# Advantage of Integration in Big Data: Feature Generation in Multi-Relational Databases for Imbalanced Learning

Farrukh Ahmed*, Michele Samorani†, Colin Bellinger*, Osmar R. Zaïane*
*Department of Computing Science, University of Alberta, Canada
†Leavey School of Business, Santa Clara University, USA
Email: *{farrukh1, cbelling, zaiane}@ualberta.ca, †{msamorani}@scu.edu

***Abstract*—Most real world applications comprise databases having multiple tables. It becomes further complicated in the realm of Big Data where related information is spread over different data repositories. However, data mining techniques are usually applied on a single flat table. This work focuses on generating a mining table by aggregating information from multiple local tables and external data sources and automatically generating potentially discriminant features. It extends data aggregation techniques by navigating paths where a single table is traversed multiple times. Such paths are not considered by existing techniques, which results in the loss of several attributes. Our framework also prevents leakage of the class information by avoiding features built after the knowledge of the class label. Experiments are performed on transactional data of a U.S. consumer electronics retailer to predict causes of product returns. In addition, we augmented the dataset with Suppliers information and Reviews to show the value of data integration. The results show that our technique improves classification accuracy and generates discriminant features that mitigate the impact of class imbalance.**

***Keywords*-Data integration; Feature construction; Classification; Class imbalance**

## I. Introduction

An ongoing and significant limitation in the realm of data mining is the fact that the majority of data mining and machine learning algorithms work with a single table, whilst the potential knowledge is stored in data spread across many tables, databases and even in raw formats such as tweets, blogs and online comments. For example, Netflix employs a recommendation system that utilizes extensive data to suggest movies to users according to their tastes [1]. These recommendations can be further improved by incorporating publicly available data from sources like IMDB [2]. Nonetheless, very few tools exist to take advantage of data integration in Big Data analytics.

Before harnessing the richness of the data mining techniques, the relevant attributes of the data must be accumulated in a single table. It is common practice for this to be done manually, often with the support of domain experts. This approach is both costly and time consuming. Moreover, human analysts are likely to have their attribute generation process implicitly, or explicitly, limited to the domain standards. As a result, they are likely to omit features that are unknowingly of great relevance to the problem.

Instead of handcrafted features, we argue that it is of great benefit to devise means of automatically generating features from dispersed data. In addition to speed and efficiency, it offers the potential of generating informative features that simplify the learning task. From the classification perspective, the generation of new features can increase the separability of the classes. Our results suggest that this has the potential to mitigate the impact of class imbalance.

To this end, we introduce the Generating Attributes with Rolled Paths algorithm (GARP) that automatically generates attributes from all the data sources and structures them in a single table. In addition, we empirically evaluate the AUC performance of the algorithm in the context of class imbalance. We study these questions on the transactional dataset from a large US electronics retailer [3].

## II. Related Work

### A. Multi-relational Data Mining

Multi-relational Data Mining (MRDM) techniques help in finding patterns and applying learning techniques on databases which store information in multiple tables. The existing approaches to feature discovery and classification in MRDM can be divided into two categories. The first approach is based on Inductive Logic Programming (ILP) to extend learning techniques so that they can handle relational data [4]. On the other hand, Propositionalization focuses on aggregating data from multiple tables into a single table so that traditional learning techniques can be applied [5].

ILP treats tables as entities comprising of facts. For example, Customer(Bob, M, 25) and Product(Laptop, HP, $900) represent facts [4]. The approach works by using the induction engine to derive rules, based on First Order Logic, for the prediction of the class. It is generally limited to the binary existence quantifier; it can only tell if the row being predicted has a related instance in the other table which satisfies the predicate. For example, *pastReturn(Customer)* tells whether the customer has returned a purchase, however, it cannot determine the number of purchases returned by a customer. Additionally, the learning phase is tightly coupled with the attribute generation phase which makes it incompatible with most of the existing learning techniques.

Propositionalization approaches divide the relational learning task into two steps. The first step is to navigate associated tables in a database and summarize information into a single table. The second step is to apply traditional learning techniques on the table generated in the first step.

The Polka algorithm is one of the initial work related to the aggregation of complex information from multiple tables into a single table [6]. The authors introduce aggregate functions in the propositionalization step to summarize information for 1-to-many relationships. This can generate deeper patterns compared to the ILP approach. It also generates some specific patterns by adding refinements to the count aggregation operator, such as the number of purchases made by customers where the price of a product is above $100. However, the Polka algorithm does not allow refinements with other operators like average, sum, min and max.

