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

by

Farrukh Ahmed

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Most data mining and machine learning techniques rely on a single flat table and assume balanced training data. However, most real-world applications comprise databases having multiple tables and imbalanced data. It becomes further complicated in the realm of Big Data where related information is spread over different data repositories. This work focuses on the automatic construction of a mining table by aggregating information from multiple local tables and additional data sources as external tables in a multi-relational database.

Our work extends data aggregation techniques by exploring paths where a single table is traversed multiple times. The existing techniques do not generate attributes that exist on such paths or do not generate them efficiently. However, these paths contain useful past information. Our framework for Generating Attributes with Rolled Paths (GARP) also prevents leakage of the class information by avoiding features built after the knowledge of the class label. While generating new attributes, our system discovers certain patterns that provide useful insights for decision making.

Experiments are performed on a transactional dataset from a U.S. consumer electronics retailer to predict product returns and identify reasons behind those returns. In addition, we augmented the retail dataset with Supplier information and Reviews to show the value of data integration. This dataset has the class imbalance problem, since product returns represent only 10% of the complete dataset. The results show that our technique improves classification accuracy and discovers new knowledge even in the presence of the class imbalance. Our scalability analysis shows that our approach can handle an increasing load of data in a linear fashion.

To my grandfather,

Mian Mushtaq Ahmed

For making me what I am.

Acknowledgements

First and foremost, praises and thanks to the Almighty God, for providing all the guidance, opportunities and abilities to achieve my goals.

I would like to thank my supervisor, Dr. Osmar R. Zaïane, for his continued guidance and encouragement throughout my research. His incredible support, in all aspects of my graduate life, helped me to reach the finish line and graduate with a Masters degree.

I would also like to thank Dr. Michele Samorani for helping me to understand his work and providing constructive feedback on my research. Prof. Mario A. Nascimento and Prof. Paul Lu also taught me amazing things during my program.

My friends have always been there for me whenever I needed them. I would especially thank Shazan and Luke for all their support. I thank Colin and Shazan for their valuable comments on my manuscript.

Most importantly, my family has played a crucial role in my success. I am blessed to have a family of such amazing people. My grandfather is the one who made me what I am today. His devoted love and efforts to bring the best out of me have no match. My parents and siblings have prepared me to face challenging situations in my life. My Maa (mother), Choti Maa and Grand Maa have always looked after me no matter what. My uncles have always been outstanding mentors. Lastly, I want to mention two wonderful individuals in my life: my caring wife and my lovely daughter, who have always kept me joyful and happily stayed beside me in all circumstances.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

In recent times, we have witnessed a substantial increase in our ability to collect data from various sources like sensors in portable devices, web portals and social networking; in different formats, from independent and connected applications. These sources are also changing the structure along with the size of data at a tremendous pace. This continuous growth of data increases the need for efficiently managing data. On the other hand, this huge scale production of data has created vast opportunities for enterprises, healthcare sector, and educational institutions [10]. It can be foreseen that Internet of things (IoT) applications will raise the scale of data to an unprecedented level [17, 53]. People and various devices (from their home gadgets to cars, to buses, to airports) are all loosely connected with each other. This generates a huge data ocean from which valuable information can be extracted and discovered to help in making better decisions and improving the quality of life. It also leads to a dramatic paradigm shift in our scientific research towards data-driven discovery [42].

This is where the idea of Big Data mining [15] comes into play. It has the capability of extracting useful information from a large amount of data available through a variety of sources[1]. Big Data is commonly defined along the popular 4 V's; Volume, Velocity, Variety and Veracity but often the focus is mainly on Volume. However, the value of data is in the integration of disparate data sources and

[1]Gartner says that Big data challenge is not only about volumes of data. http://www.gartner.com/newsroom/id/1731916

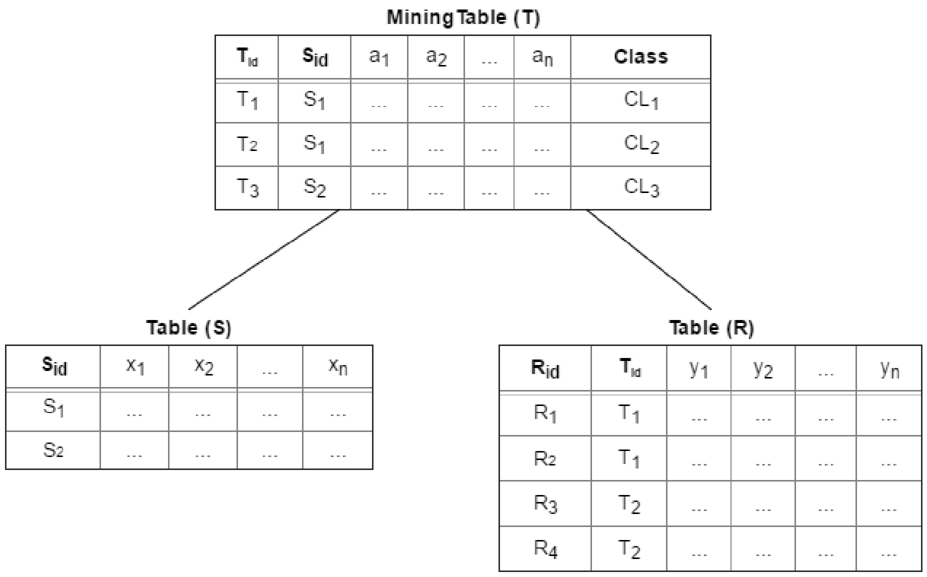
this integration is a major pillar of Big Data, yet very few tools exist to truly take advantage of data integration in Big Data analytics [58]. Today, applications rely on a data layer that stores information efficiently by spreading it over multiple tables. Additionally, integration of external data sources provides a great opportunity to harvest valuable knowledge. 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 [32].

In real world applications, data is distributed in multiple tables. However, most of the data mining and machine learning techniques rely on a single table. Before applying these techniques, data available in multiple tables and additional data sources as external tables in a multi-relational database, needs to be accumulated in a single table. This flat mining table can be constructed manually by asking domain experts to combine and aggregate attributes into known potential predictors. However, instead of handcrafted features, one might automatically generate discriminant attributes from aggregated data which is the essence of our approach herein.

Along with the issue of data spread over multiple tables, the class imbalance is yet another problem that exists in real world applications. But, most of the machine learning techniques assume balanced training data. As classifiers become biased towards the majority class to achieve better accuracy, instances belonging to the minority class are mostly misclassified. However, correctly classifying the minority class is important in several domains with imbalanced data like fraud detection, credit assessment and medical diagnosis [41, 36, 33]. The class imbalance has been identified as one of the most challenging problems in data mining research [57].

1.2 Problem Description

Figure 1.1 shows a generalized example of a mining table with related information stored in other tables. The mining table (T) has a many-to-1 relationship with table (S) and 1-to-many relationship with table (R). To utilize all the available data for the learning tasks, information available in the related tables should be aggregated into the mining table. It is easy to aggregate information from the table (S) with a simple join, as each row in the table (T) is related to a single row in the table (S). However, capturing information from the table (R) requires the application of aggregation operators to summarize the information from multiple rows related to a single row from the table (T). Additionally, more expressive attributes can be generated by combining information from both tables (R) and (S).



T_{id}	S_{id}	a_1	a_2	...	a_n	Class
T_1	S_1	...	...	...	...	CL_1
T_2	S_1	...	...	...	...	CL_2
T_3	S_2	...	...	...	...	CL_3

S_{id}	x_1	x_2	...	x_n
S_1	...	...	...	...
S_2	...	...	...	...

R_{id}	T_{id}	y_1	y_2	...	y_n
R_1	T_1	...	...	...	...
R_2	T_1	...	...	...	...
R_3	T_2	...	...	...	...
R_4	T_2	...	...	...	...

Figure 1.1: Mining table with associated information spread over a relational database

Consider an example of a retail store database that contains information related to product sales. Even the simplest structure of the database will have three different tables for Customers, Products and Purchases. Assume that each purchase record represents the sale of only one product purchased by a single customer. We can add a class attribute ‘return’ to the Purchase table, which indicates whether the purchase is returned back to the store.

Sometimes customers are not satisfied with their purchases due to several reasons related to the product like quality, value or failure to meet their expectations. This might eventually result in the return of the product. A study reveals that on average the return percentage of a product can range from 10% to 25% [22]. In the U.S. alone the cost incurred by product returns is approximately \$100 billion [49]. Therefore, it is important to identify the product returns and causes behind them. As returned products represent a small subset of the whole dataset, the retail domain suffers from the class imbalance problem as well.

The Purchase table in the retail store database is the mining table with the class label ‘return’. In order to expand the Purchase table, Customer and Product information should be aggregated from the related tables. In the absence of the automatic generation techniques, domain experts would build these attributes manually based on their knowledge and experience. Table 1.1 and 1.2 represent an example of the mining table generated by manually adding these attributes.

Table 1.1: Mining Table (Purchase) without any information from other tables

purchase_id	online	price	quantity	return
...	...	...	...	...
...	...	...	...	...

Table 1.2: Additional features created by an analyst by handpicking attributes

customer return proportion	product return proportion	amount spent by the customer	mean age of the customers
...	...	...	...
...	...	...	...

These additional attributes in Table 1.2 (from left to right) represent:

- returning behavior of customers
- market performance of products
- spending activity of customers
- age group of customers buying a same product

Although this example is fictional, it shows that analysts may build not only those attributes that are readily available but also more complex ones that could potentially be good predictors of the class. Furthermore, product reviews available on forums can be utilized to predict returns. This publicly available information can be very helpful when integrated with the Retail store database.

However, the analysts need to manually design the database queries to build these new attributes that can utilize the information from related tables and other data sources. Some complex attributes may even require the application of data warehousing techniques. This manual process is not efficient in terms of effort and productivity. Moreover, if the real predictor of the class is something unexpected, it will not be suggested, because all the generated attributes are based on the existing domain knowledge of the experts.

We aim to automate this process of generating attributes and structuring them in a single table. Our main focus is to generate attributes carrying useful information about the past, like customer's return history at a retail store. This pre-processing step of aggregation before applying traditional learning techniques can help in utilizing all the available data.

1.3 Importance of Data Integration and Big Data

The amount of data being produced and the ability to collect it from diverse sources is increasing substantially. This tremendous growth of amount of data poses great challenges to efficiently manage data. However, this also provides the opportunity to extract value out of rich information. Big data has received a lot of attention these days and several industries are already working on solutions for big data analytics [58]. Enterprises believe that data-driven strategies are necessary to stay ahead in the competition.

Big data is not just about the massive amounts of data, but it is the complexity of the data that is huge. IBM describes this complexity in four dimensions: Volume, Velocity, Variety and Veracity [24]. These dimensions are widely known as the four V's of big data. The most popular V is the Volume which represents the huge amounts of data being produced. Big enterprises encompass a large amount of data which can be utilized to extract useful information. Along with the Volume, Velocity of the data being produced is also very high. Social networks, e-commerce platforms, sensors, etc can produce tremendous amounts of data at an extremely fast pace. Due to this fast pace, it is essential to analyze this data within considerable time limits. The third V for Variety represents different types of data like text, images, audio, video as well as different sources from where the data can be obtained. These data sources can be from different departments internal to an organization like accounting, marketing, etc and publicly available external data sources like twitter, wikipedia, etc. The availability of data from different sources can give insights that would not be possible otherwise. The fourth V for Veracity reflects the uncertainty of the available data. A large amount of data and disparate sources can also comprise unreliable information. In order to take proper decisions, it is important to identify if the data being used is reliable. Recently, some other V's have been suggested by data scientists like Value, Visualization and Vulnerability [58].

The value in data integration is often overshadowed by other dimensions. However, it is one of the most important aspects of big data analytics. The inclusion of data from external sources provides greater insights compared to the locally available data. As mentioned earlier, Netflix has an extensive amount of data related to the users like movies watched, ratings provided, movies browsed etc. Netflix can use this data to build models to recommend movies to the users based on their tastes and other available information. This data can be further enriched by adding information from a publicly available external data source similar to IMDB. For example, IMDB can be used to extract the information related to a movie rated by a user at Netflix and recommendations can be provided based on that information like the plot keywords, taglines, trivia etc. Hence, data integration plays a major role in big data analytics. Our research focuses on big data integration with the extraction of information from external data sources along with the information available in a local multi-relational database.

1.4 Thesis Statements

In this work, we address the challenge of automatically aggregating information from all the available data sources and summarizing it in a single flat table. The goal is to address the following statements:

- It is possible to generate discriminant attributes carrying past information by aggregating information from all the related data sources, which can help in improving the classification performance.

- The automatic attribute generation process can find unexpected patterns in the datasets leading to the discovery of new knowledge.

- The discriminative attributes can help in learning better classification models by rectifying class overlapping regions in data, even in the presence of the class imbalance.

- It is possible to scale this extensive process of generating attributes to handle a large amount of data.

1.5 Thesis Contributions

The closest work in comparison to our research is the Dataconda framework [45] for automatically generating attributes in a relational database. However, we improved upon Dataconda on few aspects. The main contributions of this thesis are:

- A framework for automatically generating attributes based on the approach of aggregating past information. It summarizes information from all the data sources and constructs a single table which can be used to apply learning techniques. In contrast to Dataconda, the generated attributes utilize all the available training data which helps in learning better classification models. Additionally, they help in reducing the overlap of classes to achieve better prediction accuracy even in the presence of the class imbalance.

- An efficient approach for generating attributes which can scale with the increasing load of data. Our attribute generation process is significantly faster than the existing exploration approach of Dataconda.

- Knowledge discovery by finding unexpected patterns associated with the class. While constructing the mining table, we generate the description of attributes which can be used to understand the good predictors.

1.6 Organization of the Dissertation

The rest of the dissertation is organized as follows:

- In Chapter 2, we review the related work in the area of Multi-relational Data Mining and discuss the most common techniques to handle the class imbalance problem. We start with a brief introduction of the Multi-relational Data Mining domain. Then, we discuss two different directions of research towards feature discovery and classification in the context of relational data. For the class imbalance, we first explain the problems associated with an imbalanced dataset. Then, we present the two approaches to solve the class imbalance problem: (1) data pre-processing methods of sampling; and (2) handling imbalance during learning.

- In Chapter 3, we describe the process of constructing a mining table from multiple tables in the database as well as external data sources. First, we have a preliminary discussion to formulate the problem of generating attributes which carry useful insights regarding the past. Then, we explain the algorithm for Generating Attributes with Rolled Paths (GARP) based on the approach of finding the past information.

- In Chapter 4, we evaluate our framework by conducting experiments to measure the classification performance and the ability to find discriminant attributes. First, we describe the real dataset of a large US electronics retailer 'Circuit City' and synthetic external data sources of Suppliers and Reviews. Then, we demonstrate the two sets of experiments to evaluate our technique: (1) with the Retail database only; and (2) with the integration of external data sources of Suppliers and Reviews. Finally, we evaluate the scalability of our technique by varying the load of data.

- In Chapter 5, we conclude by summarizing the presented work and evaluation results. We also provide some directions to continue this research in the future.

Chapter 2

Related Work

In this chapter, we review the related work in Multi-relational Data Mining (MRDM) and some of the most prevalent techniques to handle the imbalance in the class distribution of data samples in the case of supervised learning.

2.1 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 [27]. In literature, MRDM is also referred as Relational Data Mining (RDM). The need for MRDM arose with the widespread use of the relational model for handling data in real world applications.

In a relational database, information is distributed in multiple tables to efficiently store and retrieve complex structural data. This results in incompatibility with Propositional Data Mining (PDM) techniques as they require data to be in a single table. The research in the area of MRDM has introduced relational versions of several propositional techniques including but not limited to multi-relational association rule discovery, multi-relational decision trees and multi-relational distance based methods [14].

In real applications, a relational database can have tens if not hundreds of tables. A simple example of a relational database with two tables (‘Purchase’ and ‘Customer’) is shown in Table 2.1. In this scenario, propositional learning techniques applied on the Purchase table cannot find any patterns related to the customer information. Hence, MRDM is required to determine patterns like the relationship of returns with income levels of customers. The income level of a customer could indeed influence whether the purchased product would be returned.

Table 2.1: Relational database representing purchases made by customers

Purchase table

customer_id	online	price	quantity	return
C1	Y	\$65	1	Y
C1	N	\$10	3	N
C2	N	\$45	1	Y
C3	Y	\$50	1	N
C3	N	\$15	5	N

Customer table

customer_id	age	gender	income level
C1	22	F	2
C2	25	M	3
C3	34	F	5

Thus finding the relationships scattered in the tables, that could have for instance discriminant power in classification, is pertinent. 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) [31] to extend learning techniques in a way that they can handle relational data. On the other hand, the second approach called Propositionalization [29], focuses on aggregating data from multiple tables in a single table so that traditional learning techniques can be applied. In literature, Propositionalization is also referred as an alternate to MRDM as the learning step is propositional [14]. However, the aggregation step looks for patterns in the relational context which is the essence of MRDM.

