_j using a SQL JOIN clause T_i.l_i = T_j.l_j.

### 3.1 Detailed Algorithm

We now describe the overall algorithm for predicting the class label of the test records. Algorithm NBSplit-Train will be used for data preparation (including creation of extended views of the tables by class label propagation) and training phase. Algorithm NBSplit-Test is used for classifying the test samples.

#### 3.1.1 Notations and Assumptions

Below are some symbolic notations used in this section.

- The Dataset, X contains N database tables, X_i, i = 1, ..., N
- T_i is a transformed database view of X_i and includes the (possibly propagated) class label attribute.
- H_i a classifier for table T_i appropriately chosen based on the characteristics of the table data.
- D_i is the number of dependent database tables of T_i as per the Join Graph, i.e., D_i = {T_j : edge < T_i, T_j > ∈ E}
- The target attribute takes K distinct class labels, C_k, k = 1, ...K
- P(C_k) represents the prior probability for class label C_k over the aggregated/fused dataset.
- P(w) represents the class priors obtained from a single transformed table, T_i.
- P(C_k|T_i) is the posterior probability of class label C_k just based on the data attributes of T_i got by using classifier H_i.

#### 3.1.2 Algorithm NBSplit-Train:

**Input:** Database X = {X_i, i = 1, ..., N}

**Output:** Classifiers H_i ; Class Priors P(C_k) ; P(w)

**Steps:**

1. Read database schema of base tables in X and capture the dependencies in an RDF file which the application expert can edit.
2. Read the edited RDF file and create the Join Graph < V, E, W > for the application
3. T_i = X_i, where X_i ∈ V is the vertex corresponding to the target table
4. Starting at vertex X_i, determine a valid edge order, E' ⊆ E with a breadth first search on the Join Graph.
5. For every edge, \((X_p, X_q) \in E'\) with label \(<l_p, l_q>\):
   Create \(T_q\), an extended view of \(X_q\), by joining Tables \(T_p\) and \(X_q\) with clause \(T_p.l_p = X_q.l_q\).

6. Compute priors from fusion table, \(P_{C_k}, k = 1, \ldots, K\).

7. Training: For each table view, \(T_i, i = 1, \ldots, N\)
   - Compute priors, \(P_{C_k}, k = 1, \ldots, K\).
   - Build a suitable classifier \(H_i\) from the training set \(T_i\) (classical training phase).

In step 1 and 2, an initial RDF model of the Join Graph is created by reading the meta-data of the database, which can be edited by experts to add more semantics. In step 3 and 4, the order of class label propagation is determined. Step 6 optionally computes the class priors from a fusion table or assumes uniform priors. Step 7 is the training step for the first level classification. Here, different classification algorithms can be learnt for different tables based on the attribute distribution. The output of the training phase is a set of classifiers and class priors if computed.

3.1.3 Algorithm NBSplit-Test:

**Input:** Database \(X\), Classifiers \(H_i\), Class Prior \(P_{C_k}\), \(P_{C_k}'\), Sorted edges of Join Graph \(E'\).

**Output:** Predicted class label \(L\) for each test sample.

**Steps:**
1. Create a test view \(Q\), of the target table, \(X\), to extract the required test record.
2. For every edge \((X_p, X_q) \in E'\) with label \(<l_p, l_q>\):
   Create \(Q_p\), a filtered view of \(X_q\), using a JOIN of \(Q_p\) and \(X_q\) with join clause \(Q_p.l_p = X_q.l_q\).
3. For each test record in table, \(Q_p\), \(i = 1, \ldots, N\).
   Compute table-level posterior probabilities, \(P(C_k|T_i)\) for each class \(k = 1, \ldots, K\), using the corresponding earlier trained classifier, \(H_i\).
4. Now traverse the Join Graph in depth first order to determine a table order for aggregation, \(V'\).
5. For each node \(X_j \in V'\), recursively compute scaled posterior using the following product over itself and its dependent tables, \(D_i\).
   \[P(C_k|T_j) = \frac{(p_{C_k} \prod P(C_k|T_i))}{\prod p_{C_k}}\]
6. Classify the test sample to belong to class \(L\) when the target table is reached in the recursion. \(L\) is given by the following equation:
   \[L = \text{argmax}_k \left\{ \frac{(P_{C_k} \prod P(C_k|T_i))}{\prod p_{C_k}} \right\}\]

3.2 Proof by Induction

We now give an abridged proof for the aggregation equation used in steps 5 and 6 of the algorithm NBSplit-Test, assuming binary classification. The proof is based on induction over depth of the Join Graph, \(d\). We would like to prove the decision rule Decide \(C_1\) if

\[
\frac{P(C_1|T_r)}{P(C_1|T_r)} \geq \frac{P(C_2|T_r)}{P(C_2|T_r)}
\]

(1)

The basis condition \((d = 0)\) trivially reduces to Bayes Rule. This occurs when the number of nodes in the Join Graph is just one (single table).