The ACORA framework [7] for Automatic Construction of Relational Attributes allows refinements with all types of aggregation operators. Additionally, it introduces novel distribution-based aggregations which perform well with high dimensional categorical attributes. The main idea is to construct features that utilize the information provided by object identifiers that are usually dropped to build more generalized models. Although ACORA introduces novel distribution-based aggregators, it misses several important features which can capture the past information. This limitation arises because ACORA avoids generating attributes which need a table to be joined multiple times (like Purchase $\bowtie$ Customer $\bowtie$ Purchase).

The randomized propositionalization approach [8] does not restrict the generation of attributes with such table joins. However, allowing these attributes in the proposed approach would result in leakage of the class information in the training set [9]. Our approach, GARP, generates attributes by joining a table multiple times without using any future information. Our Dataconda framework demo briefly introduces the concept of rolled paths [10].

## B. Class Imbalance

Class imbalance occurs when the training instances from one class are significantly outnumbered by the training instances of the other class(es). In this research, we consider the imbalance from a binary perspective. Class imbalance occurs in a wide variety of important domains, from radiation and oil spill to image and text classification. It has been shown to have a negative impact on the performance of induced classifiers [11]. Given the frequency of imbalanced learning problems and the possibility of negative impacts, it has been recognized as one of the ten most challenging problems in data mining research [12].

The class imbalance problem has its greatest impact when the target domain is highly complex. This may result from a variety of factors, such as distribution form and modality of the classes and the degree to which they overlap in the dataspace [11]. We are particularly interested in class overlap.

Class overlap occurs when the features of the data that are provided to model the data do not serve to differentiate classes. When there is a significant overlap in the data distribution, and one class is underrepresented in the training set, the classifier is forced to attempt a complex decision regarding the position of the decision boundary without sufficient information. The few borderline training instances in the minority class can easily be overshadowed by the many majority class training instances. As a result, the decision boundary is placed inside the minority space, causing minority instances to be misclassified.

Given the prominence of class imbalance, many methods have been offered as means of mitigating its negative impact. These include sampling methods, such as oversampling and random undersampling (RUS), which balance the training distribution by replicating minority instances or removing majority instances [13]. In order to avoid discarding informative instances from the majority class, heuristic-based undersampling methods have been proposed [14]. Cost-based methods have been shown to be theoretically related to random undersampling in some cases [15]. They influence the induction process to reduce the prediction bias by increasing the relative misclassification cost for the minority class during training. Finally, synthetic oversampling generates new training instances for the minority class based on those available in the training set [16], [17].

Although each of these methods has been shown to reduce the impact of class imbalance, the existing methods do not offer an absolute solution. The only guaranteed method to prevent the negative impact of class imbalance is to utilize discriminative features so that an accurate decision boundary can easily be induced. This is not always physically possible, and moreover, feature engineering can be a costly and time consuming process. When it is possible, however, acquiring good features is likely the best way to dealing with class imbalance, or at least the appropriate starting point.

Our proposed method, GARP, provides an automated process to generate features, rather than engineer them, in big data domains. Our empirical results show that these features enable the baseline classifiers to outperform SMOTE and random undersampling.

## III. Generating Attributes with Rolled Paths

Our objective is to construct a flat mining table by adding attributes generated by aggregating information from related tables within the database and external data sources as well. Figure 1 shows a relational database with multiple tables and external data sources. Purchase is the target table where all the information needs to be aggregated before applying learning techniques. There are multiple levels on which related information of the Purchase table is distributed.

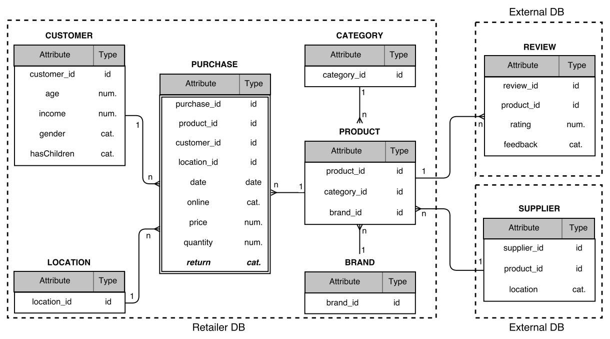

Figure 1: Retail Database with external sources of Suppliers and Reviews

In real applications, databases can have tens, if not hundreds, of tables which can exist on even deeper levels. In addition, multiple external data sources, such as Review and Supplier in Figure 1, can be available with similar complex association levels. The graph in Figure 2 shows paths up to depth '3', to aggregate information from other tables to the Purchase table within the retail store database.