2.1.1 Inductive Logic Programming (ILP)

ILP [31] treats tables as entities comprising of facts. For example, Customer(Bob, M, 25) and Product(Laptop, HP, \$900) represent facts. The approach works by using the induction engine to derive rules for the prediction of the class. A simple starting rule for defining a purchase in the retail database would be:

Purchase(Bob, Laptop) $\leftarrow$ Customer(Bob), Product(Laptop)

A Relational decision tree (RDT) based on ILP is constructed with nodes corresponding to First Order Logic (FOL) clauses [14]. It is a binary tree, because a conditional clause in an internal node can only result in *true* or *false*. The tree is traversed based on these conditions and the leaf level predicts the class value.

An example RDT for the retail store is shown in Figure 2.1. The FOL clause *purchase(C,P)* at the root node shows the purchase of the product 'P' by the customer 'C' and the predicate *isCheap(P)* reports if the product 'P' is cheap or not (based on the defined level of price). The subsequent clause *pastReturn(C)* determines if the customer 'C' has returned any purchased product in the past. The tree predicts a purchase to be a return if the product is not cheap and the customer has returned any purchase in the past.

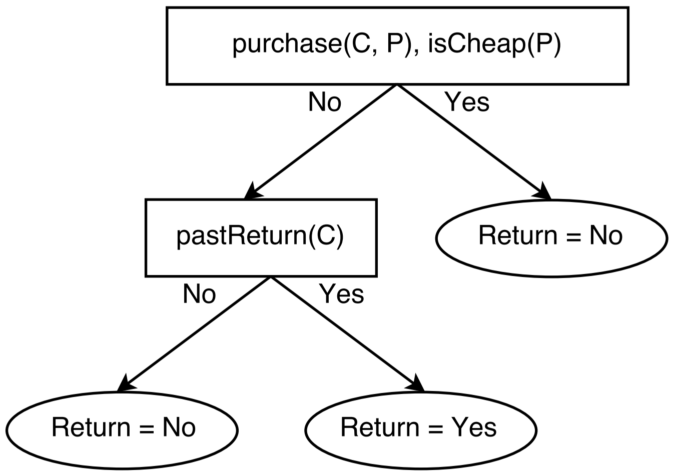


Figure 2.1: ILP based Relational Decision Tree (RDT) for the retail store database

As ILP based approach 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(C)* determines whether the customer has returned a purchase, however, it does not tell how many purchases made by the customer ended being a return. Additionally, the learning phase is tightly coupled with the attribute generation phase which makes it incompatible with most of the existing learning techniques.

2.1.2 Propositionalization

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 the traditional learning techniques on the flat table generated in the first step.

The Propositionalization phase starts with the exploration of a database by determining the possible paths to aggregate information. Figure 2.2 shows an example of paths in a database with a search depth of 3. The table 'T_0' is the target table with a class label, where all the information has to be aggregated and becomes the flat mining table. Each depth level shows the related tables that can be used to aggregate information to the mining table.

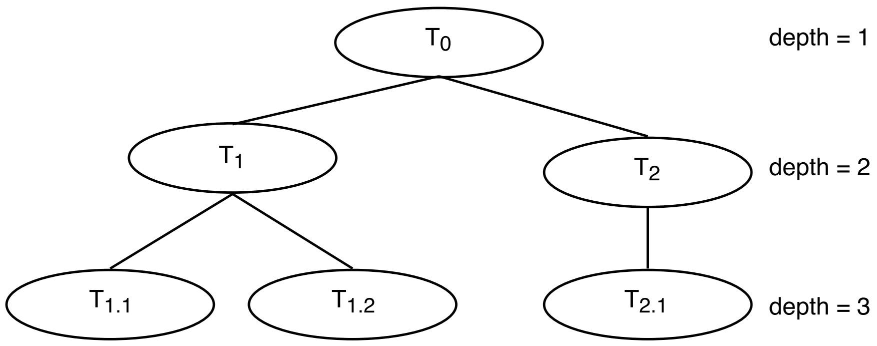


Figure 2.2: The demonstration of paths in a database up to depth 3

After generating possible paths, aggregation is performed to summarize information that exists on each path. The aggregation process starts from the table at the end of the path and brings the information back to the target table. For example, on the path T_0 — T_1 — $T_{1.1}$, information is aggregated from $T_{1.1}$ to T_1 and then from T_1 to T_0. In this way, information is aggregated from all the related tables and summarized in a single flat table.

In the second step, the single flat table generated at the end of the Propositionalization phase is directly used with traditional learning techniques. Hence, Propositionalization can be complemented with different learning techniques like classification, feature selection, etc.

One of the initial work related to the aggregation of complex information from multiple tables in a single table is presented by Knobbe et al. [28]. The authors introduce an algorithm, called Polka, based on the two-step model of the Propositionalization technique:

```
Polka (DB $D$; DM $M$; int $r, p$)
    $P$ := MRDM ($D, M, r$); /* propositionalization */
    $R$ := PDM ($P, p$); /* propositional data mining */
```

The input for their algorithm is a database ‘D’ and a data model ‘M’ along with the integers ‘r’ and ‘p’ that specify the search extent. The Propositionalization step defined as Multi-relational Data Mining (MRDM) is to transform the relational dataset to a propositional one by summarizing the information from all the tables into a single table ‘P’. This single flat table ‘P’ is then used to apply the propositional data mining (PDM) technique.

The main contribution of the Polka algorithm lies in the MRDM step to generate a single table ‘P’. The authors introduce aggregate functions in the Propositionalization step to summarize information for 1-to-many relationships. The aggregate functions like Count, Avg, Sum, Min and Max are useful to aggregate information from a set of records related to a single record.

Consider a database with Customer and Purchase tables presented in Table 2.2. In this example, the Customer table is the mining table with a class attribute ‘income level’ and the Purchase table is related to the Customer table. The goal is to predict the income level of a customer by aggregating information from the Purchase table to the Customer table. However, attributes cannot be directly attached to the Customer table as each customer can have more than one purchases. In this situation, aggregate functions can help to summarize the information from multiple purchases of a single customer. For example, the following two attributes can be added to the Customer table by applying ‘Sum‘ and ‘Count‘ aggregations on ‘total’ and ‘price‘ attributes in the Purchase table:

- a1: Total money spent by a customer
- a2: Maximum price of a product purchased by a customer

Table 2.2: Relational database representing 1-to-many association

Customer table

customer_id	age	gender	income level
C1	22	F	2
C2	25	M	3
C3	34	F	5

Purchase table

customer_id	online	price	quantity	total
C1	Y	$65	1	$65
C1	N	$10	3	$30
C2	N	$45	1	$45
C3	Y	$50	1	$50
C3	N	$15	5	$15

Mining table

customer_id	age	gender	...	a1	a2	...	income level
C1	22	F	...	$95	$65	...	2
C2	25	M	...	$45	$45	...	3
C3	34	F	...	$125	$50	...	5

The aggregation operators used by the Polka algorithm can generate deeper patterns compared to the ILP approach, irrespective of the number of tables. It also generates some specific patterns by adding refinements to the Count aggregation operator. Figure 2.3 shows an example of a selection graph that can be used with the Count operator to generate an attribute involving Purchase, Customer and Product tables. In this case, the Count function will return the total number of purchases made by customers where the price of a product is above $100.

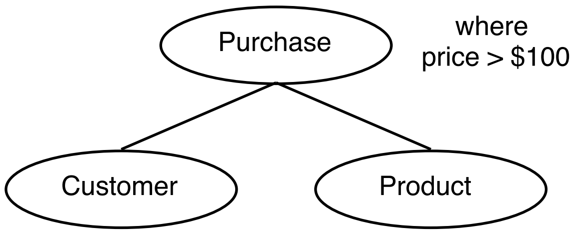


Figure 2.3: Selection graph for the retail store database

However, there is a limitation on the specific patterns produced by the Polka algorithm because conditions to refine data are applied only with the 'Count' aggregation operator. So, the Polka algorithm generates a specific pattern of 'Count the total number of purchases with price $>$ \$100', but misses the patterns with other operators like '(Average/Sum/Min/Max) of the amount spent on a purchase when price $>$ \$100'.

The ACORA framework [39] 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. Such attributes can capture important information about specific subsets of objects.

For example, patterns related to a specific retail location can reflect if there are any problems associated with that particular location. Consider a product that is being sold at multiple locations in Canada. The frequent returns of the product in 'Calgary' and 'Vancouver' for instance can reveal problems associated with the stores, product stock, staff, etc. in these cities.

Table 2.3 illustrates an example of the retail store with the number of returns (out of 10) for each location. The ACORA framework generates attributes by finding the class distribution for purchase locations and generating distance-based features so that an object of a particular class has a smaller distance from its own class distribution. The distribution-based features generated by ACORA would have a smaller distance from the class value 'return = 1' for purchases made at 'Calgary' and ‘Vancouver’. However, these distribution-based features are not interpretable.

Table 2.3: Class (return) distribution based on the purchase locations

Purchase Location	Returns (Out of 10)
Calgary	6
Edmonton	2
Montreal	3
Ottawa	1
Toronto	2
Vancouver	7

The ACORA framework targets the learning task in a relational database by following these steps:

- Exploring the database,
- Constructing new features,
- Selecting good predictors,
- Estimating the model for prediction.

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 [46] 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 [44]. Leakage of the class occurs by using data that has the information of the class itself. This problem can arise when the training set contains data that is part of the test set or the data used for prediction contains the future information. Hence, it is not legitimate to use those features for prediction that are built with the knowledge of the class. For example, attribute with the reasons for product return should not be used for the prediction of a product return. Leakage results in less generalized models and overestimation of the performance. Along with additional paths, the authors also introduce comparison refinements where attributes can be compared to other attributes with the same type and dimension.

Dataconda [45] introduces a framework to generate attributes by joining a table multiple times without using any future information. Our approach for Generating Attributes with Rolled Paths (GARP) is based on the methodology proposed by Dataconda. Our technique resolves the scalability issue with the extensive attribute generation process in Dataconda to cope up with a large amount of data. We provide a detailed discussion in Chapter 3.

2.2 Class Imbalance

The class imbalance arises when one of the classes dominates the other in the dataset. In some cases, this problem can occur due to the limitations in the data collection process, however, several domains intrinsically suffer from imbalance because examples of one class (mostly positive one) are rare [9]. Examples of this include product returns prediction in retail, detection of fraudulent calls, cancer diagnosis, credit assessment, etc. In all these cases, a minority class, typically the class to predict, is overshadowed by a majority class. The class imbalance varies in levels and ratio of the minority to the majority class can be as drastic as 1 to 10,000 [9]. Other than the number of samples, the class imbalance can also occur due to the cost of making mistakes. In some cases, the cost of misclassifying the samples of one class can be significantly higher than the other class. The class imbalance can also occur in the case of multi-class classification problems [55]. In multi-class datasets, imbalance happens when some of the classes are dominated by other classes.

The class imbalance can be divided into two categories based on its impact on classifiers [20]. The first category of imbalance is created by quantity, where the majority class outnumbers the minority class. However, the minority class still has a good representative population so that the classifiers are not necessarily affected. The second type of imbalance is absolute where the minority class is rare in its own. Class imbalance problems of this nature have been shown to be the more challenging due to the lack of representative data. Moreover, these classification problems are further exacerbated by the complexity of the target domain, which amongst other things is effected by the class overlap [20]. Class overlap occurs when the features cannot differentiate the two classes well. Thus, one way to assist the problem of class imbalance is to utilize more discriminative features to reduce the overlap problem.

The class imbalance problem can be solved by: (1) data pre-processing methods of sampling; and (2) handling imbalance in classification. In the following subsections, we discuss two of the most common sampling techniques called random undersampling and oversampling with SMOTE. Then, we discuss the cost-sensitive classification and one-class classification.

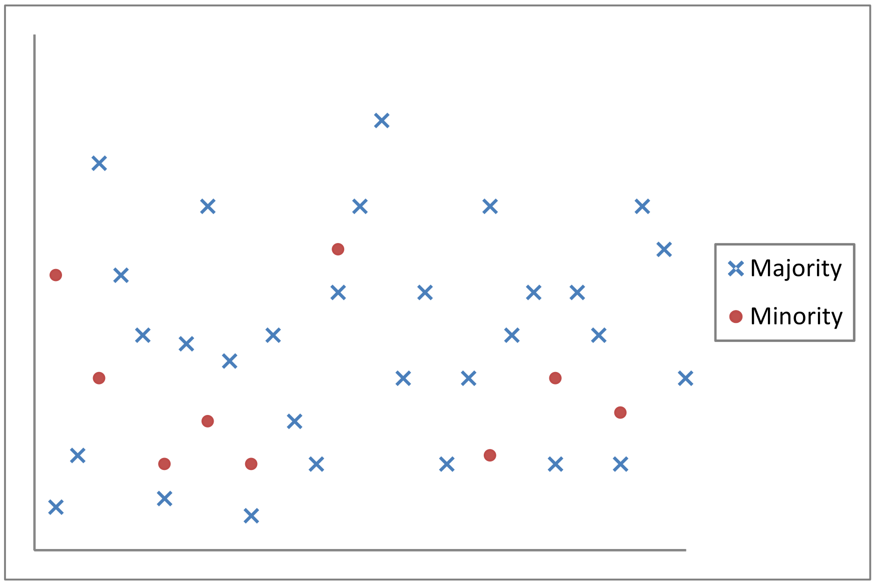


Figure 2.4: A representation of a dataset 'DB' with the Class Imbalance Majority(x) = 30, Minority(.) = 9

2.2.1 Random Undersampling

The undersampling approach solves the class imbalance problem by removing samples representing the majority class. The number of discarded samples depends on the overall class distribution. In order to achieve an equal distribution, samples will be drawn from the majority class until they equalize the minority class in number.

In random undersampling, there is no systematic approach and each instance has an equal probability of being removed.

The simple technique of random undersampling is effective as it removes bias from the induced model [54]. However, the elimination process can remove important information from the majority class. The classification performance can go down if representative examples are discarded from the majority class [23]. Several heuristic-based methods have been proposed to solve this problem [19, 20, 52]. Although these methods have produced good results on certain domains, in general their results have been mixed [2].

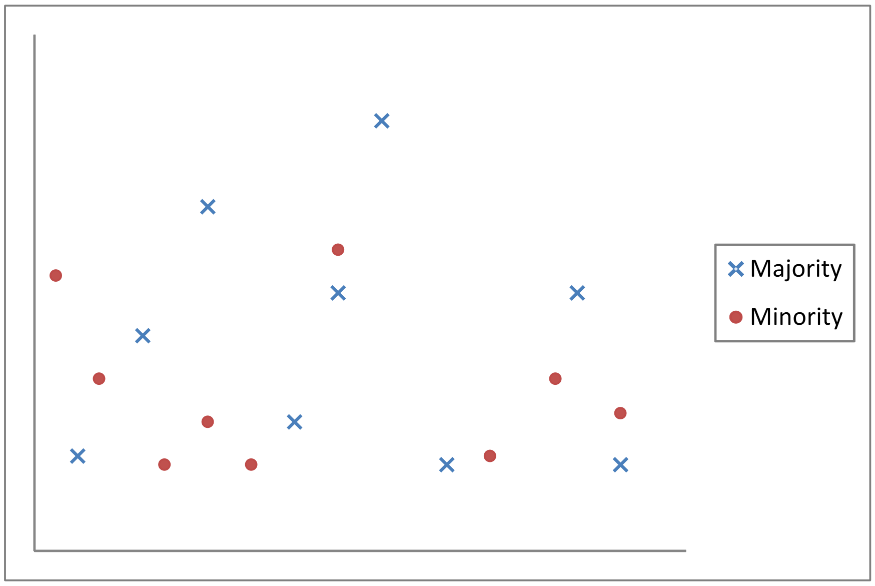


Figure 2.5: 'DB' after Random Undersampling to achieve a uniform distribution Majority(x) = 9, Minority(.) = 9

2.2.2 Oversampling with SMOTE

In contrast to undersampling, oversampling increases the number of samples of the minority class. However, the random approach is not much effective as it only duplicates the existing examples of the minority class which results in the induction of an overfitted model [30].

Chawla et al. proposed a solution by introducing a systematic approach of oversampling known as Synthetic Minority Oversampling Technique (SMOTE) [8]. They solve the overfitting problem by generating new synthetic instances instead of replicating the existing ones. The idea works by looking at k-nearest neighbors of an example from the minority class and generating a new instance by interpolating among them. In this way, SMOTE generates synthetic points that reduce the risk of overfitting by learning a more generalized classifier.

The advantage of oversampling with SMOTE is that it utilizes all the available information by not removing any samples from data. However, SMOTE can suffer from performance degradation in high-dimensional space [54].

