An Abstract Algebra for Knowledge Discovery in Databases

Luciano Gerber and Alvaro A. A. Fernandes

Department of Computer Science
University of Manchester
Oxford Road, Manchester M13 9PL, UK
{l.gerber|a.fernandes}@cs.man.ac.uk

Abstract. Knowledge discovery in databases (KDD) plays an important role in decision-making tasks by supporting end users both in exploring and understanding of very large datasets and in building predictive models with validity over unseen data. KDD is an ad-hoc, iterative process comprising tasks that range from data understanding and preparation to model building and deployment. Support for KDD should, therefore, be founded on a closure property, i.e., the ability to compose tasks seamlessly by taking the output of a task as the input of another. Despite some recent progress, KDD is still not as conveniently supported as end users have reason to expect due to three major problems: (1) lack of task compositionality, (2) undue dependency on user expertise, and (3) lack of generality. This paper contributes to ameliorate these problems by proposing an abstract algebra for KDD, called **K-algebra**, whose underlying data model and primitive operations accommodate a wide range of KDD tasks. Such an algebra is a necessary step towards the development of optimisation techniques and efficient evaluation that would, in turn, pave the way for the development of declarative, surface KDD languages without which end-user support will remain less than convenient, thereby damaging the prospects for mainstream acceptance of KDD technology.

1 Introduction

In spite of great interest and intense research activity, the field of knowledge discovery in databases (KDD) is still far from benefiting from unified foundations. Research focus has been patchy by and large. Efforts are all too often devoted to devising new, or extending existing, algorithms that are very specifically tailored to very specific contexts, and hence not as widely applicable as one would hope, given the large variety of application scenarios in which KDD could have a significant impact.

At present, there is no foundational framework that unifies the different representations in use, let alone a well-founded set of generic operations that both cohere, when taken together, and are flexible enough to express a significant number of complex KDD tasks in a convenient way. The lack of such foundations is a significant impediment in the way of mainstream acceptance, and of wider deployability, of KDD technologies. This is because without such foundational framework it is very difficult to develop declarative, surface KDD languages without which end-user support may always remain less than convenient.

This is very much reflected in the difficulties data analysts face in interacting with KDD systems. KDD is an ad-hoc, iterative process comprising tasks that range from data understanding and preparation to model building and deployment. The typical flow of work in KDD projects requires end-users (i.e., data analysts and knowledge engineers) to operate a KDD suite that, particularly with respect to the data mining phase, acts as a front-end to a collection of algorithms. Unfortunately, each algorithm tends to be very specialised, in the sense that it produces optimal outcomes only under very narrow assumptions and in very specific circumstances. Moreover, different algorithms tend to depend on specific representations of their inputs and outputs, and this often requires the end user carrying out representational mappings in order for tasks to compose in a flow of work. More importantly, if all one has is a suite of algorithms, then end users bear the ultimate responsibility for overall success, since they must choose the most appropriate algorithm at each and every step of the KDD process. Bad choices may well delay or derail the process. Given that the KDD literature has given rise to a myriad different, very specific algorithms for each phase of the KDD process, and given that sometimes the differences are subtle in the surface but with significant import in terms of effectiveness and efficiency, the choices faced by end users could be bewildering. Thus, it can be seen that the state of the art in KDD still suffers from some significant shortcomings. The main issues identified above are:

1. *Lack of compositionality*, insofar as contemporary KDD suites fail to achieve seamless composition of tasks. Inputs and outputs have different representations (e.g., different file format) and the need to map across them hinders the transparency of the process. Moreover, due in part to type mismatches, there is a lack of orthogonality in the way the tasks are composed. For example, while it is possible to apply selection to (i.e., take a subset of) a dataset, that same generally does not apply to a decision tree (i.e., choose a subset of branches), despite being naturally reasonable to do so. The challenge here is to find a unified representation for KDD inputs and outputs.
2. *Dependency on user expertise*, insofar as there is a large number of candidate algorithms at almost every step, for almost every task, and each in turn has several parameters which require expertise in being set if good results are to be obtained. Often these choices affect considerably not only how satisfactory the results are but also how efficiently they are obtained. Typically, these decisions are far too complex to be undertaken by casual users. The challenge here is to find specification mechanisms that are amenable to formal manipulation by an optimisation algorithm so as to protect the end user as much as possible from inadvertent choices.
3. *Lack of generality*, insofar as it would be much more convenient if the body of algorithms proposed in the literature were better structured and if abstract generalisations were available that could help the selection of concrete instances. It has been observed empirically that, despite the large number

of different algorithms proposed for similar tasks, they do seem to be based on the same basic principles, differing in very particular respects that make them more applicable to certain contexts than others. The challenge here is to find abstractions that are comprehensive enough to unify the concrete, specialist algorithms that comprise the bulk of the KDD literature. This would allow the basic principles to be more vividly understood and make it possible to define, and reason about, a wide range of KDD tasks more systematically, and even mechanically.