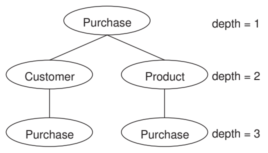

Figure 2: Paths up to depth 3 within the retail database

The paths available at depth 3 are examples of rolled paths as they navigate the Purchase table twice. GARP avoids including duplicate information on rolled paths by performing step-wise joins. For path Purchase — Customer — Purchase, the aggregation operator is first applied for Customer $\bowtie$ Purchase (e.g. Total amount spent by a customer in the past) which results in a single row for each customer. Then, Purchase $\bowtie$ Aggregation(Customer $\bowtie$ Purchase) adds this attribute to the Purchase table. In this way, GARP generates attributes carrying past information. Our method aggregates information from external data sources in a similar manner.

Leakage occurs when attributes are generated in the presence of the class label [9]. This can lead to an overestimation of the classification accuracy. It is not appropriate to use the return information of the purchase; however, including the past information of returns is legitimate and provides useful insights. For example, at the time of predicting a return, a customer's return history before that point can be used for prediction. We avoid leakage of the class information by considering dates to avoid future information.

## IV. Methodology

### A. Preliminaries

Our running example uses the database structure of our retail store example with external data sources in Figure 1. The features aggregated from the database are to be attached to the $Purchase$ table to form the flat mining table. It includes a binary class attribute called $return$ that indicates if the purchase was returned or not. Each table in the database is characterized by a name and a set of attributes. These attributes have a type that is used to identify the applicable aggregations and refinements. While aggregating information from the related tables to the mining table, we deal with $1-to-many$ and $many-to-1$ relationships.

The external tables in our example are $Supplier$ and $Review$. Customers often review products on social media and other forums which provide useful feedback and information about the acceptance of products. It is beneficial to use this information to predict product returns. For the purpose of demonstration, we assume that reviews have been extracted from a public forum where users have rated products with stars and left comments about them. Star ratings are used directly as a numeric attribute and the comments are analyzed via an opinion mining technique to produce a categorical value of positive, negative or neutral. Additionally, Product related information available on the supplier's website can be extracted and used along with the customer transactions data to identify causes of returns that might not be obvious without this information; for example, returns due to the use of specific material or manufacturing location may be discovered.

### B. Path Generation

We first generate the possible paths to find out the potential attributes in the database. Each path starts from the target table which contains the class attribute and ends at a table based on a specified depth level. The depth level reflects the number of tables used to aggregate information. It also determines the complexity of generated attributes. The aggregation starts from the table at the end of the path and brings the information to the target table.

For the retail database presented in Figure 1, we start with the Purchase table at depth = 1. To find the paths that exist at depth = 2, we look at the tables associated with the Purchase table (Location, Customer and Product). For depth = 3, we add more paths by joining the related tables for each of the paths generated at depth = 2. In this way, we generate paths by attaching the tables up to a specified depth level. However, we restrict subpaths with a structure 'A—B—A', where the relationship between A to B is 1-to-many and the foreign key identifier joining A—B and B—A is same, as such paths cannot result in any additional information.

### C. Attribute Generation

After completing the path generation process, we aggregate the information available at each path and add new attributes to the target table. Aggregation starts from the end of the path and information is rolled back to the target table. Consider a path $T_0$ — $T_1$ — $T_2$ — ... — $T_{l-1}$, where

$l$ is the length of the path. For $T_i$ — $T_{i+1}$ with $i$ ranging from *l-2* to *0*, an attribute is added to $T_i$ by aggregating information from $T_{i+1}$. The generated attribute is virtual to these intermediate tables i.e. not materialized in these tables and appears as an attribute directly in the target table $T_0$.

**Algorithm 1** Generating Attributes with Rolled Paths

```
Input: Target table $T_0$, max depth level $L$
Output: Mining table
1: $P$ = Paths generated up to depth level $L$
2: for $l$ = 1 to $L$ do
3:   for each path $p$ = ($T_0$, ..., $T_{l-1}$) in $P$ do
4:     for $i$ = $l$ – 2 down to 0 do
5:       attributes[$T_{i+1}$] = non-id attributes $a_1..a_n$
6:       virtual_attributes[$T_{i+1}$] = non-id attributes $v_1..v_n$
7:       if virtual_attributes[$T_{i+1}$] is not empty then
8:         candidate_attributes = virtual_attributes[$T_{i+1}$]
9:       else
10:        candidate_attributes = attributes[$T_{i+1}$]
11:      if $T_i$ — $T_{i+1}$ is many-to-1 or 1-to-1 then
12:        for each $a_j$ in candidate_attributes[$T_{i+1}$] do
13:          virtual_attributes[$T_i$].add($a_j$)
14:      else if $T_i$ — $T_{i+1}$ is 1-to-many then
15:        for each $a_j$ in attributes[$T_{i+1}$] + virtual_attributes[$T_{i+1}$] do
16:          for each $Agg$ compatible with $a_j$ do
17:            /* Algorithm 2 */
18:            $V_i$ = Aggregation($Agg$, $a_j$, $T_i$, $T_{i+1}$)
19:            virtual_attributes[$T_i$] += $V_i$
20: Mining table = attributes[$T_0$] + virtual_attributes[$T_0$]
```