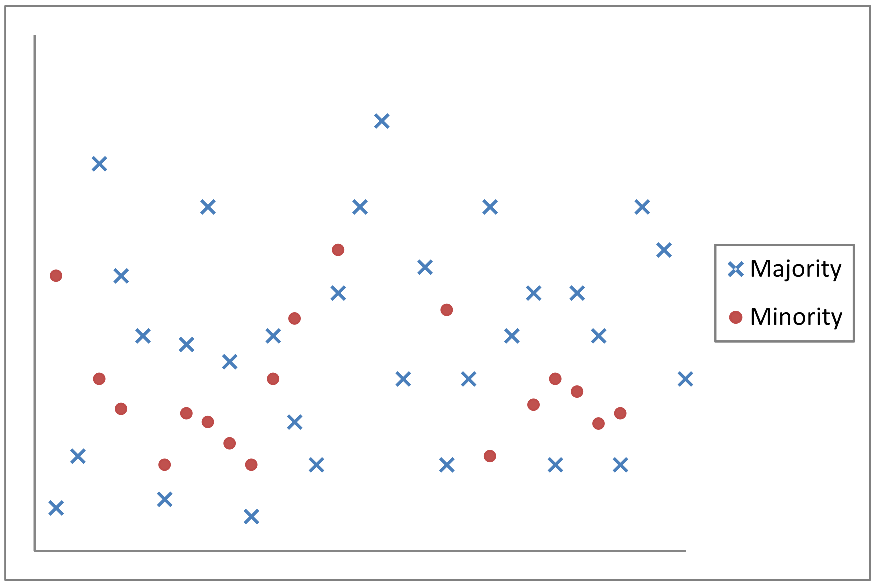


Figure 2.6: ‘DB’ after applying SMOTE to increase the minority class by 100%
Majority(x) = 30, Minority(.) = 18

2.2.3 Cost Sensitive Classification

Cost-sensitive learning takes into account the misclassification cost for each class so that the performance in an unbalanced dataset can be improved. The idea is to maintain a balance by increasing the cost of misclassifying the minority class and optimizing the objective function to reduce the overall cost [34]. For example, the cost of classifying positive cancer as negative (false negative) can be increased so that positive cases can be given more weight in a huge set of negative cases.

The main challenge in cost-sensitive classification is to find an optimal cost for each class. One study suggests a relationship between class distribution and cost thresholds [6], however, the exact relationship is difficult to determine. Consequently, different costs are often tried to find the optimal point.

Some studies have shown that cost-sensitive classifiers produce similar results as sampling [6, 37]. Another research specifically suggests undersampling over cost-sensitive classifiers in the case of high-dimensionality [54]. The main advantage of using sampling techniques is that all the existing classifiers can be used directly without any modifications to incorporate the cost factor.

Table 2.4: Cost matrix with C_p and C_n as assigned costs for incorrectly classifying positive and negative samples respectively

	Positive prediction	Negative Prediction
Positive Class	0	C_p
Negative Class	C_n	0

2.2.4 One-class Classification

One-class classification is used to identify the objects belonging to a single class known as the target class. It learns by looking at the samples from the target class only without any information about the other class. The induction of a one-class classifier involves learning a representation of, or a boundary around, the target class. This is typically applied in connection with the threshold to differentiate the target objects from outliers [50].

This approach of classification is helpful when most or all of the samples in the training data represent the target class. The lack of availability of the other class can be due to an expensive or difficult collection process [11]. One-class classification is ideal for problems such as machine failure detection or prediction, typist recognition and radiation detection [50, 25, 21, 48]. In the domain of machine failure, for example, there is plenty of data for normal operation of the machine whereas training examples for the machine failure class are much more rare and difficult to come by.

One-class classification is useful in domains suffering from extreme class imbalance. However, a study suggests that one-class classification becomes less beneficial with increasing number of samples from the minority class [3]. In that case, sampling is a better approach to solve the class imbalance.

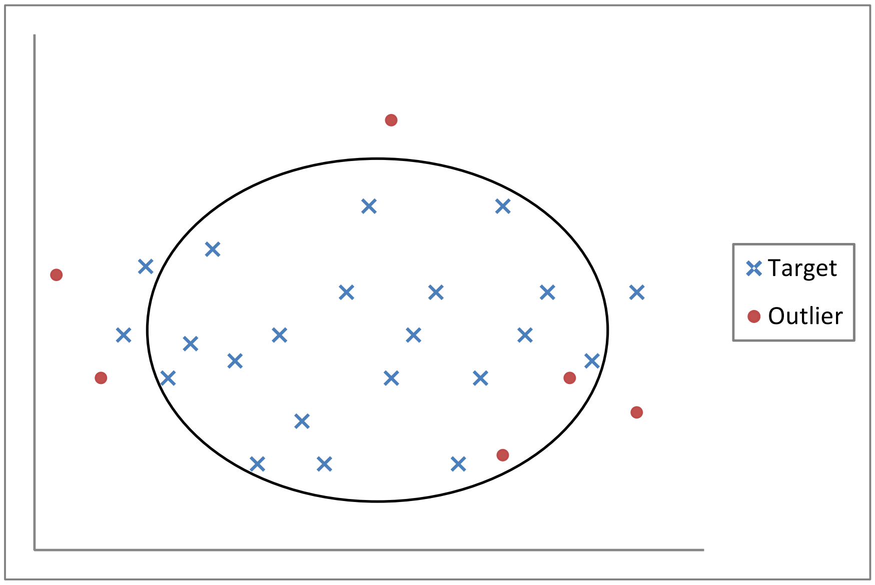


Figure 2.7: A representation of a decision boundary learned by a one-class classifier to identify targets

Chapter 3

Generating Attributes with Rolled Paths (GARP)

Our objective is to construct a flat mining table by adding attributes generated by aggregating information from the related tables within the database and external data sources as well. Figure 3.1 shows a relational database with information scattered over multiple tables and an integrated external data source. The table T_0 represents the target table where all the information needs to be aggregated before applying learning techniques. There are multiple levels on which related information of table T_0 is distributed. Tables T_1 and T_2 exist on the second level which is directly associated with the table T_0. On subsequent levels, tables T_3, T_4 and T_6 exist on depth

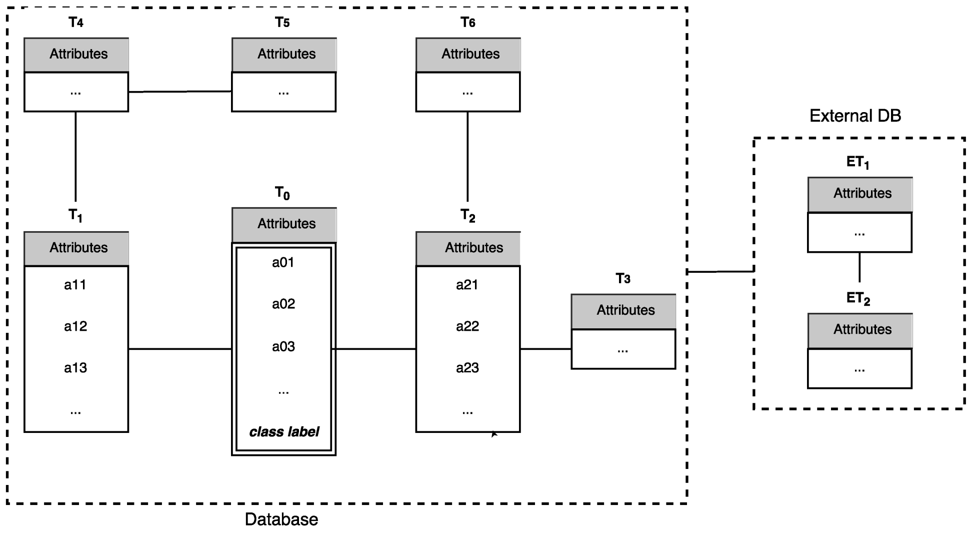


Figure 3.1: Relational Database with integrated External Data Source

level three and table T_5 exists on level four. In real applications, databases can have tens if not hundreds of tables which can exist on even deeper levels. Additionally, multiple external data sources can be available with similar complex association levels. Hence, it becomes challenging to manually aggregate all the available information in a single table. The automatic aggregation of attributes from all the related tables to a single mining table makes the process efficient and can result in features that might be missed while handcrafting attributes.

The aggregation step is performed by joining related tables on a path and adding newly generated attributes to the mining table. The existing aggregation techniques, except Dataconda, do not generate attributes that exist on a ‘rolled’ path. A path is ‘rolled’ if it requires a table to be joined multiple times. In other words, a table appears more than once in the series of joins to perform the aggregation. However, the approach adopted by Dataconda suffers from a scalability issue which makes it difficult to explore an extensive number of attributes for a large amount of data. Our technique introduces an efficient pre-processing method in the aggregation step which can help to generate multiple attributes, while at the same time reducing the load significantly.

The graph in Figure 3.2 shows paths that can be taken to aggregate information from other tables to the Purchase table in the retail store database used as example in Chapter 2. The information available at each depth level is as follows:

- The depth level of ‘1’ represents only Purchase table without any information from the related tables.
- At depth ‘2’, Purchase table can be expanded to include related information by joining Customer and Product tables.
- The paths with depth ‘3’ can generate attributes related to purchase history of customers (Purchase — Customer — Purchase) and products (Purchase — Product — Purchase).

The ‘rolled’ paths shown at depth ‘3’ visit the Purchase table twice. The claim of the opponent of the rolled path is that such paths result in duplicate information available at shorter paths [39]. However, this problem arises because of joining all the tables at once (Purchase ⋈ Customer ⋈ Purchase). Such a path provides information on historical purchases of the product ‘P’ by the customer ‘C’ who just purchased ‘P’.

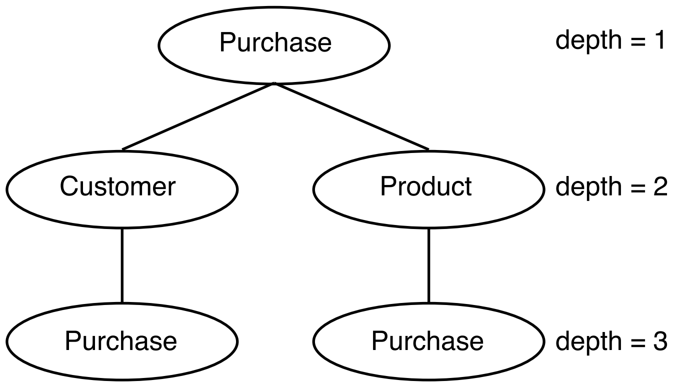


Figure 3.2: Graph representing the paths for data aggregation in the retail store database

Our technique, Generating Attributes with Rolled Paths (GARP), avoids this problem by performing step-wise joins. The aggregation operator is first applied for Customer ⋈ Purchase (e.g. Total amount spent by a customer in the past) which results in a single row for each customer. Then, Purchase ⋈ Aggregation(Customer ⋈ Purchase) adds this attribute to the Purchase table. In this way, GARP generates attributes carrying useful information about the past. Our method aggregates information from external data sources in a similar manner. Hence,

$$\mathrm{A} \bowtie \mathrm{B} \bowtie \mathrm{C} = \mathrm{R}$$

$$\mathrm{A} — \mathrm{B} — \mathrm{C} \neq \mathrm{R}$$

i.e., aggregation on a path A — B — C is not equivalent to A ⋈ B ⋈ C.

We also avoid leakage [44] of the class information like Dataconda by considering dates to avoid future information. The generation of attributes carrying the class information leads to overestimation of the classification accuracy. Hence, it is not appropriate to use the return information of the purchase, however, including the past information of returns is legitimate and provides some useful insights. For example, at the time of predicting a return, a customer’s return history before that point can be used for prediction. However, in contrast to Dataconda, we use dates to avoid future information only in the test data and the training set can benefit from all the available training data resulting in better predictive accuracy. Hence, GARP collects useful information about the past without leakage of the class information. GARP extends Dataconda with:

- an efficient attribute generation process that can handle large amounts of data.
- the generation of features that utilize all the available training data without leakage of the class, resulting in better classification performance.

3.1 Preliminaries

To introduce a running example, we present the complete database structure of our retail store example with external data sources in Figure 3.3. The main characteristics of the dataset are described as follows:

- Purchase table is the **target table**, i.e., the table where all new attributes will be attached and becomes the flat mining table we seek. It has a binary **class attribute** called ‘return’.
- Each table is characterized by a name and a set of attributes. Each attribute has a type which is used to identify the applicable aggregation operators and refinements.
- While aggregating information from the related tables to the mining table, we deal with **1-to-many** and **many-to-1** relationships.
- Suppliers information and Reviews are the two external data sources connected to the retail database. The inclusion of these sources helps to utilize the information available from other sources and shows the value in data integration. Customers often review products on social media and other forums which provide useful feedback and information about the acceptance of products by customers. Hence, it is beneficial to use this information to predict product returns. We assume that reviews are extracted from a public forum

where users have rated products with stars and left comments about them. Star ratings from users can be used directly as a numeric attribute. For comments, we assume that an opinion mining technique [35] has been applied to label them as a positive, negative or neutral rating. Additionally, product related information available with the supplier can be used along with the customer transactions to identify causes of returns that might not be obvious without that information, like returns due to the material used by the supplier.

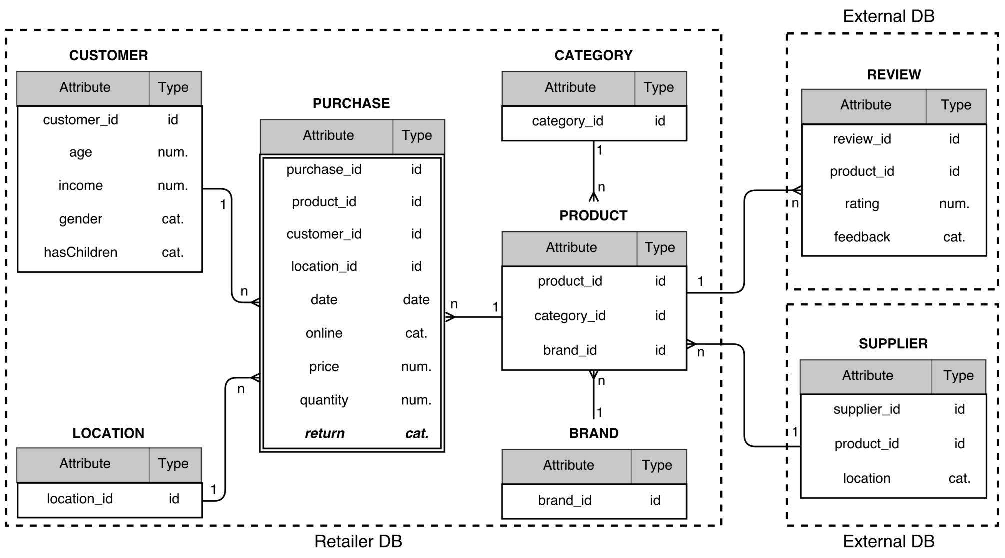


Figure 3.3: Retail Database with external sources of Suppliers & Reviews

3.2 Methodology

In this section, we present our methodology for constructing a flat mining table by generating new attributes. The next section presents an improvement in the aggregation step to scale the attribute generation process.

Our framework starts with reading the schema of the available data sources to determine the relationships between tables. In our example, we have a Retail database and two external data sources of Suppliers and Reviews. The Product table in the Retail database is the link to external databases. In this work, we assume that the supplier and review information is already mapped to the Product table in the retail database. Our goal is to attach new attributes to the *Purchase* table, generated by navigating the Retail, Supplier and Review databases. Our attribute generation approach is illustrated in Algorithm 1, 2 and 3.

We follow the same two-step approach as the previous work on Propositionalization [28, 39, 45], by generating paths that can be taken and then aggregating the information present on those paths.

3.2.1 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 3.2, we start with the Purchase table (target table) at depth = 1. To find the paths that exist at depth = 2, we look at the tables associated with the Purchase table. The tables related to the Purchase table are ‘Location’, ‘Customer’ and ‘Product’. So, the possible paths at depth = 2 are:

1. Purchase — Location
2. Purchase — Customer
3. Purchase — Product

For depth = 3, we add more paths by joining the related tables for each of the paths generated at depth = 2. The subscript is used to represent the addition of a table that already exists on the path.

4. Purchase — Location — Purchase_2
5. Purchase — Customer — Purchase_2
6. Purchase — Product — Purchase_2
7. Purchase — Product — Brand
8. Purchase — Product — Category

When we attach the external data sources of Suppliers and Reviews, two additional paths can be found at depth = 3:

9. Purchase — Product — Supplier
10. Purchase — Product — Review

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. For example, the following paths at depth 4 repeat the information available at depth 2:

1. Purchase — Customer — Purchase$_2$ — Customer$_2$
2. Purchase — Location — Purchase$_2$ — Location$_2$
3. Purchase — Product — Purchase$_2$ — Product$_2$

Consider the path 'Purchase — Customer — Purchase$_2$ — Customer$_2$'. The aggregation starts by bringing the information from 'Customer$_2$' to 'Purchase$_2$'. For each of the purchase, customer information is attached, by using customer_id, which results in additional attributes shown in Table 3.1.

Table 3.1: Additional attributes generated by bringing information from Customer$_2$ to Purchase$_2$

purchase_id	age	income	gender	hasChildren
...	...	...	...	...
...	...	...	...	...

Then, aggregation operators like average, sum, min or max are applied on these attributes to include this information in the table 'Customer' as the relationship between 'Customer' and 'Purchase' is 1-to-many. It results in useless attributes like average(income) of a customer which in turn is the same as income and available at depth = 2 itself. Hence, we avoid paths of a structure 'A—B—A', where the relationship between A to B is 1-to-many and the foreign keys joining A—B and B—A are same. The path generation procedure is illustrated in Algorithm 1.