A *unifying formal framework for KDD task composition* is a possible, concerted response to the challenges identified. As a concrete candidate for such formal framework this paper contributes the **K-algebra**, an abstract algebra for KDD whose underlying data model and primitive operations accommodate a wide range of KDD tasks. The K-algebra is a database algebra, i.e., its underlying concern is with flow throughput in situations where the volume of input at the leaves of the computation are significant enough to be the dominant factor in the efficiency of the latter. It must be stressed that the K-algebra is an *abstract algebra*, i.e., it is proposed as a candidate intermediate representation between surface languages and executable plans. It is best understood, therefore, as the proposal of an abstract machine that paves the way for systems to take decisions on behalf of the users as to how complex KDD tasks are configured and composed, similarly to what current query optimisers deliver for database systems. It is not a proposal for dealing directly with concrete effectiveness and efficiency concerns: this is what the bulk of the KDD literature is already doing so commendably.

This paper, therefore, takes the view that defining the foundations for a KDD language that can be subject to optimisation is more likely to succeed than contributing yet another too-specific tool to an already very large tool set. The motivation for our approach stems from the observation that database technology was once limited to a comparable motley collection of poorly related techniques, and the same consequences were noticed then. A historical breakthrough was the development, by Codd, of a unifying formal foundation as an abstract algebra, which was shown to be expressive enough to express complex combination of tasks, that were mappable to the concrete algorithms and techniques. The subsequent introduction of declarative languages empowered end users to query databases in an ad-hoc manner, expanding greatly the possibilities for query retrieval. Ultimately, this was all made concrete with the development of query optimisers, which can take advantage of algebraic properties to seek efficient plans to evaluate the queries over databases. This algebraic approach proved successful for other database models and languages developed subsequently, such as to object-oriented [4] and spatial [8] data, and, more recently, to XML [11] and streams [16], to name a few.

Other researchers have had similar motivations, and it seemed natural to them to consider expressing KDD tasks in a standard database language, for which a robust concretely implemented counterpart exists. However, significant technical impediments stand in the way. For example, standard database languages only have limited capability (and often none at all) to express recursion or iteration, which seem essential in KDD. For example, it is shown by Sarawagi et al. [15] that the implementation in SQL of Apriori, a classical scalable algorithm for association rule discovery, falls short in terms of performance. A similar algorithm is expressed in terms of an extended relational algebra by Ceri et al. [13]. However, it is not clear whether this approach would apply to other more challenging KDD tasks.

To illustrate the K-algebra, the paper includes a specification of a top-down decision-tree induction process in abstract-algebraic form. To the best of our knowledge, this is the first such formulation. In order to have a concrete example, the next subsections make use of the simple dataset [14] in Fig. 1. Briefly, it records weather conditions over a time period (say, days) and labels each instance with the information as to whether a particular sport was played in that day.

ID	outlook	temp	humidity	windy	play?
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rainy	mild	high	false	yes
5	rainy	cool	normal	false	yes
6	rainy	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rainy	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rainy	mild	high	true	no

Fig. 1. The Weather Dataset

The general goal is to use such a labelled dataset to construct a decision tree that, based on the values of certain of the attributes (e.g., outlook and windy) can determine the value of the desired attribute (called the *class*) play. In other words, the decision tree hypothesises a label given an non-labelled instance. Fig. 2 depicts the decision tree that Quinlan's ID3 [14] outputs given the dataset in Fig. 1. The remainder of this paper is structured as follows. Section 2 describes the K-algebra informally. Section 3 shows how it can express top-down decision-tree induction. Section 4 briefly discusses related work. Section 5 draws some conclusions.

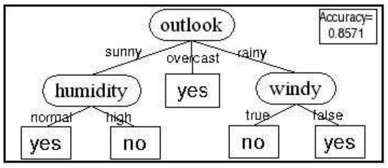

Fig. 2. Example Decision Tree

2 The K-Algebra: An Abstract Algebra for KDD

2.1 The Key Underlying Concepts

The K-algebra is based on an abstract view of the KDD process as consisting of arbitrarily complex compositions of steps that *expand*, *compress*, *evaluate*, *accumulate*, *filter out* and *combine*, in a *iterative* fashion, *collections* of *summaries of data*. The italicised terms in the previous sentence denote the key underlying concepts of the K-algebra. They underpin the expression of a wide range of KDD tasks.

The K-algebra has two carriers: *K-cubes*, which are annotated multi-dimensional structures to represent individual summaries; and *K-cube-lists*, which, unsurprisingly, collect K-cubes into lists, in order to support iterative collection-at-a-time processing of K-cubes.

The typical strategy for specifying KDD tasks in the K-algebra is to obtain a most-detailed K-cube from which further candidates are derived, directly or indirectly. At each iteration, this base K-cube is often used to give rise to derived ones through the application of operations that either *expand* or *compress* a K-cube into one or more K-cubes. By 'expanding' (resp., 'compressing') is meant applying operations that, given a K-cube, increase (resp., decrease) its dimensionality or its cardinality, to yield a (usually slightly) different summarised perspective on the data. These derived summaries are *evaluated*, as a result of which they are annotated with one or more measures of interestingness [1].