The process of generating an attribute differs on the basis of the relationship between $T_i$ and $T_{i+1}$. For a many-to-1 or 1-to-1 relationship between $T_i$ — $T_{i+1}$, an attribute from the table $T_{i+1}$ can be directly attached to the table $T_i$. For example, Purchase — Customer has a many-to-1 relationship, so we can directly add attributes from Customer to Purchase. However, if $T_{i+1}$ is not the last table in the path then only virtual attributes from $T_{i+1}$ should be attached to $T_i$ because attributes that belong to $T_{i+1}$ are already attached to the target table on a shorter path.

For a 1-to-many relationship between $T_i$ — $T_{i+1}$, multiple rows from $T_{i+1}$ are associated with each row in $T_i$. So for each attribute in $T_{i+1}$, we need to apply an aggregation operator which can summarize the information from multiple rows related to a single row in $T_i$. We explain the aggregation process in greater detail in the following subsection.

*1) Aggregations and Refinements:* It is essential to apply aggregation operators for 1-to-many relationships to avoid information loss in the attribute generation process. In addition, aggregation operators can be complemented with refinements to generate specific patterns. The procedure to apply aggregation and refinements is shown in Algorithm 2.

**Algorithm 2** Aggregation Procedure

```
Input: Aggregation $Agg$, attribute $a_j$, Tables $T_i$ and $T_{i+1}$
Output: Returns the list of virtual attributes, based on $a_j$
1: $V_i$ = []
2: if virtual_attributes[$T_{i+1}$] is not empty then
3:   if $a_j$ is in virtual_attributes[$T_{i+1}$] then
4:     candidate_ref_attributes = attributes[$T_{i+1}$]
5:   else
6:     candidate_ref_attributes = virtual_attributes[$T_{i+1}$]
7: else
8:   candidate_ref_attributes = attributes[$T_{i+1}$]
9: if both $T_0$ and $T_{i+1}$ have a date then
10:  date refinement = $T_{i+1}.date < T_0.date$
11: /* Aggregation without refinement */
12: if virtual_attributes[$T_{i+1}$] is empty then
13:   for each $x$ in $T_i$ do
14:     $R$ = records in $T_{i+1}$ associated to $x$
15:     $v_i$(x) = $Agg(R.a_j)$ from $R$
        [and $R.date < T_0.date$]
16:   $V_i$.add($v_i$)
17: /* Aggregation with refinement */
18: for each $a_k$ in candidate_ref_attributes do
19:   for each $Ref$ compatible with $a_k$ do
20:     for each $c$ in refinement_values do
21:       for each $x$ in $T_i$ do
22:         $R$ = records in $T_{i+1}$ associated to $x$
23:         $v_i$(x) = $Agg(R.a_j)$ from $R$ where $R.a_k$ $Ref$ $c$
            [and $R.date < T_0.date$]
24:       $V_i$.add($v_i$)
25: return $V_i$
```

The aggregation process can be understood by the following query:

SELECT Agg($T_2.a_j$) FROM $T_1$
JOIN $T_2$ on $T_1$.pk = $T_2$.fk GROUP BY $T_1$.pk

In Customer — Purchase relationship, the query can have:
$a_j$ = [online, price, quantity, return]
Agg = [Average, Sum, Min, Max]; for numeric attributes
= [Count, Count Distinct]; for categorical attributes

GARP applies all compatible aggregations with each of the attributes. In addition, it is possible to define custom aggregation functions that, given a list of values, return a single value. The following attributes are examples of aggregations applied on the path Customer — Purchase:

- Max(price): Maximum amount spent on a purchase
- Avg(return): Average number of returned purchases

The attributes shown above are generated by aggregation operators without any refinements. The refinements can be used to filter data in order to find specific patterns. The refinement can be introduced as a where clause in the query:

WHERE $a_k$ Ref c

The suitable Ref operators based on the type of $a_k$ are:

Ref = [$>$, $\leq$, =, $\neq$]; for numeric

= [=, $\neq$]; for categorical

The two possible types of refinements based on the value of 'c', in the refinement clause above are:

- toValue refinements; where c is a constant
- comparison refinements; where c is an attribute

The term 'c' for toValue refinements can be substituted with all possible values of the attribute or values determined by discretization techniques like binning. All values should be used for categorical attributes and binning should be used for numerical attributes. The following examples represent toValue refinements for the aggregation Avg($P_2$.return) on the path Purchase (P) — Customer — Purchase ($P_2$):

- online = 1: Return history for online purchases
- price $<$ \$500: Return history for purchases below \$500

On the other hand, comparison refinements can be used to compare an attribute with another attribute of the same type and dimension. An example attribute generated with a comparison refinement on the path Purchase (P) — Customer — Purchase ($P_2$) is Avg($P_2$.return) where P.online = $P_2$.online.