Algorithm 1 Generating Rolled Paths

```
Input: Target table $T_0$, Table Relationships $R$, max depth level $L$
Output: Possible Paths $P$
1: paths = [$T_0$]
2: level_paths = [$T_0$]
3: current_level = 1
4: for $i$ = 1 to $L$ do
5:    new_paths = []
6:    for each $p$ in level_paths do
7:       last_table = $p$[length($p$)-1]
8:       if $L > 1$ then
9:          previous_table = $p$[length($p$)-2]
10:      else
11:         previous_table = empty
12:      end if
13:      for each $r$ in $R$[last_table] do
14:         if previous_table != $r$ or
            relation(previous_table, last_table) != 1-to-many then
15:            new_paths.add($p$ + $r$)
16:         end if
17:      end for
18:   end for
19:   level_paths = new_paths
20:   paths.add(new_paths)
21: end for
```

3.2.2 Attribute Generation

After completing the path generation process, we aggregate the information available at each path and add new attributes to the target table. For each path, aggregation starts from the end of the path and information is rolled back to the first table in the path i.e. 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 the intermediate tables and appears as an actual attribute directly in the target table 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.

Algorithm 2 Generating Attributes with Rolled Paths

```
Input: Target table $T_0$, max depth level $L$
Output: Mining table
1: $P$ = Paths generated by Algorithm 1 up to 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[$T_{i+1}$] has non-id attributes [$a_1$, $a_2$, ..., $a_n$]
6:             virtual_attributes[$T_{i+1}$] has non-id attributes [$v_1$, $v_2$, ..., $v_n$]
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:            end if
12:            if $T_i$ — $T_{i+1}$ is many-to-1 or 1-to-1 then
13:                for each $a_j$ in candidate_attributes[$T_{i+1}$] do
14:                    virtual_attributes[$T_i$].add($a_j$)
15:                end for
16:            else if $T_i$ — $T_{i+1}$ is 1-to-many then
17:                for each $a_j$ in attributes[$T_{i+1}$]] + virtual_attributes[$T_{i+1}$] do
18:                    for each $Agg$ compatible with $a_j$ do
19:                        /* Algorithm 3 returns list of virtual attributes for $T_i$ */
20:                        $V_i$ = Aggregation($Agg$, $a_j$, $T_i$, $T_{i+1}$)
21:                        virtual_attributes[$T_i$] += $V_i$
22:                    end for
23:                end for
24:            end if
25:        end for
26:    end for
27: end for
28: Mining table = attributes[$T_0$] + virtual_attributes[$T_0$]
```

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.

Aggregations and Refinements

It is essential to apply aggregation operators for 1-to-many relationships to avoid information loss while generating attributes for the mining table. In addition, aggregation operators can be complemented with refinement operators to generate specific patterns. The procedure to apply aggregation and refinements is shown in Algorithm 3. 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 above 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 the compatible aggregations with each of the attributes. Table 3.2 shows some example attributes attached to the Customer table based on the choice of aggregation operator applied on each attribute in the Purchase table. As all the attributes (shown as a_j above) are defined as numeric, each of them can be aggregated with four possible aggregation operators (shown as Agg above).

Table 3.2: Some example attributes generated by aggregation operators for the path Customer — Purchase

attribute	description
Sum (online)	Total number of online purchases
Max (price)	Maximum amount spent on a purchase
Min (quantity)	Minimum number of items in a purchase
Avg(return)	Average number of returned purchases

In addition, it is possible to define custom aggregation functions that, given a list of values, return a single value. In other words, we are not restricted to the aggregation operators of SQL.

The attributes shown above are generated by aggregation operators without any refinements. The refinements can be used to filter data so that they can represent 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 the attribute a_k are:

Ref: $>$, $\leq$, =, $\neq$; for numeric

: =, $\neq$; for categorical

Like Dataconda [45], the two possible types of refinements based on the value of 'c', in the above clause are:

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

Table 3.3 shows some the examples of toValue refinements for the aggregation Avg(P_2.return) on the path Purchase (P) — Customer — Purchase (P_2). Each attribute summarizes the return history of customers based on a specific condition. These attributes are helpful to determine specific patterns.

Table 3.3: Some example attributes generated with toValue refinements for aggregation Avg(P_2.return) on the path Purchase (P) — Customer — Purchase (P_2)

attribute	description
where online = 0	Return history for purchases that are not online
where online = 1	Return history for online purchases
where price $<$ \$500	Return history for purchases below \$500
where price $<$ \$1000	Return history for purchases below \$1000

The values of the constant 'c' for toValue refinements applied on an attribute can be determined by using:

1. all possible values of the attribute
2. values determined by equal-width binning (other discretization ways can be possible, like equal-frequency binning)

The first option is better for attributes that contain categorical data. For example, the two possible values for online are '0' and '1', so the toValue refinements can have conditions like online = 0 or online = 1. The second option, binning, is suitable for numeric attributes. For example, binning the price range of \$1500 into equal-width bins with 3 intervals gives \$500, \$1000 and \$1500. The toValue refinements can have conditions price $<$ \$500 or price $<$ \$1000.

The 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

For each record in Purchase (P), this new attribute represents customer's return history only for those purchases that were conducted via the same medium. For instance, when the current purchase is made online, this attribute represents the average of return for online purchases only. However, if the current purchase is not online, the same attribute represents the average of return for purchases that were not online.

However, as mentioned earlier, attributes that are built using the class leak future information. Such attributes inject the class information in the generated attributes which leads to overestimation of the classification accuracy. This issue is resolved by allowing information only from the past by adding date refinements. We add dates while generating attributes for the test data so that the purchase being predicted does not use any information from the future. Consider one of the example attributes mentioned earlier on the path Purchase (P) — Customer — Purchase (P_2), we can add date refinement as follows:

Avg (P_2.return) where online = 1 and P_2.date $<$ P.date

In this way, we generate attributes for the target table without using any future information.

Algorithm 3 Aggregation Procedure

Input: Aggregation Operator Agg, attribute a$_j$, Tables T_i *and* T_{i+1}

Output: Returns a list of virtual attributes V_i, based on a_j, to Algorithm 1

```
 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:     end if
 8: else
 9:     candidate_ref_attributes = attributes[$T_{i+1}$]
10: end if
11: if both $T_0$ and $T_{i+1}$ have a date then
12:     date refinement = $T_{i+1}.date < T_0.date$
13: end if
14: /* Aggregation without refinement */
15: if virtual_attributes[$T_{i+1}$] is empty then
16:     for each $x$ in $T_i$ do
17:         $R$ = records in $T_{i+1}$ associated to $x$
18:         $v_i$(x) = $Agg(R.a_j)$ from $R$
            [and $R.date < T_0.date$]
19:     end for
20:     $V_i$.add($v_i$)
21: end if
22: /* Aggregation with refinement */
23: for each $a_k$ in candidate_ref_attributes do
24:     for each $Ref$ compatible with $a_k$ do
25:         for each $c$ in refinement_values do
26:             for each $x$ in $T_i$ do
27:                 $R$ = records in $T_{i+1}$ associated to $x$
28:                 $v_i$(x) = $Agg(R.a_j)$ from $R$ where $R.a_k$ $Ref$ $c$
                    [and $R.date < T_0.date$]
29:             end for
30:             $V_i$.add($v_i$)
31:         end for
32:     end for
33: end for
34: return $V_i$
```

3.3 Scaling the Attribute Generation Process

The attribute generation phase results in a very high number of attributes based on different selections of aggregation operators 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 Figures 3.4 and 3.5, we present the internal details of the complete execution process of generating an attribute with aggregation and refinement steps. The Agg and Agg + Ref blocks in the figures provide details for aggregating information 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 final aggregation. This pre-processing time in the aggregation and refinements step grows with increasing load of data.

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

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

As all these attributes are numeric, aggregation operators used for each of the attributes are:

Agg (4) = avg, sum, min, max

The total number of possible aggregations are:

$$\begin{aligned} \text{Total aggregations} &= \sum_{i=1}^{n} \text{Number of Agg}_i; \\ &\quad \text{where n = number of attributes} \\ &= 16 \end{aligned} \tag{3.1}$$

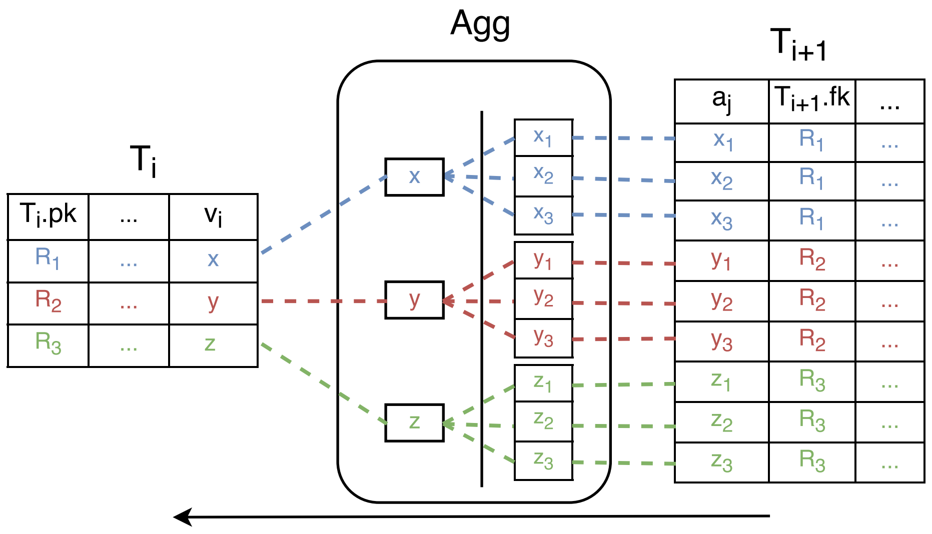


Figure 3.4: Aggregating attribute v_i from T_{i+1} to T_i on path (T_i — T_{i+1}) without Refinement

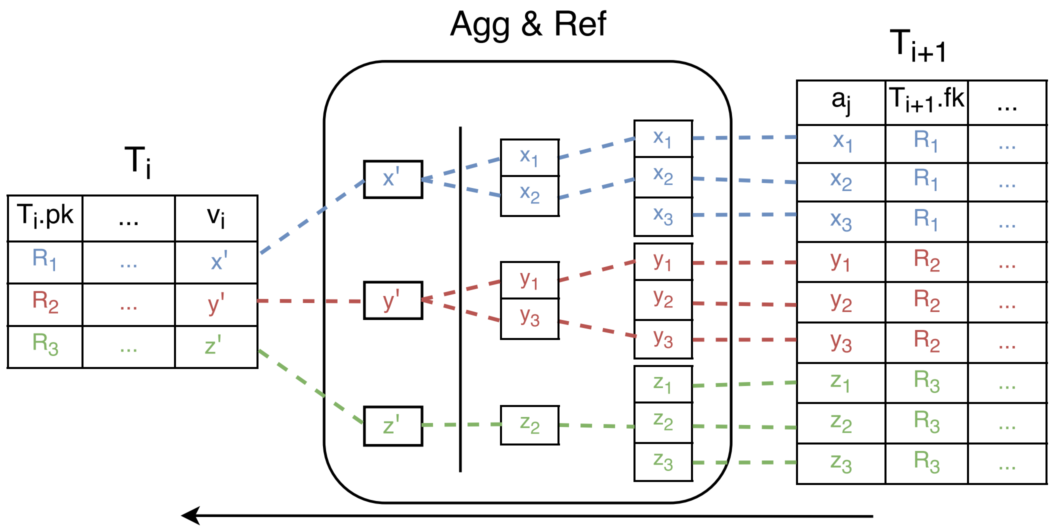


Figure 3.5: Aggregating attribute v_i from T_{i+1} to T_i on path (T_i — T_{i+1}) with Refinement

Assume that refinement values and operators for each attribute are:

online(2): 0, 1; based on all possible values

operators : [=]

price(4): p_1, p_2, p_3, p_4; based on 5 equal-width bins

operators : [<]

quantity(4): q_1, q_2, q_3, q_4; based on 5 equal-width bins

operators : [<]

return(2): 0, 1; based on all possible values

operators : [=]

The total number of possible refinements are:

$$\begin{aligned} \text{Total refinements} &= \sum_{i=1}^{n} \text{Number of values}_i \text{ * Number of operators}_i; \\ &\text{where n = number of attributes} \\ &= (2*1) + (4*1) + (4*1) + (2*1) \\ &= 12 \end{aligned} \tag{3.2}$$

The total number of possible attributes are:

$$\begin{aligned} \text{Total attributes} &= \text{Total aggregations * (Total refinements + 1);} \\ &\text{+1 for aggregations without refinements} \\ &= 16 * (12 + 1) \\ &= 208 \end{aligned} \tag{3.3}$$

In this scenario, a total of 208 attributes are generated for Customer — Purchase aggregation only. This means that the aggregation process has to go through the structuring, filtering and aggregation phases (shown earlier) 208 times.

However, we can make this process efficient by separating the structuring and filtering phases from the aggregation phase as shown in Figures 3.6 and 3.7. 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, by directly applying the aggregation. For example, all the (16) aggregations are performed in conjunction with the refinement 'price $< p_1$', shown in Table 3.4. A temporary result can be generated for the refinement ‘price $< p_1$’ and all these aggregations can be applied together.

Table 3.4: all the (16) aggregations performed with refinement price $< p_1$

	avg	**sum**	**min**	**max**
online	agg_1	agg_2	agg_3	agg_4
price	agg_5	agg_6	agg_7	agg_8
quantity	agg_9	agg_{10}	agg_{11}	agg_{12}
return	agg_{13}	agg_{14}	agg_{15}	agg_{16}

Based on Equation 3.3, the structuring and filtering steps would be reduced to:

$$\begin{aligned}\text{Total preprocessing steps} &= 1 * (\text{Total refinements + 1});\\ &= 1 * (12 + 1) \qquad (3.4)\\ &= 13\end{aligned}$$

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

We also looked into Hadoop [56] to parallelize our technique with a MapReduce [12] based solution. MapReduce is highly scalable and handles inter-machine communication and machine failures within the framework. It’s model works in two phases: the map phase divides the work on multiple machines to produce an intermediate result and the reduce phase merges this result. As our technique generates attributes by following different paths, we tried to divide the load among map tasks where each map task can work on generating attributes for a subset of paths. However, several paths navigate through the same tables and dividing the paths into disjoint sets is difficult. We did not succeed to express our solution in a MapReduce model and suggest a future study to investigate the possibility of a parallel solution for our technique.

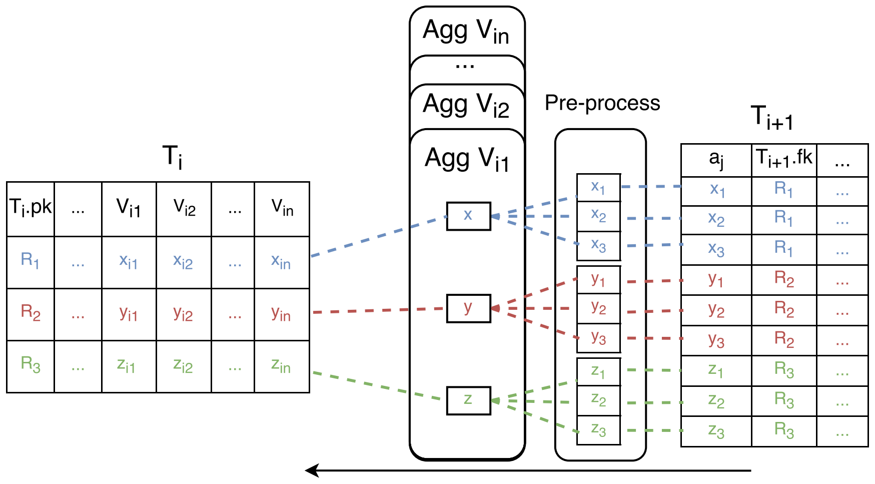


Figure 3.6: Aggregating v$_{i1}$, v$_{i2}$, ... v$_{in}$ from T$_{i+1}$ to T$_i$ on path (T$_i$ — T$_{i+1}$) without Refinement

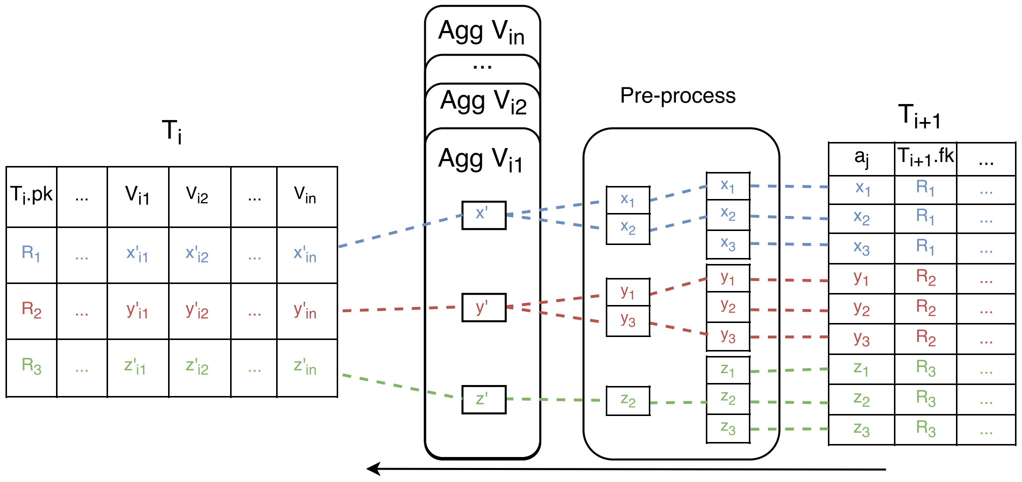


Figure 3.7: Aggregating v$_{i1}$, v$_{i2}$, ... v$_{in}$ from T$_{i+1}$ to T$_i$ on path (T$_i$ — T$_{i+1}$) with Refinement

3.4 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 and assist the knowledge discovery process 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) [51] 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 [26, 13] which are interpretable like subset selection and widely used in the sciences and social sciences [59].