Now consider an intermediate node, \(R\) at depth \(d\) and with \(r\) successors. Without loss of generality, figure 2 for our running example could represent one such intermediate node \(R\) (table Researcher) which is dependent on node \(J\) (Publications table) and \(D\) (Division). So by induction hypothesis (the decision rule holds for table \(D\) and \(J\)), there exists a classifier \(H_k\) that can classify a test sample distributed across \(D\) and its descendants based on rule (1). This classifier will give the same decision as that given by fusing all attributes of \(D\) and those of its descendants. Similarly for table \(J\) and its descendants. Specifically, this could be Naive Bayes. So, we get

\[P(C_1|X_R) = \frac{P(r_1|C_1)P(r_2|C_1)\ldots P(r_j|C_1)}{\sum_{c} P(X_R|C_c)P_{C_c}}\]

For proving the induction rule, we apply Naive Bayes rule across columns of a hypothetical table \((X)\) got by fusing columns from tables \(R\), \(D\), and \(J\), and use the above equation to simplify to the required form

- \(P(X|C_1) = \prod_{i=1}^{r} (P(r_i|C_1)P(r_2|C_1)\ldots P(r_j|C_1))\)
- \(P(D_1|C_1) \ldots P(D_2|C_1)P(J_1|C_1) \ldots P(J_2|C_1)\)
- \(P(X|C_1) = \frac{P(C_1|X_R)P(C_1|X_P)P(C_1|X_D)\notin Q}{P(C_1|X_R)P(C_1|X_P)P(C_1|X_D)\notin Q}\)

Proof of (1) follows from (2) by Bayes Rule.

4 Implementation and Results

Our implementation is in Java and uses the Weka 3.5.7 ML package for the classifiers and the Jena library to model the Join Graph in RDF. The test databases are on PostgreSQL 8.2. We evaluated the proposed algorithm on three types of datasets (a) TPCH database benchmark to show practical applicability (b) UCI (Lung Cancer) dataset to prove its correctness (c) a multi-relational dataset (the financial database from PKDD Cup 99) to compare with prior relational classification techniques.
4.1 TPCH decision-support benchmark

TPCH is a database application for Customer Relationship Management. We compared the training time of our Split-NB algorithm with Naive Bayes on fused tables. Our approach shows a great improvement due to reduced number of training samples as seen in Table 1. Further, use of RDF to specify the data of interest (in step 1 of §3.1.2) was very useful. Out of the 8 tables, only 3 tables were needed (CUSTOMER, ORDERS and NATION) to predict whether a given customer was likely to buy an household item. Also, the ORDERS table originally had 9 attributes of which we found that only four attributes were meaningful for the classification. Our solution is fully automated, just a change in the RDF file is sufficient to run the classification task on a new database.

4.2 Financial Database from PKDD Cup

We compared our algorithm with another multi-relational classifier, Cross Mine [2], over a well-known relational dataset. As shown in Table 2, our technique performs extremely well. Here, we tried multiple classifiers on individual tables and chose the best classifier (SVM). Clearly, just using root table for classification was insufficient. We also found that elimination of non-contributing tables (Transaction table) gives much better results - proving the benefit of using hidden semantics of schema design. Similarly, inclusion of Account and Order tables increased the accuracy results.

4.3 Lung-Cancer Dataset from UCI

Finally, we confirmed the correctness and accuracy of our algorithm using a standard single-table dataset. We artificially loaded the 58 attributes of the Lung Cancer dataset into 7 tables in Star schema. We executed our algorithm over this distributed data and compared the results with fused dataset for multiple single-table classifiers. The results are shown in the Radar graph of Figure 3. As seen, the predictions got by both these procedures were exactly the same when we used Naive Bayes classifier at local level (both assuming Naive Bayes condition). While this exercise is not meant to compare the classifiers, it proves the effectiveness and correctness of our localized aggregation technique for diverse table-level classifiers.

5 Conclusions

We proposed a practical, 2-phase meta-classification algorithm for relational databases. Our main premise is that since most databases are well designed by experts, we need to use this implicit semantics. We proposed a new recursive aggregation technique to collate heterogeneous classifiers applied at individual tables. We have demonstrated obvious improvement in classification training time for TPCH and proved that there is no loss of accuracy due to aggregation. We also showed the benefits of our approach over prior efforts. As next step, we would like to extend this technique to include auto-selection of the right classifier at the table level. Secondly, classification accuracy can be improved by eliminating some non-contributing tables. For this, we plan to associate an entropy metric with individual database tables and select the right subgraph from the Join Graph that minimizes the information loss.

References

1. H Liu, X Yin, J Han, “An Efficient Multi-relational Naive Bayesian Classifier Based on Semantic Relationship Graph”, ACM MRDM.
2. X Yin, J Han, et al, “CrossMine: Efficient Classification Across Multiple Database Relations”, ICDE 04, 2004
3. HGuo, H L Victor, “Mining Relational Databases with Multi-View Learning”, ACM MRDM 05.
5. P A. Flach, N Lachiche, “Naïve Bayesian Classification of Structured Data”, Machine Learning, 2004