## V. Scaling the Attribute Generation Process

The attribute generation phase results in a large number of attributes based on different aggregations and refinements. This extensive process becomes very time-consuming for large datasets. In order to scale the algorithm, we analyzed the algorithm to reduce its time complexity. We focused on the aggregation and refinements step of the algorithm as it is the most expensive stage in the whole process.

In Figure 3, we present the internal details of aggregation and refinement steps. The Agg & Ref block provides details of aggregation on the path $T_i$ — $T_{i+1}$ with a 1-to-many relationship. The aggregation step needs to structure data based on the joined table before applying the aggregation operator to summarize data. The inclusion of the refinement adds one more step to filter this data before the aggregation.

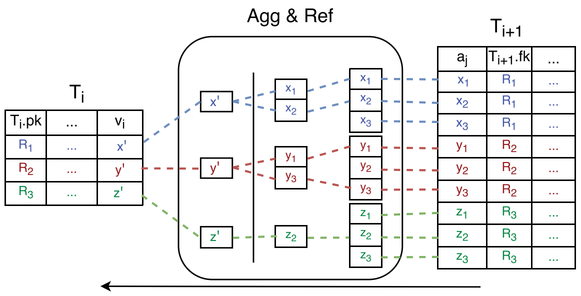

Figure 3: Aggregating attribute $v_i$ from $T_{i+1}$ to $T_i$ on path ($T_i$ — $T_{i+1}$) with Refinement

Consider the path Customer — Purchase to aggregate information related to the purchasing history of each customer. The attributes in the Purchase table are:

attributes(4) = [online, price, quantity, return]

The aggregation operators used are:

aggregation operators(4) = [avg, sum, min, max]

Thus, there are 16 possible aggregations. Assume that refinement values and operators for each attribute are:

based on all possible values; with operators = [=]:

online(2) = [0, 1], return(2) = [0, 1]

based on 5 equal-width bins; with operators = [<]:

price(4) = [$p_1$, $p_2$, $p_3$, $p_4$], quantity(4) = [$q_1$, $q_2$, $q_3$, $q_4$]

Based on this, there are 12 possible refinements (refs):

refs $= \sum_{i=1}^{n}$ Number of values$_i$ $*$ Number of operators$_i$.

The total number of possible attributes are:

attributes = aggregations * ( refinements + 1)

= 16 * (12 + 1) = 208

This means that the aggregation process has to go through the structuring, filtering and aggregation phases 208 times. However, we can make this process efficient by separating the structuring and filtering phases from the aggregation phase as shown in Figure 4. The temporary result generated by the pre-processing step can be reused to compute several attributes, that rely on the same structured and filtered data. For example, all the (16) aggregations are performed in conjunction with the refinement 'price $<$ $p_1$'. A temporary result can be generated for the refinement 'price $<$ $p_1$' and all these aggregations can be applied together. The structuring and filtering steps would be reduced to 13 as:

Total preprocessing steps = 1 * (Total refinements + 1)

Hence, in the Customer — Purchase relationship, the number of times this structuring has to be performed is reduced from 208 to 13. This helps in reducing the time complexity significantly with an increasing data load.

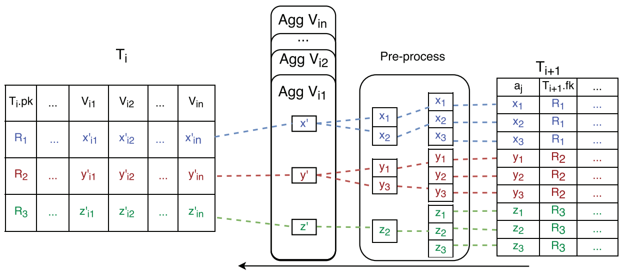

Figure 4: Aggregating $v_{i1}$, $v_{i2}$, ... $v_{in}$ from $T_{i+1}$ to $T_i$ on path ($T_i$ — $T_{i+1}$) with Refinement

## VI. Attribute Selection

The attribute generation process results in a huge feature space for the mining table. A feature selection technique can help to reduce the dimensionality by finding the best predictors. For example, to find the causes of returns in the retail database, feature selection helps to limit the attributes to only real causes of returns.

We use lasso (least absolute shrinkage and selection operator) [18] to select the features that are highly related to the class. Lasso works by applying regression and forcing the coefficients of attributes to be less than a specified value. In this process, coefficients of some attributes are set to zero and these attributes can be removed from the feature space. Hence, Lasso results in sparse models which are interpretable like subset selection and widely used in the sciences and social sciences [19].