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.

Chapter 4

Experiments

In this section, we evaluate our technique of Generating Attributes with Rolled Paths (GARP) in terms of improvement in predictive accuracy by exploring 'rolled paths' and generation of discriminant attributes from the knowledge discovery aspect. We compare the classification performance of GARP with existing Propositionalization techniques which do not explore rolled paths. We also evaluate the scalability of our technique to generate attributes with an increasing load of data.

4.1 Experimental Design

To determine the predictive accuracy, we conducted experiments with the following classification techniques in Weka [18]:

1. Ripper (JRip) [18]
2. C4.5 (J48) [43]
3. Bayesian Network [16]
4. Random Forest [5]
5. Support Vector Machines (LibSVM) [7]
6. Multilayer Perceptron [47]

We perform our experiments with the Retail database as well as the external data sources of Suppliers and Reviews. The retail database has the class imbalance problem as the number of returned purchases is comparatively smaller than the number of purchases that are not returned. In this case, the classification accuracy measure is not useful as the classifiers predict most or all of the purchases as 'not returned' to achieve better overall accuracy. However, it is important to correctly classify the purchases that are returned. Hence, we use a different measure: Area under the Receiver Operating Characteristic (ROC) curve known as AUC [4]. It represents the probability that a randomly chosen positive instance will be ranked higher than a randomly chosen negative instance. AUC is a good metric to measure classification performance in imbalanced datasets. ROC curve is plotted with the true positive rate and the false positive rate by varying the threshold of probability to assign an instance to a particular class. A random classifier will have AUC around 0.5 and AUC for a perfect classifier would be 1.

We also apply data-preprocessing techniques of undersampling and SMOTE [8] to resolve the class imbalance and compare them with our technique to validate the ability of our approach to handle the class imbalance.

We divided the dataset into training and test set rather than dividing into folds. We take this approach to avoid leakage of the class information as our technique generates attributes in the presence of the class labels of the past information. For example, the mining table generated for purchases in the retail database contains an attribute 'Sum(return) for the past purchases' which is built on the class information of the purchases made before that point. In the 10 fold cross validation, if the selected testing set has purchases made prior to the purchases in the training set then the class information is being leaked. Hence, we need to separate the training and test set to make sure that the testing set does not have any attributes built on the purchases that were performed after that point.

4.2 Dataset and Setup

We conduct experiments on a real transactional dataset from Circuit City, which was a large US electronics retailer. The data is available at the INFORMS Marketing

Science Society[1] [38]. It contains around 115K purchases of around 20K products made by 20K customers. Each transaction in the database contains information about:

- Purchase: price, quantity, returned or not, etc.
- Customer: age, gender, etc.
- Product: category, brand, etc.

We normalized this data as a Retail database presented in Figure 3.2. Since the available data from the retailer has a ratio of around 6:1 for purchase:customer and purchase:product in the complete dataset, it does not have enough representation of the purchasing behavior 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 generated a subset that is also a good representative of the complete dataset in terms of returns.

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. Three groups have products with: high return percentage (>40%), medium return percentage (10-30%) and low percentage(<10%). Next, we retrieve all the purchases and customer records corresponding to these products. Our selection results in 18,182 purchases with 1,868 returns and we take 75% as the training set (13,500).

To show the applicability of our framework for Big Data, we generated synthetic external datasets to simulate the information from Suppliers and Reviews. We generated reviews to simulate a realistic situation where good products usually have good ratings and returned products are more likely to get bad reviews and some products might not have ratings at all. The Supplier dataset has some generic information that suppliers can easily share with retailers. We simulated this database in a way that if manufacturing location is from a specific country it has a high chance of

[1] https://www.informs.org/Community/ISMS/ISMS-Research-Datasets

return. The reason behind keeping this pattern is to evaluate whether our technique can automatically detect it.

Let us briefly discuss some settings used. We enable only those aggregate functions and refinements that are 'logically compatible' with each attribute. For example, for the attribute Customer.age, we enable max, min, and avg, but disable sum, as the sum of ages of a group of customers does not make sense. We defined two categorical attributes Purchase.return and Purchase.online as numeric with two possible values 0 and 1, so as to enable numeric aggregation functions of min, max and avg. The values of the attribute Purchase.price were binned into 5 partitions in order to create the refinements on the price. In the Customer table, the attributes income and age were also binned into 5 partitions. The bins were built using the equal-width criterion.

4.3 Evaluation with Retail Database only

In this section, we evaluate our technique with the Retail database only without considering any external data sources. For classification, we run three sets of experiments on the Purchase table:

1. without aggregated information from other tables,
2. with aggregated information without rolled paths,
3. with aggregations applied using our GARP technique with depth 3 and 4.

For each set of experiments, we first apply the classification techniques on the Purchase table with the class imbalance. Then, we use undersampling and SMOTE to resolve the class imbalance and measure the classification performance in a balanced dataset.

Table 4.1 to 4.6 show the classification results (AUC values) for our experiments with the Retail database. For a baseline, we classified Purchase table without applying any aggregations, so maximum depth = 1 with only three non-id attributes. Then for Propositionalization baseline, we aggregated information from the database without rolled paths, with a depth level of 2 as all the paths beyond level 2 are rolled paths. It resulted in only 7 attributes (additional 4 from Customer table). Finally, we generated attributes using our technique at the depth level of 3 and 4 which resulted in 514 and 846 attributes respectively. These attributes carry useful information related to the past purchases.

Table 4.1: AUC for the Retail Database with C4.5 (J48)

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations (1)	0.51	0.57	0.57
No Rolled Paths (2)	0.51	0.56	0.51
GARP (3)	**0.68**	0.66	**0.68**
GARP (4)	**0.68**	0.65	0.67

Table 4.2: AUC for the Retail Database with Random Forest

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations (1)	0.58	0.57	0.58
No Rolled Paths (2)	0.57	0.60	0.55
GARP (3)	0.75	0.73	0.73
GARP (4)	**0.78**	0.76	0.76

Table 4.3: AUC for the Retail Database with Bayesian Network

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations (1)	0.61	0.59	0.62
No Rolled Paths (2)	0.61	0.59	0.56
GARP (3)	**0.77**	**0.77**	0.55
GARP (4)	0.76	**0.77**	0.56

Table 4.4: AUC for the Retail Database with Ripper (JRip)

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations (1)	0.50	0.57	0.58
No Rolled Paths (2)	0.50	0.55	0.50
GARP (3)	**0.61**	0.60	**0.61**
GARP (4)	0.60	0.58	0.59

Table 4.5: AUC for the Retail Database with Support Vector Machines (SVM)

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations (1)	0.50	0.51	0.52
No Rolled Paths (2)	0.50	0.52	0.50
GARP (3)	**0.76**	0.70	0.61
GARP (4)	0.72	0.70	0.67

Table 4.6: AUC for the Retail Database with Multilayer Perceptron

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations (1)	0.64	0.64	0.64
No Rolled Paths (2)	0.54	0.55	0.50
GARP (3)	**0.77**	0.72	**0.77**
GARP (4)	0.74	0.67	0.71

The experimental results show that resolving the class imbalance with undersampling and SMOTE without aggregating information gives an improvement of at most 8%. Both the techniques give similar results and the improvement is not significant because the attributes do not have enough discrimination power. Even the aggregations without rolled paths do not make much difference and sometimes even drop the performance. However, our technique shows significant improvement in prediction accuracy with all the classifiers, even with the imbalanced dataset, due to the fact that discriminant patterns have been generated. However, depth 4 does not always outperform depth 3. At depth = 4, patterns in the current database can result in over-fitting for some models.

The paths beyond depth level of 4 would generate attributes by applying aggregations upon aggregation operators. For example, ‘Sum(Average of return for the past purchases of a product) for the past purchases of a customer’ lies on the path ‘Purchase — Product — Purchase — Customer — Purchase’ at depth 5. Such complex attributes are less interpretable i.e. difficult to read, understand and transform into actionable knowledge. Hence, levels of relationships in a database can be used to identify the maximum depth level for the attribute generation process.

Additionally, the undersampling and SMOTE results with our technique do not show any improvement, as the generated features have good discrimination power which can help to reduce the class overlap [20]. A comparison of the classification performance with the Retail database is presented from Figure 4.1 to 4.6.

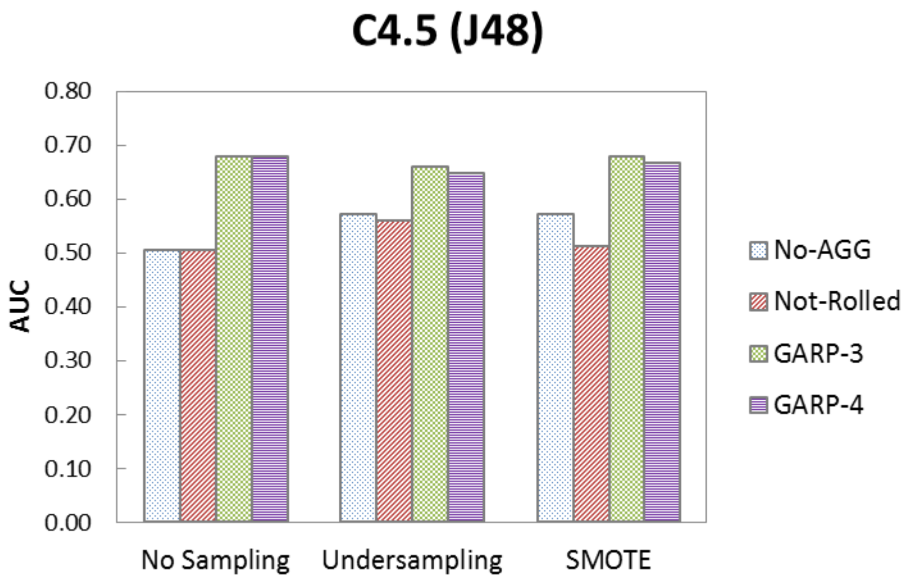


Figure 4.1: AUC comparison in the Retail database with C4.5 (J48)

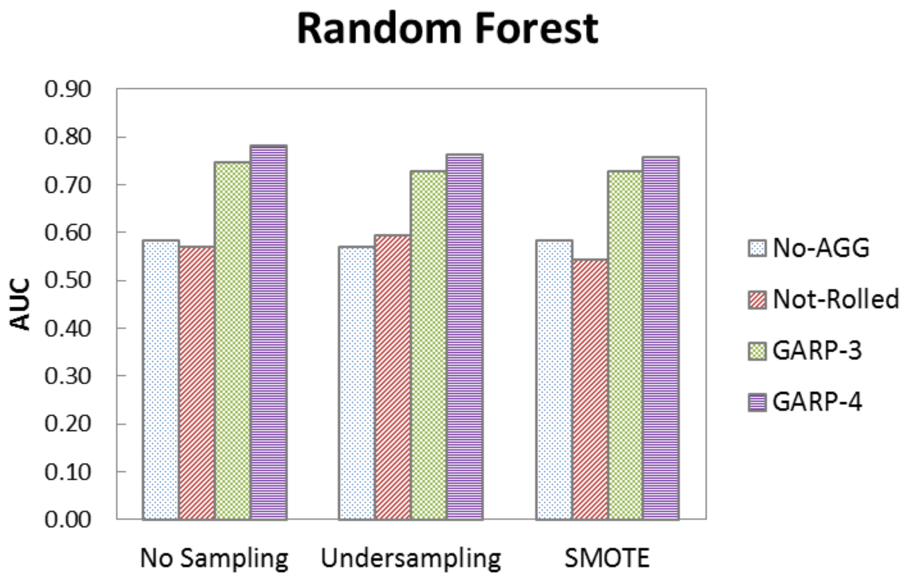


Figure 4.2: AUC comparison in the Retail database with Random Forest

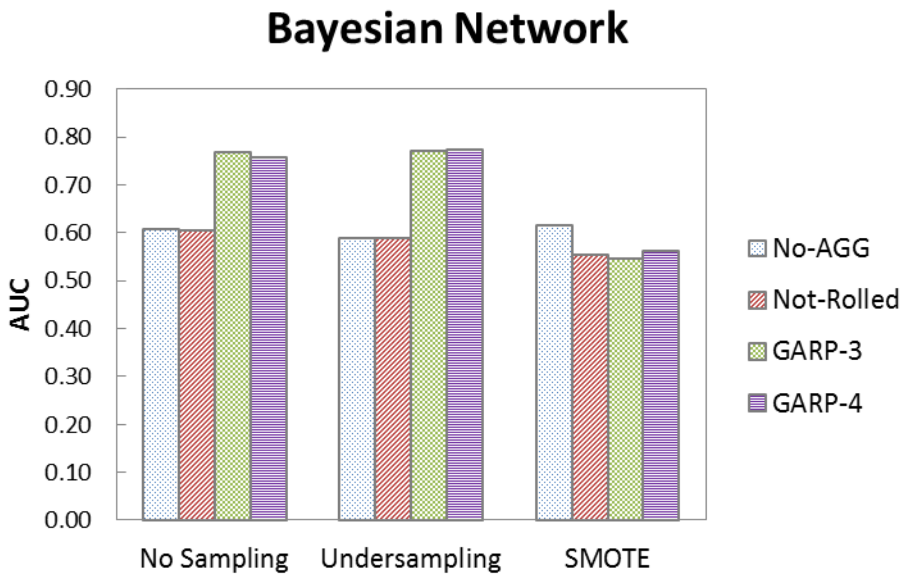


Figure 4.3: AUC comparison in the Retail database with Bayesian Network

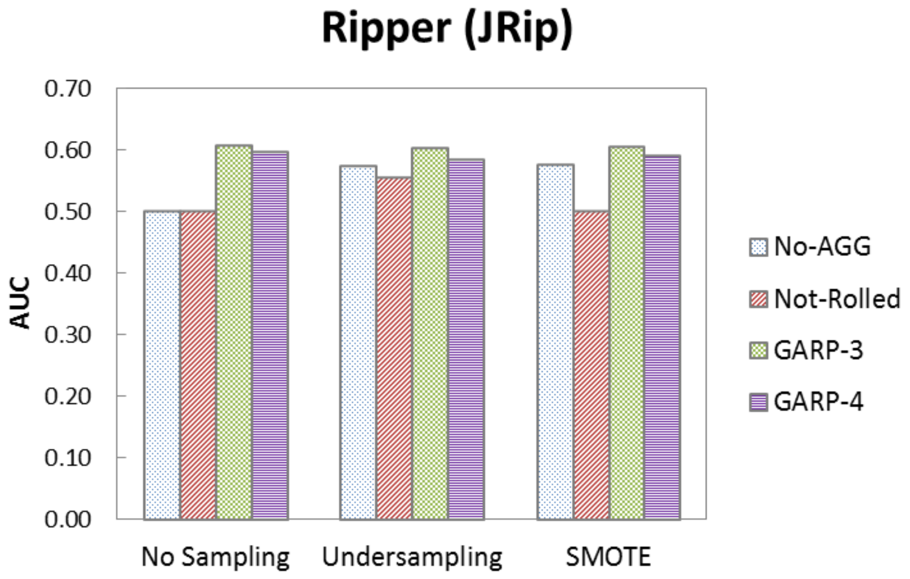


Figure 4.4: AUC comparison in the Retail database with Ripper (JRip)

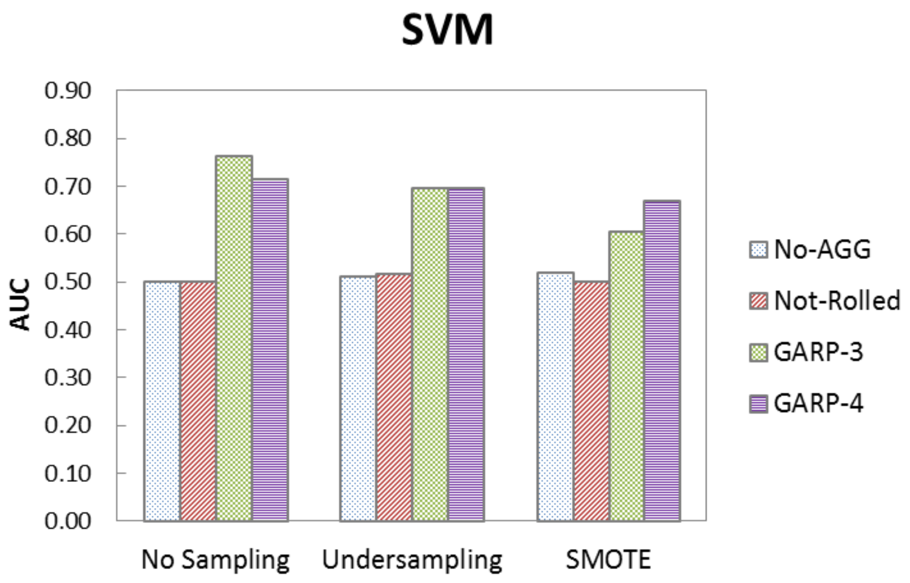


Figure 4.5: AUC comparison in the Retail database with Support Vector Machines (SVM)

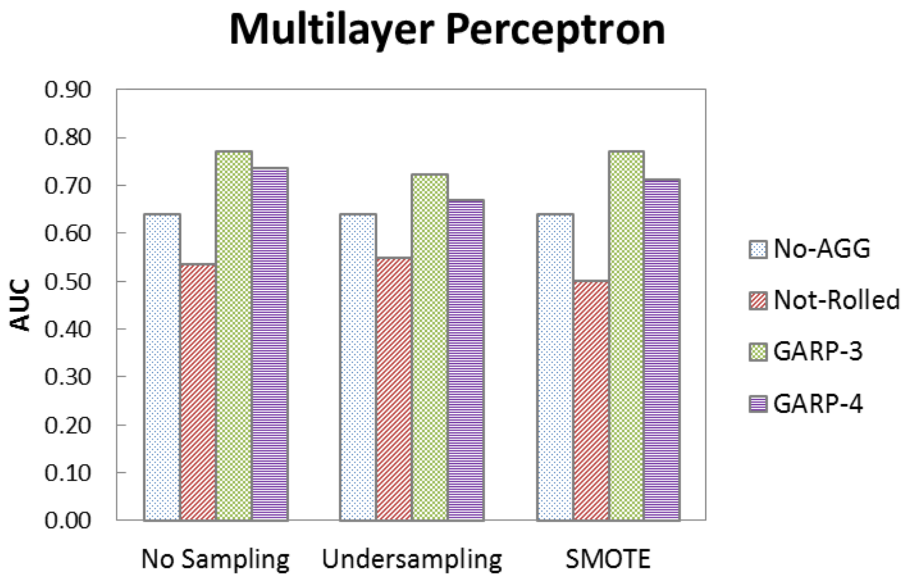


Figure 4.6: AUC comparison in the Retail database with Multilayer Perceptron

Moreover, we look at some of the automatically generated attributes that help us uncover reasons behind product returns. Our default attribute selection procedure selects 10 most discriminant attributes from the whole set generated by our technique. We discuss five of them, reported in Table 4.7, to show that our technique facilitates knowledge discovery. Figure 4.7 to 4.11 show the return percentage of purchases, based on different possible values for these attributes.