The benefit of using lasso is that it selects attributes that are highly correlated with the class but do not have a high correlation among themselves. In our feature generation process, several attributes are generated that are very similar to each other. For example, our technique generates an attribute 'average of return for a customer' with several refinements like 'price $<$ \$500', 'price $<$ \$1000', 'price $<$ \$1500', etc. As these attributes are highly correlated with each other, lasso will try to pick the best from them. In this way, diverse attributes are selected and different reasons behind product returns can be determined.

## VII. Experimental Setup

These experiments serve to evaluate GARP in terms of three factors: *a*) the relative benefit of rolled paths in the the aggregation of higher level tables in a relational database, *b*) the benefit of aggregating features from external tables, and *c*) the robustness of GARP to the problem of class imbalance in classification of product returns.

### A. Data

We conduct our evaluation on the real-world transactional dataset from Circuit City, which was a large US electronics retailer [3]. It contains around 115,000 purchases of around 20,000 products made by 20,000 customers. The ratio of purchase:customer and purchase:product in the data is approximately 6:1. This does not provide enough representation of the purchasing behaviour of several customers and products. Therefore, we sampled the dataset around products to generate an information-rich subset to evaluate the full potential of our technique.

We divide the dataset into three groups based on the percentage of returns and randomly select products from these groups ensuring that the percentage of returns is similar to the percentage of returns in the complete dataset (around 10%). In this way, we try to select products with varying chances of return. There are three groups having products with: high return percentage ($>$40%), medium return percentage (10-30%) and low percentage($<$10%). Next, we retrieve all the purchases and customers corresponding to these products. Our selection results in 18,182 purchases with 1,868 returns and we take 75% as the training set.

### B. Classifiers and Sampling methods for Class Imbalance

We have selected six classification methods from the Weka machine learning software to apply to the retail dataset [20]. In particular, we apply ripper (JRip), C4.5 (J48), Bayesian network (BN), random forest (RF), support vector machines (SVM) and multilayer perceptron (MLP). These have been selected to capture a wide breadth of learning biases in order to maximize the generality of our results.

In order to evaluate our hypothesis that GARP selects and attaches features to the target table that are helpful for data suffering from the problem of class imbalance, we consider two standard sampling methods, random undersampling and SMOTE. The performance benefits of these methods are compared to those of GARP.

## VIII. Experiments

### A. Scalability

To evaluate the scalability of our GARP technique, we executed the attribute generation procedure by varying the sample size of the Retail database. We generated four samples of the dataset with about 25%, 50%, 75%, and 100% of the available data. All the experiments were conducted on a laptop with Intel Core i7-6700HQ processor (6M Cache, 2.6GHz) and 12GB RAM.

We run our technique up to depth levels 3 and 4. Figure 5 reports the time taken to generate new attributes for the flat mining table. The results presented in Figure 5 show that increasing the size of the database increases the execution time of GARP in a linear fashion at both depth levels of 3 and 4. The depth level 4 takes more time than depth 3, as it involves an additional table in the joins.

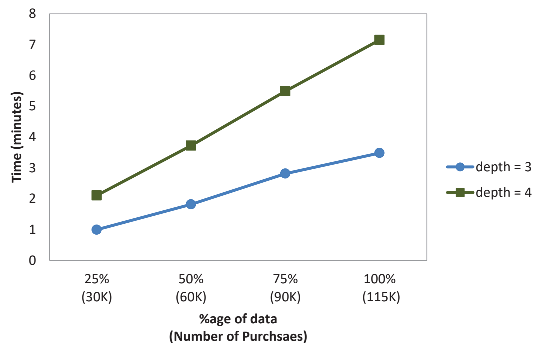

Figure 5: Execution times for GARP at depth 3 and 4 by varying the size of the dataset

### B. Classification Performance

*1) Retail Database Results:* The results in this section serve to demonstrate the relative benefit of aggregating data into the target table from tables that are not directly connected to the target table, and enabling rolled paths. To do this, we report the AUC results produced on just the target table (level 1), the aggregation of the target table with tables directly connected (level 2), and the aggregation, with rolled paths, of data from tables indirectly connected to the target table at level 3 and level 4.

The first four rows of Table I depict the AUC results for each classifier on the retail dataset without the inclusion of the external data sources. The baseline results are shown in the first row. These are the product of the target table without aggregation. The second row shows the AUC results when aggregation is performed without rolled paths. Finally, the third and fourth rows depict the results produced with GARP with rolled paths at levels three and four, respectively. The last two rows report results which include the external data sources, discussed in the subsequent section.

Table I: AUC results produced on the original table and the augmented tables. The results in the rows marked by GARP* utilized the external tables of Supplier and Reviews as well.

| Method (Depth) | J48 | RF | BN | JRIP | SVM | MLP |
|---|---|---|---|---|---|---|
| No Aggregations (1) | 0.51 | 0.58 | 0.61 | 0.50 | 0.50 | 0.64 |
| No Rolled Paths (2) | 0.51 | 0.57 | 0.61 | 0.50 | 0.50 | 0.54 |
| GARP (3) | 0.68 | 0.75 | 0.77 | 0.61 | 0.76 | 0.77 |
| GARP (4) | 0.68 | 0.78 | 0.76 | 0.60 | 0.70 | 0.74 |
| GARP (3)* | 0.70 | 0.80 | 0.83 | 0.61 | 0.87 | 0.82 |
| GARP (4)* | 0.70 | 0.82 | 0.80 | 0.61 | 0.88 | 0.82 |

In each case, the classifiers that are induced on the data aggregated by GARP from the retail database outperform the alternate approaches. Moreover, the increase in performance beyond both the baseline and the aggregation at level 2 without rolled paths is uniformly greater than 0.1 AUC. This represents good improvement. RF produced the best overall AUC result of 0.78 using the data aggregated by GARP (4). BN and MLP produced similarly good improvements in the AUC (0.77, 0.77) with data aggregated by GARP (3).

*2) Retail Database + External Tables Results:* Using rolled paths and aggregating data into the target table from less directly connected tables unveils a wealth of information to the learner. In the previous section, we demonstrated that this can lead to a large improvement in the AUC performance of the induced classifier. As we have previously stated, there is arguably even more potential in outside data sources. This potential is demonstrated in the bottom two rows of Table I.

The entries GARP (3)* and GARP (4)* in this table show the AUC performance achieved at the respective levels of the retail database when the outside sources, Review and Supplier, are also included. In each case, with the exception of JRIP, the AUC results improve when the outside data sources are included. JRIP, however, maintains a similar improvement over the baseline with GARP and GARP*.

The best overall performance reported in the complete table is, indeed, produced when the outside data sources are included in the aggregation. This is a natural but important result. If there is more good knowledge that could be included, then doing so is of benefit to the classifier.

The classifier that benefits the most here is SVM. Interestingly, although it perviously only improved slightly, the availability of the external data elevates its performance beyond the other classifiers. This shows that the added features may have different impacts on the classifiers. Moreover, the fact that, in general, all of the classifiers improve after aggregation with GARP provides strong evidence that GARP is adding good discriminative features to the target table.

*3) Class Imbalance:* A potential benefit of our proposed algorithm is that the process of aggregation discovers new features that increase the separability in the classification problem. When more discriminative features are added to the target table, it becomes easier for the classifier to induce a decision boundary that accurately separates the classes. In this section, we report the AUC results of the six classifiers using the aggregated data produced by GARP when deployed in conjunction with RUS and SMOTE to deal with the class imbalance. Given that the retail dataset is imbalanced, our objective is to observe the impact of GARP on the classifiers relative to the methods that were specifically designed to mitigate the impact of class imbalance.

Table II summarizes our results with respect to the problem of class imbalance. The top row specifies the performance of the target table without any aggregation; once again, this is the baseline. The second and third rows depict the AUC results produced on the target table aggregated by GARP at levels three and four. These aggregations include the external data sources. The columns specify the classifier along with the method used to resolve the class imbalance in the retail dataset. The sub-column marked by '-' indicates that no correction was applied before the induction of the classifier, RUS indicates that random undersampling was applied and SMT indicates that SMOTE was applied.

The results show that applying RUS prior to classifier induction on the unaggregated table leads to a subtle increase in the AUC for three of the six classifiers and applying SMOTE leads to a small improvement on four of the six classifiers. In no case, however, does the improvement caused by be the sampling methods ameliorate the results to the same degree as simply applying GARP at levels three or four. In addition, the table includes the results of applying GARP followed by the sampling methods prior to the induction of the classifiers. The results show that application of GARP alone achieves the highest AUC performance. When sampling is applied in conjunction with GARP the results are unchanged or get worse; for example, the latter is the case for BN and SVM.

Whilst too narrow to generalize from, and we recognize that these results do not include a comprehensive list of the methods available to manage the class imbalance, we see them as providing some initial evidence that GARP can manage the class imbalance problem. Indeed, we see that on this data, GARP leads to improvements that are not matched by the imbalanced classification methods tested above. Our future work will aim to generalize and solidify this finding via an empirical study on a wider variety of datasets.