Table 4.7: Discriminant predictors within the Retail Database

Attributes	Depth
1. Avg(return) where online = 0, among past purchases of customers	3
2. Sum(quantity) where return = 0, among past purchases of products	3
3. Avg(return) where price $<$ \$1230, among past purchases of products	3
4. CountDistinct(gender of customers), among past purchases of products	4
5. Max(age of customers), among past purchases of products	4

The first discriminant attribute represents a customer's history of returns. This attribute was found at depth 3 on path Purchase — Customer — Purchase. A customer with high *'Avg(return) where online = 0, among past purchases'* is more likely to return products. This fact is verified by analyzing our dataset (shown in Figure 4.7). Additionally, this trend of the customer's buying and return behaviour is suggested by empirical studies in the literature [40]. It shows that customers who tend to buy more products in a sale season often buy undesired items which end up as a return. Similarly, buying unfamiliar products can lead to purchases that do not meet expectations. The *'online = 0'* refinement captures information about our dataset where most of the purchases are not performed online.

The second discriminant attribute is related to the return history of products, with a refinement *'return = 0'*, found on path Purchase — Product — Purchase. This attribute shows the product's reception by customers as *'the total number of purchases of a product without a return'*. Our dataset verifies that product's return history is important by showing that products with a higher number of purchases without a return in the past are less likely to be returned, as presented in Figure 4.8.

FINDING: Return history - Products & Customers

A product can see high returns for several reasons that are particularly related to a product, like quality, value for the price, advertised in a way that product is not meant to be etc. On the other hand, the product may be returned because of the customer related reasons like less careful purchasing behavior or buying products for trial. Hence, past records can help to identify if there is some problem related to a product or customer that may result in future returns.

The third attribute is also related to the return history of products but with a price 'below \$1230'. The previous attribute already shows that products returned in the past are more likely to be returned. However, our technique found this attribute with a refinement *'where price < \$1230'*. The value '\$1230' is set during the binning process, where the price attribute is binned into 5 partitions.

FINDING: the $1230 threshold

This provides information related to the products that are expensive and represent the best available quality. These products are purchased by customers who can spend a big sum of money for the best available product. On the other hand, customers who are more careful while spending will look for a good enough product in a lesser range. These customers are more likely to return products that are not a good fit compared to the customers who can spend money on the most expensive products.

Another significant attribute represents information found at depth 4, gender of the customers buying the product. This attribute lies on the path Purchase — Product — Purchase — Customer. Figure 4.10 shows that the return percentage of the products that are used by a particular gender are more likely to be returned.

Finally, the last attribute in Table 4.7 suggests that returns can be predicted by looking at the maximum age group of customers previously buying that product. There is a significant difference in returns for different age groups. The data suggests that products that are being bought by younger people are returned often.

All these patterns exist on rolled paths, where Purchase table is navigated twice. Our technique found these patterns automatically from real customer transactions data. The selection of the last two attributes related to the age and gender of the customers shows the ability of our methodology to find non-trivial attributes that can be used to further investigate the domain.

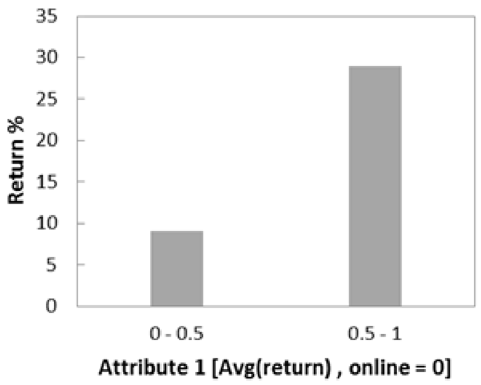


Figure 4.7: Attribute 1: Return percentage based on Avg(return) where ‘online = 0’, among past purchases of Customers

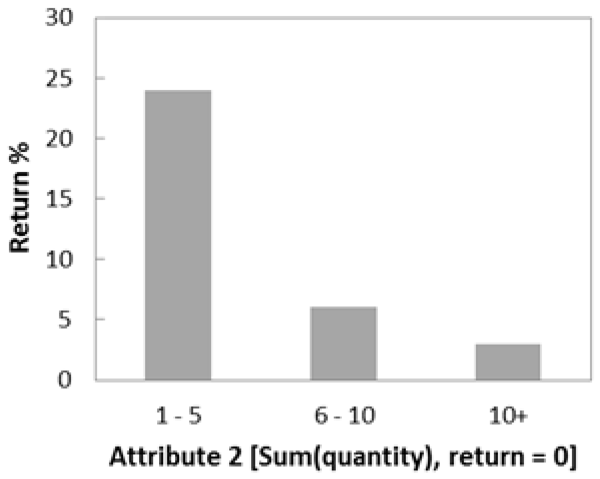


Figure 4.8: Attribute 2: Return percentage based on Sum(quantity) where ‘return = 0’, among past purchases of Products

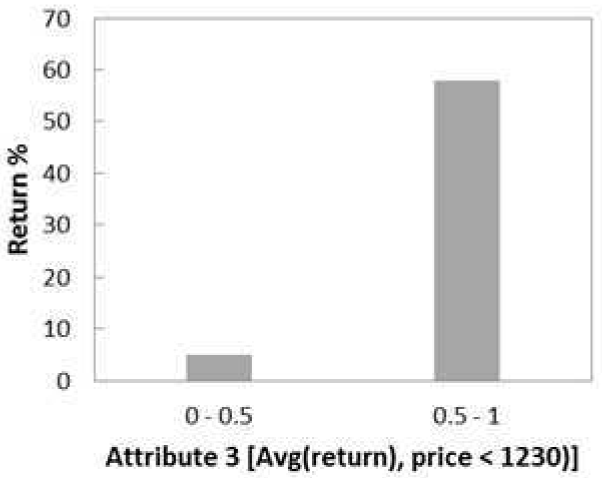


Figure 4.9: Attribute 3: Return percentage based on Avg(return) where ‘price < 1230’, among past purchases of Products

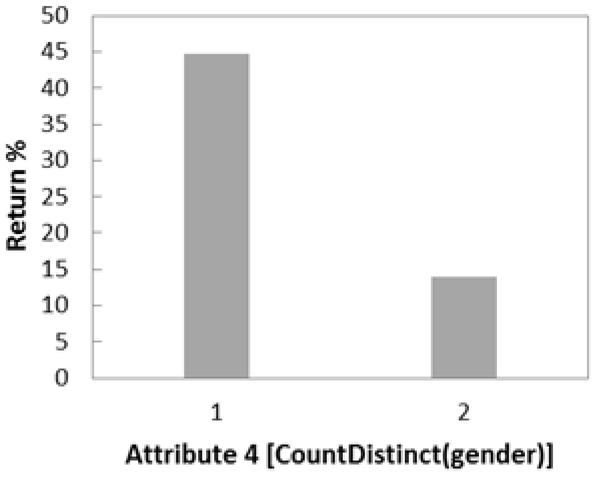


Figure 4.10: Attribute 4: Return percentage based on CountDistinct(gender of customers), among past purchases of Products

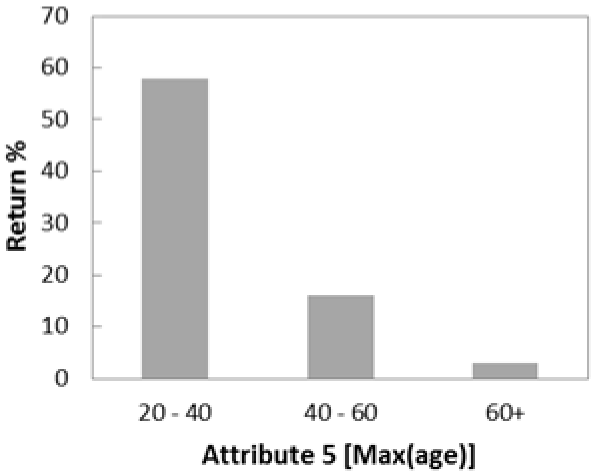


Figure 4.11: Attribute 5: Return percentage based on Max(age of customers), among past purchases of Products

4.4 Evaluation with the integration of External Data Sources

In the second phase of our experiments, we combine our two external data sources of supplier information and reviews. We generate new attributes with the integration of data from:

1. Supplier only,
2. Review only,
3. Both Supplier and Review.

The classification results for the mining table generated along with the external data sources are shown from Table 4.8 to 4.13. The predictive accuracy increases with the integration of external data sources in almost all the cases. Figure 4.12 - 4.17 compare the accuracy achieved with external sources, at a maximum depth of 3 and 4.

Table 4.8: AUC for integrated datasets with C4.5 (J48)

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations			
Retail (1)	0.51	0.57	0.57
Aggregations using GARP			
Supplier (3)	0.68	0.66	0.68
Supplier (4)	0.68	0.65	0.68
Review (3)	0.70	0.69	0.68
Review (4)	**0.71**	0.69	0.68
Supplier & Review (3)	0.70	0.69	0.69
Supplier & Review (4)	0.70	0.69	0.67

Table 4.9: AUC for integrated datasets with Random Forest

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations			
Retail (1)	0.58	0.57	0.58
Aggregations using GARP			
Supplier (3)	0.77	0.73	0.73
Supplier (4)	0.79	0.76	0.76
Review (3)	0.81	0.75	0.74
Review (4)	**0.82**	0.78	0.78
Supplier & Review (3)	0.80	0.76	0.75
Supplier & Review (4)	**0.82**	0.78	0.78

Table 4.10: AUC for integrated datasets with Bayesian Network

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations			
Retail (1)	0.61	0.59	0.62
Aggregations using GARP			
Supplier (3)	0.79	0.79	0.55
Supplier (4)	0.77	0.78	0.56
Review (3)	0.82	0.80	0.55
Review (4)	0.79	0.79	0.58
Supplier & Review (3)	**0.83**	0.81	0.56
Supplier & Review (4)	0.80	0.80	0.56

Table 4.11: AUC for integrated datasets with Ripper (JRip)

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations			
Retail (1)	0.50	0.57	0.58
Aggregations using GARP			
Supplier (3)	**0.61**	**0.61**	0.60
Supplier (4)	0.60	0.59	0.60
Review (3)	0.60	0.59	**0.61**
Review (4)	**0.61**	0.59	**0.61**
Supplier & Review (3)	**0.61**	0.59	0.60
Supplier & Review (4)	**0.61**	**0.61**	0.60

Table 4.12: AUC for integrated datasets with Support Vector Machines (SVM)

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations			
Retail (1)	0.50	0.51	0.52
Aggregations using GARP			
Supplier (3)	0.76	0.70	0.72
Supplier (4)	0.72	0.70	0.73
Review (3)	0.87	0.70	0.79
Review (4)	**0.88**	0.70	0.79
Supplier & Review (3)	0.87	0.70	0.78
Supplier & Review (4)	**0.88**	0.70	0.79

Table 4.13: AUC for integrated datasets with Multilayer Perceptron

Method (Depth)	No sampling	Undersampling	SMOTE
No Aggregations			
Retail (1)	0.64	0.64	0.64
Aggregations using GARP			
Supplier (3)	0.77	0.73	0.76
Supplier (4)	0.77	0.73	0.74
Review (3)	0.82	0.75	0.80
Review (4)	0.82	0.75	**0.83**
Supplier & Review (3)	0.82	0.75	0.82
Supplier & Review (4)	0.82	0.75	0.82

The integration of external data sources also improves the classification accuracy significantly. The Review data source performs better than the Supplier source, because reviews are simulated in a way that the discriminant pattern of bad reviews for returned products is followed by most of the returned products; however, only two manufacturing locations have a specific pattern of returns, one with high returns and one with low returns. In the case of aggregating information from both the databases, accuracy is around the level of Review database. Still there is a benefit in using both datasets which is obvious by looking at best predictors in Table 4.14. Figure 4.12-4.17 and 4.18-4.23 give the comparison of the classification performance with the integration of external data sources at depth 3 and 4 respectively.

The lesson is that we should always try to integrate relevant external data sources rather than limiting our analysis to the available internal database, hence the advantage ot big data.

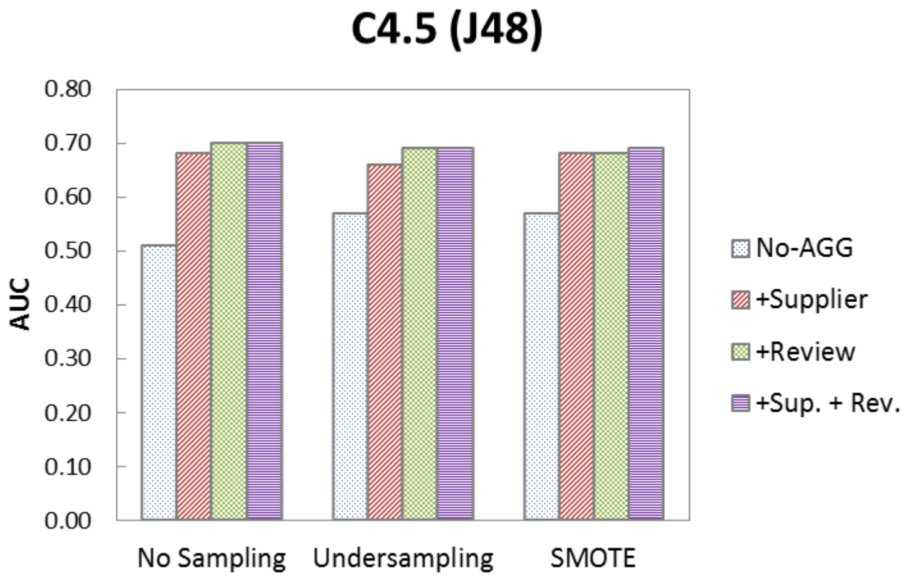


Figure 4.12: AUC comparison for integrated datasets at ‘depth=3’ with C4.5 (J48)

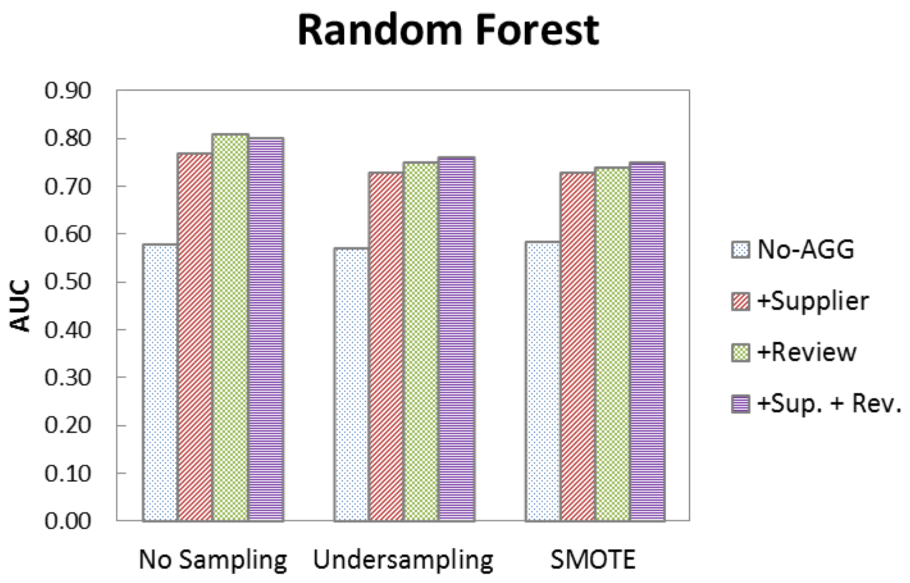


Figure 4.13: AUC comparison for integrated datasets at ‘depth=3’ with Random Forest

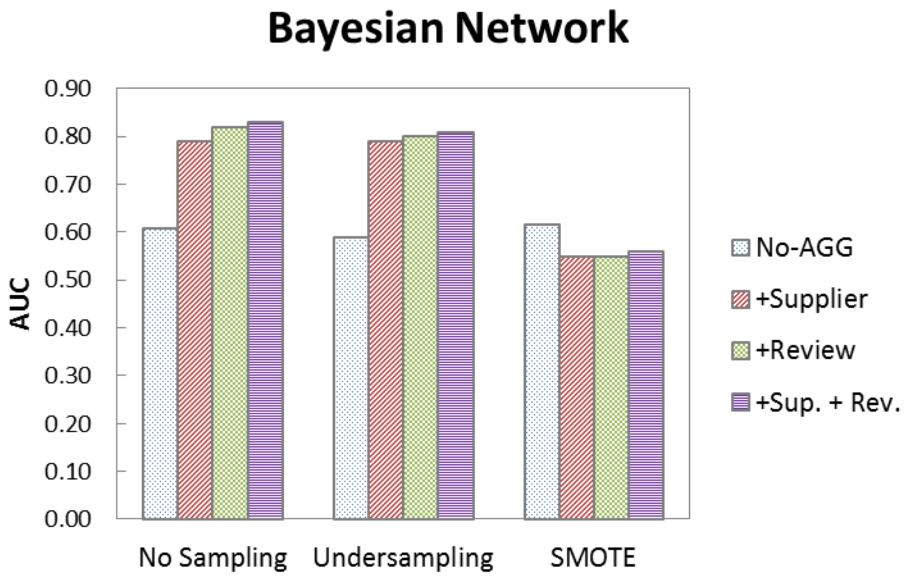


Figure 4.14: AUC comparison for integrated datasets at ‘depth=3’ with Bayesian Network

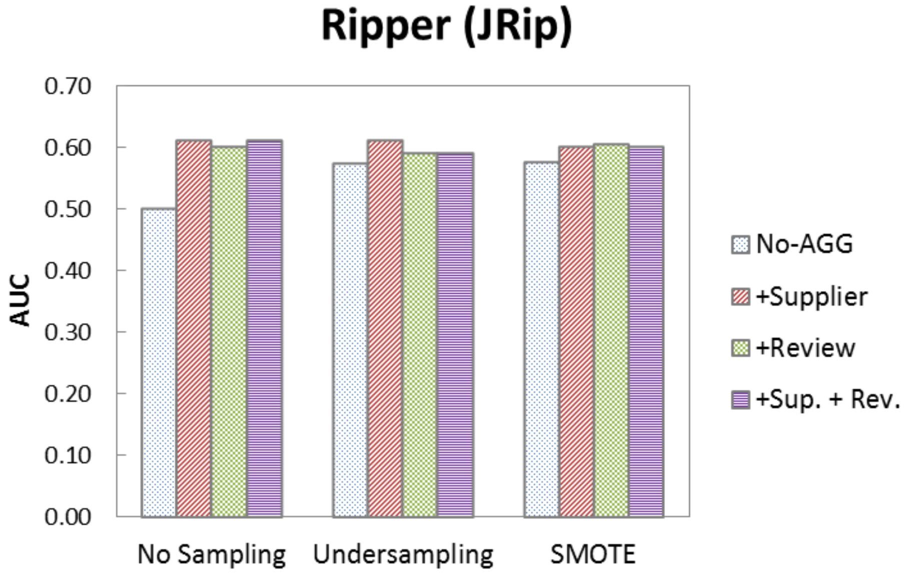


Figure 4.15: AUC comparison for integrated datasets at ‘depth=3’ with Ripper (JRip)

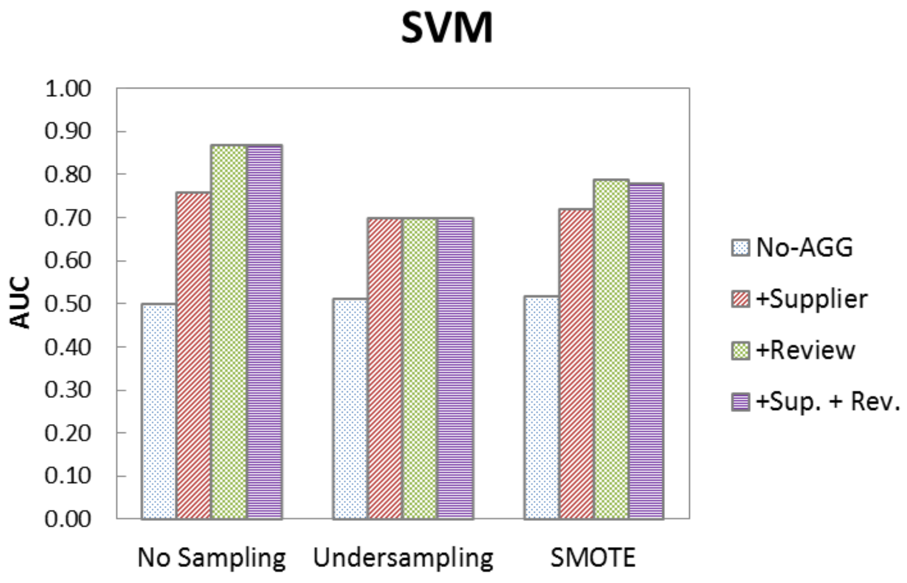


Figure 4.16: AUC comparison for integrated datasets at ‘depth=3’ with Support Vector Machines (SVM)

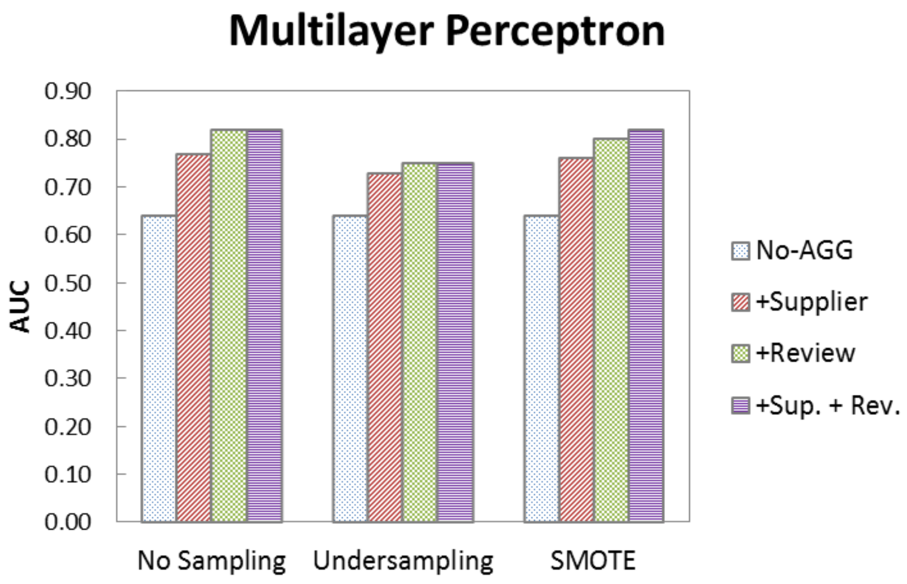


Figure 4.17: AUC comparison for integrated datasets at ‘depth=3’ with Multilayer Perceptron

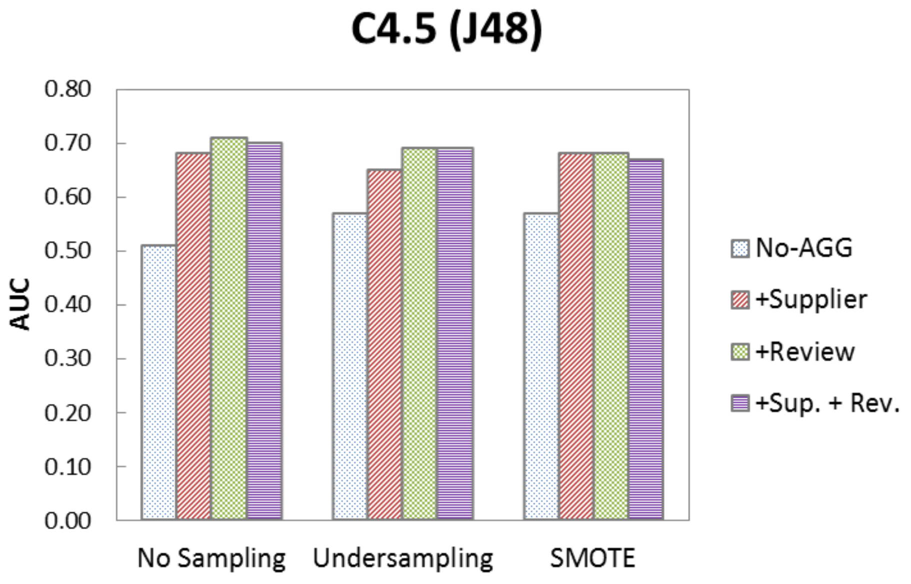


Figure 4.18: AUC comparison for integrated datasets at 'depth=4' with C4.5 (J48)

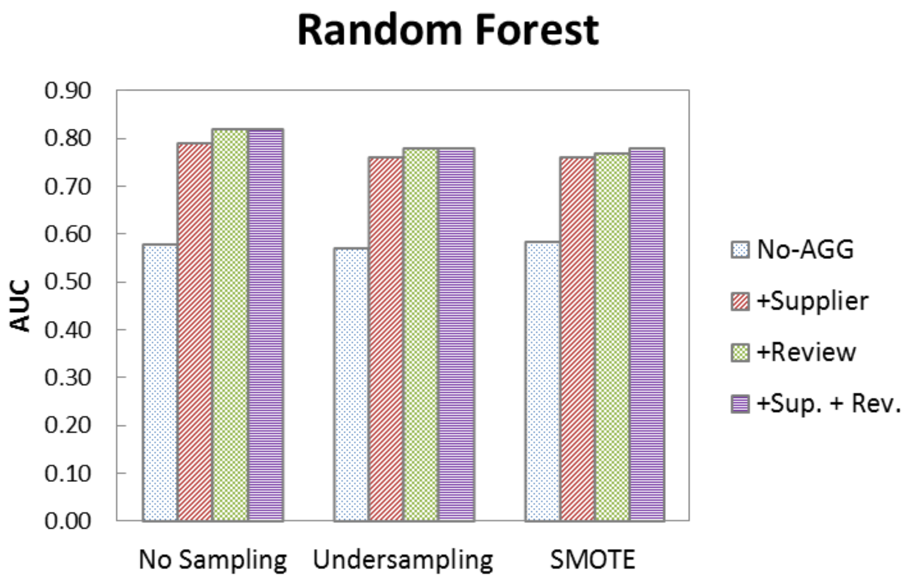


Figure 4.19: AUC comparison for integrated datasets at 'depth=4' with Random Forest

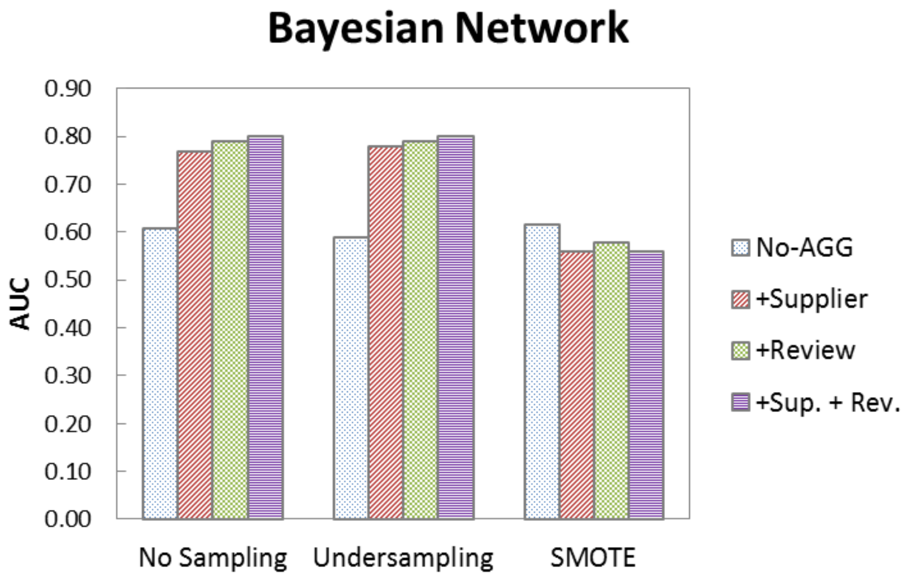


Figure 4.20: AUC comparison for integrated datasets at 'depth=4' with Bayesian Network

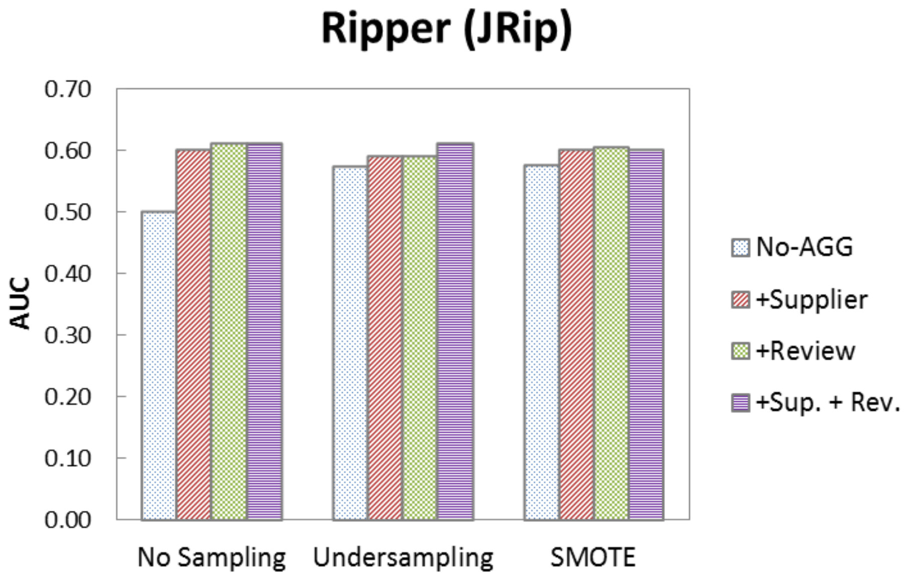


Figure 4.21: AUC comparison for integrated datasets at 'depth=4' with Ripper (JRip)

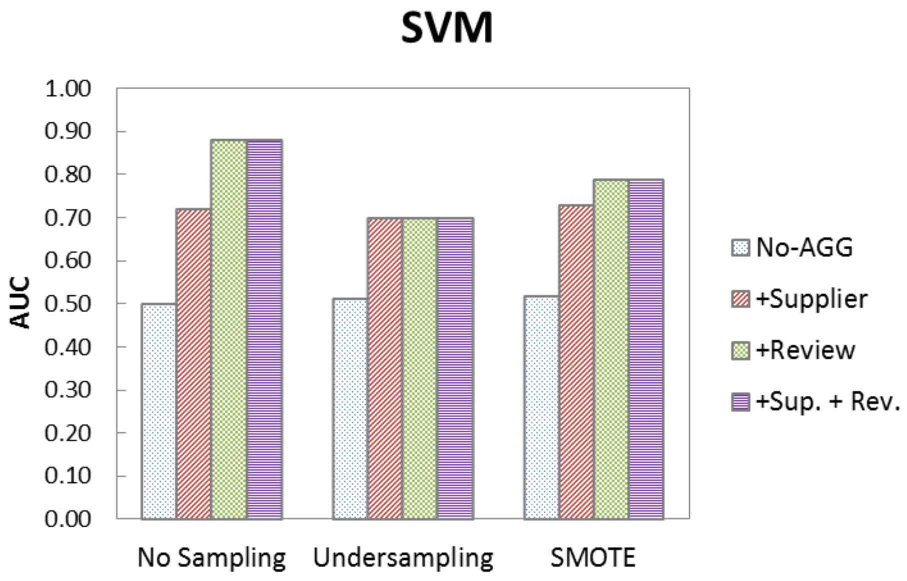


Figure 4.22: AUC comparison for integrated datasets at ‘depth=4’ with Support Vector Machines (SVM)

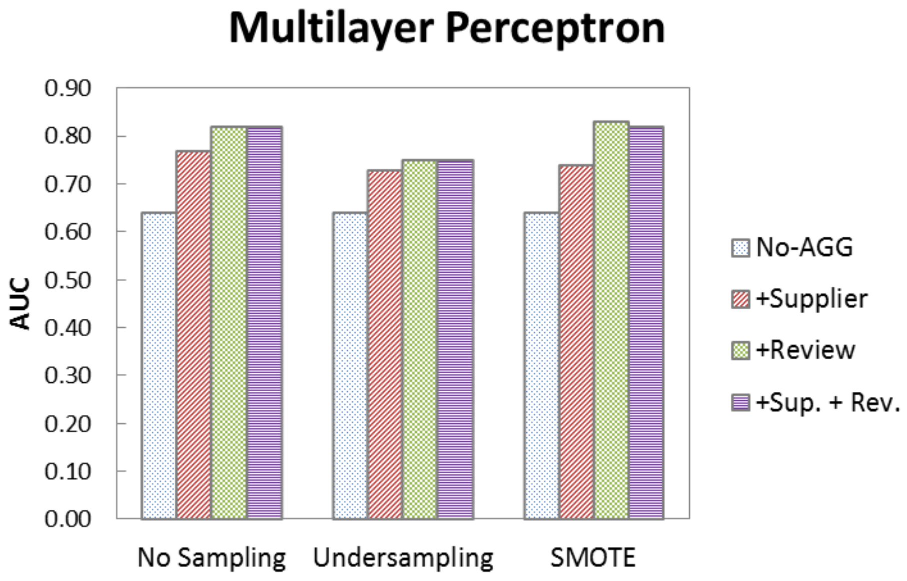


Figure 4.23: AUC comparison for integrated datasets at ‘depth=4’ with Multilayer Perceptron

Table 4.14: Discriminant predictors from External Data Sources

Attributes	Depth
1. Location of supplier (Thailand)	3
2. Avg (stars) among reviews of products	3

From the knowledge discovery perspective, we are interested in additional discriminant predictors generated with the integration of external data sources. Our technique successfully discovers patterns from both datasets automatically. The first attribute in Table 4.14 represents the pattern that was intentionally created while generating synthetic data for the Supplier database. It was automatically rediscovered by GARP which validates our approach. A supplier can have multiple manufacturing locations for their products. In our dataset, products manufactured in *'Thailand'* have a high proportion of returns (hypothetical: injected synthetically). Our method not only found the *'location'* attribute, but also the value that makes it discriminant. This strength of our technique can be very helpful in finding different patterns like a problem with a specific facility of a supplier, defects in the material used for production, issues with the specific version of the product, etc. This is definitely helpful for the stores and retailers as it could help them to deal with suppliers and mutually identify the problems with products.