Table II: Demonstration of the robustness of GARP to class imbalance.

| Method (Depth) | J48 | | | RF | | | BN | | | JRIP | | | SVM | | | MLP | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | - | RUS | SMT | - | RUS | SMT | - | RUS | SMT | - | RUS | SMT | - | RUS | SMT | - | RUS | SMT |
| No Aggregations | 0.51 | 0.57 | 0.57 | 0.58 | 0.57 | 0.58 | 0.61 | 0.59 | 0.62 | 0.50 | 0.57 | 0.58 | 0.50 | 0.51 | 0.52 | 0.64 | 0.64 | 0.64 |
| GARP (3)* | 0.70 | 0.69 | 0.69 | 0.80 | 0.76 | 0.75 | 0.83 | 0.81 | 0.56 | 0.61 | 0.59 | 0.60 | 0.87 | 0.70 | 0.78 | 0.82 | 0.75 | 0.82 |
| GARP (4)* | 0.70 | 0.69 | 0.67 | 0.82 | 0.78 | 0.78 | 0.80 | 0.80 | 0.56 | 0.61 | 0.61 | 0.60 | 0.88 | 0.70 | 0.79 | 0.82 | 0.75 | 0.82 |

## IX. Conclusion and Future Work

We proposed a method, GARP, that automatically generates attributes for a mining table from the entire database as well as external data sources. GARP generates attributes, containing useful information from the past, missed by existing techniques.

Our experiments on the data from a large U.S. consumer electronic retailer Circuit City suggest that our method can improve classification accuracy and mitigate the effect of class imbalance by generating discriminant features. In addition, the benefits of data integration are obvious in experiments with the inclusion of Supplier and Review data.

Our methodology has room for improvement. GARP generates a large number of attributes by exploring all possible paths in the database. This extensive process also generates some attributes that are not very useful. A future study can focus on pruning some less beneficial paths based on some heuristics to reduce the overall computation overhead of generating attributes. Another study can be focused on parallelizing the attribute generation process. Our scalability analysis shows that the time complexity of our approach is linear. The computation time can be further reduced with an efficient approach to distribute the work on several machines.

## References

[1] X. Amatriain, "Mining large streams of user data for personalized recommendations," *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 37–48, 2012.

[2] J. Lees-Miller, F. Anderson, B. Hoehn, and R. Greiner, "Does wikipedia information help netflix predictions?" in *Seventh International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2008, pp. 337–343.

[3] J. Ni, S. A. Neslin, and B. Sun, "Database submission-the isms durable goods data sets," *Marketing Science*, vol. 31, no. 6, pp. 1008–1013, 2012.

[4] N. Lavrac and S. Dzeroski, "Inductive logic programming," *WLP*, pp. 146–160, 1994.

[5] S. Kramer, N. Lavrač, and P. Flach, *Propositionalization approaches to relational data mining*. Springer, 2001.

[6] A. J. Knobbe, M. De Haas, and A. Siebes, "Propositionalisation and aggregates," in *Principles of Data Mining and Knowledge Discovery*. Springer, 2001, pp. 277–288.

[7] C. Perlich and F. Provost, "Distribution-based aggregation for relational learning with identifier attributes," *Machine Learning*, vol. 62, no. 1-2, pp. 65–105, 2006.

[8] M. Samorani, M. Laguna, R. K. DeLisle, and D. C. Weaver, "A randomized exhaustive propositionalization approach for molecule classification," *INFORMS Journal on Computing*, vol. 23, no. 3, pp. 331–345, 2011.

[9] S. Rosset, C. Perlich, G. Świrszcz, P. Melville, and Y. Liu, "Medical data mining: insights from winning two competitions," *Data Mining and Knowledge Discovery*, vol. 20, no. 3, pp. 439–468, 2010.

[10] M. Samorani, "Automatically generate a flat mining table with dataconda," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, Nov 2015, pp. 1644–1647.

[11] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[12] Q. Yang and X. Wu, "10 challenging problems in data mining research," *International Journal of Information Technology & Decision Making*, vol. 5, no. 4, pp. 597–604, 2006.

[13] B. C. Wallace, K. Small, C. E. Brodley, and T. A. Trikalinos, "Class imbalance, redux," in *11th International Conference on Data Mining*. IEEE, 2011, pp. 754–763.

[14] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One-sided selection," in *14th International Conference on Machine Learning*, 1997.

[15] K. McCarthy, B. Zabar, and G. Weiss, "Does cost-sensitive learning beat sampling for classifying rare classes?" in *1st international workshop on Utility-based data mining*, 2005, pp. 69–77.

[16] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[17] C. Bellinger, "Beyond the boundaries of smote: A framework for manifold-based synthetic oversampling," Ph.D. dissertation, University of Ottawa, 2016.

[18] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[19] P. Zhao and B. Yu, "On model selection consistency of lasso," *Journal of Machine Learning Research*, vol. 7, no. Nov, pp. 2541–2563, 2006.

[20] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.