The second attribute in Table 4.14 comes from the Review database and shows that average star rating of a product can determine whether the product will be returned or not. The synthetic data generated to simulate reviews follow a pattern that products with an average rating around '1–2' stars have high return rates, products with '3' stars can fall on either side and star ratings of '4–5' are given to products with low returns or no returns at all. For some products, we might not have any user ratings. This pattern was created with an intention to simulate a real world situation. Hence, the star rating is a 'true predictor' in our simulated data and our technique found an interpretable attribute for this pattern.

Both attributes were selected among the top ten discriminant predictors when we integrated our Retail database with Supplier and Review data sources. This shows the strong correlation of both the attributes with our class label *return*. Along with the prediction, the attribute from Supplier database provides information about the cause of return, which is related to the manufacturing location. On the other hand, the attribute from the Review table suggests that bad ratings are associated with product returns. The reason for bad ratings might be clear from user comments like the product has a short life span, quality is not up to the mark, better alternatives are available in the market, etc. However, sometimes a reason is not explicitly mentioned in the accompanying user comment. In this case, finding the root cause of return needs further analysis based on this attribute.

Data integration can determine complex patterns without explicit feedbacks. Consider an example of a new product launched in the market. The product performs well with a large number of sales throughout the year and receives good feedback from customers. The next year, the manufacturer decides to change the material of a specific part for the new version of this product from metal to plastic. This results in unsatisfied customers with negative feedback and high product returns. This pattern can be detected with negative user ratings in that specific year and change in manufacturing material. This example shows the value of bringing the pieces together to see the big picture [58].

4.5 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 4.24 reports the time taken to generate new attributes for the flat mining table. The results presented in Figure 4.24 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. We also compare our technique with Dataconda's attribute generation approach at depth level 3. We only show the comparison at depth 3 as Dataconda takes too long for depth 4. Figure 4.25 shows that our suggested improvement to Dataconda performs significantly better than the existing approach.

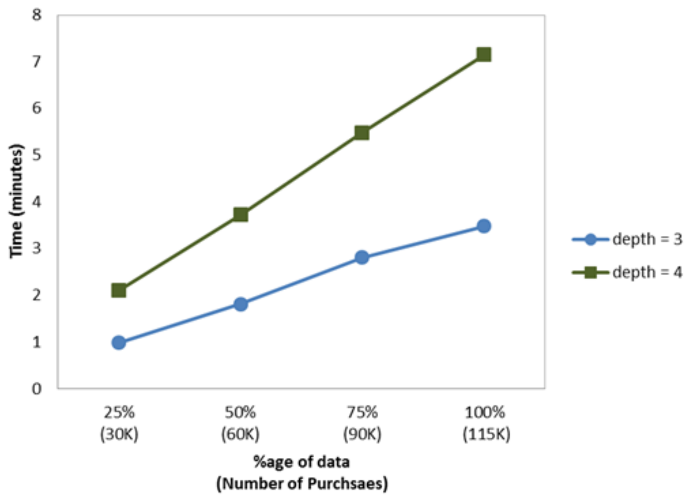


Figure 4.24: Execution times for GARP at depth 3 and 4 by varying the size of the dataset

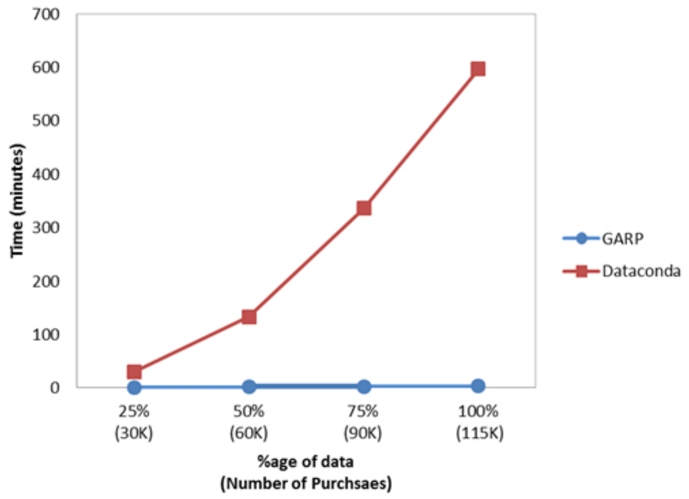


Figure 4.25: Execution times for GARP and Dataconda by varying the size of the dataset at depth 3

Chapter 5

Conclusions and Future Work

5.1 Summary

In this study, we investigate the problem of applying data mining and machine learning techniques on relational datasets. Most of these learning techniques rely on a single table and assume that training data has a balanced class distribution. However, this is rarely true in real applications where data is distributed in multiple tables and often available from additional data sources that can be thought of as external tables in a multi-relational database. Additionally, the class imbalance is also very common in such applications.

In order to resolve the compatibility issues of existing learning techniques and multi-relational datasets, data needs to be aggregated in a single table from all the available sources. This flat mining table can be constructed, based on the knowledge of domain experts, by generating attributes that represent potential predictors. However, the manual process of building a mining table is not very efficient and the attributes are generated based on the existing domain knowledge. On the other hand, automatic generation of attributes by aggregating information from all the available data results in a large feature space which can contain predictors that are unexpected and cannot be determined during the manual process. The generation of discriminative features can also help to reduce the class imbalance problem by differentiating the classes well.

The existing work in Multi-relational Data Mining (MRDM) provides two different alternatives for feature discovery and classification in the relational context.

One approach is to extend the existing learning techniques to handle relational data based on Inductive Logic Programming (ILP). The other approach, as mentioned earlier, is to generate a single table from all the available sources, known as Propositionalization, before applying traditional learning techniques. The Propositionalization approaches generate more rich and specific patterns compared to the ILP approach, especially in the case of numeric data. Our work extends the Propositionalization approach by efficiently generating attributes that represent the past information. All the existing Propositionalization techniques either do not generate such attributes or do not generate them efficiently.

Our technique of Generating Attributes with Rolled Paths (GARP) generates attributes on paths in which a table appears more than one time in a series of joins to aggregate information. The exploration of 'rolled paths' results in attributes that contain useful information from the past. While generating these attributes for the test data, we add dates to assure that attributes are built without using any future information. We also scale this extensive attribute generation process to handle very large amounts of data by introducing a pre-processing step that can help to apply multiple aggregations for similar attributes at a same time.

We empirically demonstrate the effectiveness of our technique, GARP, in improving the classification performance and facilitating the knowledge discovery process with an imbalanced dataset. Our results show that GARP performed better than other approaches without having to alter the data distribution by undersampling or oversampling. In addition, the inclusion of external data sources of Suppliers and Reviews also improves the predictive accuracy and results in detection of additional patterns.

5.2 Conclusions

With our research on automatically constructing a mining table by aggregating information from multiple tables and external data sources, we conclude that:

- The attributes generated by utilizing the past information can help to improve the classification performance.

- The automatic framework for generating attributes facilitates the knowledge discovery process by finding unexpected patterns.

- The discriminant attributes generated by aggregating information from multiple tables and data sources can help in differentiating the classes by reducing the overlap in classes, even in the presence of class imbalance.

- The extensive process of generating attributes from the dataset can be scaled to cope with a large amount of data.

5.3 Future Work

Future research can focus in the following directions:

- 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.

- A future study can be conducted to deal with changes in the database and external data sources. Such changes can occur due to the modification of data or structural revision of the underlying data sources. In such a study, the main focus should be on propagating changes to the mining table without repeating the whole process.

- 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 overall computation time can be further reduced with an efficient approach to distribute the work on several machines. It might seem straightforward to dedicate each machine to generate attributes for a subset of paths. However, it is not trivial to divide the workload without replicating data on different machines as several paths go through the same tables.

Bibliography

[1] Xavier Amatriain. Mining large streams of user data for personalized recommendations. *ACM SIGKDD Explorations Newsletter*, 14(2):37–48, 2012.

[2] Colin Bellinger. *Beyond the Boundaries of SMOTE: A Framework for Manifold-based Synthetic Oversampling*. PhD thesis, University of Ottawa, 2016.

[3] Colin Bellinger, Shiven Sharma, and Nathalie Japkowicz. One-class versus binary classification: Which and when? In *11th International Conference on Machine Learning and Applications (ICMLA)*, volume 2, pages 102–106. IEEE, 2012.

[4] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[5] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[6] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[7] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[8] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[9] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter*, 6(1):1–6, 2004.

[10] X Chen and XIAOTONG Lin. Big data deep learning: Challenges and perspectives. *IEEE Access*, 2:514–525, 2014.

[11] Gilles Cohen, Melanie Hilario, and Christian Pellegrini. One-class support vector machines with a conformal kernel. a case study in handling class imbalance. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 850–858. Springer, 2004.

[12] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[13] David L Donoho. For most large underdetermined systems of linear equations the minimal 1-norm solution is also the sparsest solution. *Communications on pure and applied mathematics*, 59(6):797–829, 2006.

[14] Saso Dzeroski. Multi-relational data mining: An introduction. *ACM SIGKDD Explorations Newsletter*, 5(1):1–16, 2003.

[15] Wei Fan and Albert Bifet. Mining big data: current status, and forecast to the future. *ACM SIGKDD Explorations Newsletter*, 14(2):1–5, 2012.

[16] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[17] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[18] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[19] Peter E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, IT-14:515–516, 1968.

[20] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[21] Kathryn Hempstalk, Eibe Frank, and Ian H Witten. One-class classification by combining density and class probability estimation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 505–519. Springer, 2008.

[22] James D Hess and Glenn E Mayhew. Modeling merchandise returns in direct marketing. *Journal of Interactive Marketing*, 11(2):20–35, 1997.

[23] Robert C Holte, Liane E Acker, and Bruce W Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the 11th international joint conference on Artificial intelligence*, pages 813–818. Morgan Kaufmann Publishers Inc., 1989.

[24] IBM. The four v's of big data. `http://www.ibmbigdatahub.com/infographic/four-vs-big-data`, 2015.

[25] Nathalie Japkowicz. Supervised versus unsupervised binary-learning by feedforward neural networks. *Machine Learning*, 42(1-2):97–122, 2001.

[26] Keith Knight and Wenjiang Fu. Asymptotics for lasso-type estimators. *Annals of statistics*, pages 1356–1378, 2000.

[27] Arno Knobbe, Hendrik Blockeel, Arno Siebes, and Daniel van der Wallen. Multi-relational data mining. 1999.

[28] Arno J Knobbe, Marc De Haas, and Arno Siebes. Propositionalisation and aggregates. In *Principles of Data Mining and Knowledge Discovery*, pages 277–288. Springer, 2001.

[29] Stefan Kramer, Nada Lavrač, and Peter Flach. *Propositionalization approaches to relational data mining*. Springer, 2001.

[30] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *In Proceedings of the Fourteenth International Conference on Machine Learning*, 1997.

[31] Nada Lavrac and Saso Dzeroski. Inductive logic programming. *WLP*, pages 146–160, 1994.

[32] John Lees-Miller, Fraser Anderson, Bret Hoehn, and Russell Greiner. Does wikipedia information help netflix predictions? In *Seventh International Conference on Machine Learning and Applications (ICMLA)*, pages 337–343. IEEE, 2008.

[33] Der-Chiang Li, Chiao-Wen Liu, and Susan C Hu. A learning method for the class imbalance problem with medical data sets. *Computers in biology and medicine*, 40(5):509–518, 2010.

[34] Charles X Ling and Victor S Sheng. Cost-sensitive learning. In *Encyclopedia of Machine Learning*, pages 231–235. Springer, 2011.

[35] Bing Liu. *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge University Press, 2015.

[36] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(2):539–550, 2009.

[37] Marcus A Maloof. Learning when data sets are imbalanced and when costs are unequal and unknown. In *ICML-2003 Workshop on Learning from Imbalanced Data Sets*, 2003.

[38] Jian Ni, Scott A Neslin, and Baohong Sun. Database submission-the isms durable goods data sets. *Marketing Science*, 31(6):1008–1013, 2012.

[39] Claudia Perlich and Foster Provost. Distribution-based aggregation for relational learning with identifier attributes. *Machine Learning*, 62(1-2):65–105, 2006.

[40] J Andrew Petersen and V Kumar. Are product returns a necessary evil? antecedents and consequences. *Journal of Marketing*, 73(3):35–51, 2009.

[41] Foster Provost and Tom Fawcett. Robust classification for imprecise environments. *Machine learning*, 42(3):203–231, 2001.

[42] Foster Provost and Tom Fawcett. Data science and its relationship to big data and data-driven decision making. *Big Data*, 1(1):51–59, 2013.

[43] J Ross Quinlan. *C4. 5: Programs for machine learning*. Morgan Kaufmann Publishers, 1993.

[44] Saharon Rosset, Claudia Perlich, Grzergorz Świrszcz, Prem Melville, and Yan Liu. Medical data mining: insights from winning two competitions. *Data Mining and Knowledge Discovery*, 20(3):439–468, 2010.

[45] Michele Samorani. Automatically generate a flat mining table with dataconda. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1644–1647. IEEE, 2015.

[46] Michele Samorani, Manuel Laguna, Robert Kirk DeLisle, and Daniel C Weaver. A randomized exhaustive propositionalization approach for molecule classification. *INFORMS Journal on Computing*, 23(3):331–345, 2011.

[47] Warren S Sarle. Neural networks and statistical models. In *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, pages 1538–1550. Citeseer, 1994.

[48] Shiven Sharma, Colin Bellinger, Nathalie Japkowicz, Rodney Berg, and Kurt Ungar. Anomaly detection in gamma ray spectra: A machine learning perspective. In *2012 IEEE Symposium on Computational Intelligence for Security and Defence Applications*, pages 1–8. IEEE, 2012.

[49] Herbert Shear, Thomas Speh, and James Stock. Many happy (product) returns. *Harvard business review*, 80(7):16–17, 2002.

[50] DMJ Tax. *One-class classification*. PhD thesis, TU Delft, Delft University of Technology, 2001.

[51] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[52] Ivan Tomek. Modifications of cnn. *IEEE Trans. Systems, Man and Cybernetics*, 6(11):769–772, 1976.

[53] Chun-Wei Tsai, Chin-Feng Lai, Ming-Chao Chiang, and Laurence T Yang. Data mining for internet of things: A survey. *Communications Surveys & Tutorials, IEEE*, 16(1):77–97, 2014.

[54] Byron C Wallace, Kevin Small, Carla E Brodley, and Thomas A Trikalinos. Class imbalance, redux. In *11th International Conference on Data Mining*, pages 754–763. IEEE, 2011.

[55] Shuo Wang and Xin Yao. Multiclass imbalance problems: Analysis and potential solutions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(4):1119–1130, 2012.

[56] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[57] Qiang Yang and Xindong Wu. 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, 5(04):597–604, 2006.

[58] Osmar R Zaïane. Rich data: Risks, issues, controversies & hype. In *2nd Annual International Symposium on Information Management and Big Data - SIMBig, Cusco, Peru, Sep 2-4, 2015, CEUR-WS.org, online CEUR-WS.org/Vol-1478*, pages 21–24.

[59] Peng Zhao and Bin Yu. On model selection consistency of lasso. *Journal of Machine Learning Research*, 7(Nov):2541–2563, 2006.