The interesting ones, i.e., those who meet or exceed a given threshold, are *accumulated*, whereas the remaining ones are either refined further or *filtered out*. Alternatively, the K-cubes in the current accumulation may be *combined* (assembled back) into a single K-cube for further exploration.

2.2 The K-Algebra Carriers

The fundamental carrier of the K-algebra is a set of structures referred to as *K-cubes*. It is designed to unify the inputs and outputs of a range of KDD tasks. Informally, a K-cube is a hypercube to which is associated a vector of attributes called *annotations*. A *hypercube* is an aggregated representation of extensional data in a multi-dimensional space, in which attributes called *dimensions* play the roles of axes. The (discrete) dimensional domains divide the space into units called *cells*. Attributes called *measures* represent properties, usually aggregated values, of the data inside cells. For example, a K-cube representing a contingency table from the weather dataset is depicted in Fig. 3 in its flat representation. The attributes outlook and play are dimensions. Absolute frequency, denoted by af, is an *aggregate measure*. These are usually associated with functions on *sets of values* such as Sum, Max and Min. In contrast, rf is a *scalar measure* denoting the relative frequency obtained by a function on scalars (in this case, $rf \leftarrow af/aftotal$, where $aftotal = 14$). Each row in the (flat representation of a) K-cube corresponds to a cell.

outlook	play	af	rf
overcast	yes	4	0.286
rainy	no	2	0.143
rainy	yes	3	0.214
sunny	no	3	0.214
sunny	yes	2	0.143
aftotal: 14			

Fig. 3. Example K-Cube

Annotations decorate the summarised information (represented as a hypercube) with *interestingness* values. For example, here, it might range from grand totals and global measures, such as the total frequency aftotal in Fig. 3, to more specific measures such as the predictive accuracy of a decision tree. As it is shown

[1] Note that, here, the concept of interestingness is inspired by, but more loosely construed than is the case in, [3].

later, in the K-algebra, annotations result from the computation of measures. K-cubes are collected in *K-cube-lists*, which are the second carrier of the K-algebra. There is no significant import on using lists as the concrete collection type: they are chosen because they simplify the presentation of those K-algebraic functions that operate on collections of K-cubes.

2.3 The K-Algebraic Operations

Recall that the K-algebra was said to be based on an abstract view of the KDD process as consisting of arbitrarily complex compositions of steps that *expand*, *compress*, *evaluate*, *accumulate*, *filter out* and *combine*, in a *iterative* fashion, *collections* of *summaries of data*. The previous subsection threw some light into what is meant by 'collections of summaries of data'. This subsection presents the operations that support the above steps and that provide the means for iterating over intermediate results. The full formal definition [7] is not given here for reasons of space.

Figs. 4 and 5 list the signatures for the operations and organise them in categories. In Figs. 4 and 5, the intended domains and ranges are as follows: N denotes a finite segment of the natural numbers. K is instantiated by a single K-cube, L by a K-cube-list. For example, the signatures of the operations in Fig. 4(g) reflect the conversion of a K-cube into a K-cube-list. D, M and A are, respectively, instantiated by a set of names of dimensions, measures and annotations. For example, generalise (Fig. 4(a)) requires the identification of the dimensions to be removed from a K-cube. In turn, G denotes generators that assign the result of an expression, usually involving scalar and aggregate elements, to an attribute. This is illustrated by the signature of the operation add-scalar-measure in Fig. 4(c), where G represents the association of a new measure with an expression as a composition of scalar functions and values. F is instantiated by an aggregation function, such as Max or Min, applied on a particular measure. V and E apply to higher-order functions only. The former denotes a binding variable to K-cube elements in the lists being processed, whereas the latter represents an algebraic expression (over the binding variables) to be applied to a K-cube-list. Finally, R is concerned only with bridge operations: it is instantiated by a relation in the extensional layer, as explained later in this subsection.

To see the intended structure in Fig. 4, note that Fig. 4(a) lists the operations on dimensions, Fig. 4(b) those on annotations, Fig. 4(c) those on measures, Fig. 4(d) those on attributes, and, finally, Fig. 4(e) those on cells. As expected, the basic design is that one can add, remove, convert and rename dimensions (e.g., specialise, generalise), annotations and measures (e.g., add-aggregate-measure, add-scalar-measure, copy-mea-as-ann). The first four categories comprise operations that reshape a K-cube without direct reference to its cells. These operations make a K-cube grow or shrink in dimensionality, or be evaluated by more or less measures, or be annotated in more or less ways. The operations in Fig. 4(b) provide means to assign a different role to a particular attribute in a K-cube. For example, copy-dim-as-mea adds a new measure, to a K-cube, whose

specialise	$\lhd : K \times K \times 2^D \times 2^G \rightarrow K$
generalise	$\rhd : K \times 2^D \times 2^F \rightarrow K$
rename-dimension	$\nu^d : K \times 2^G \rightarrow K$

(a) Manipulating Dimensions

remove-annotation	$\rho^a : K \times 2^A \rightarrow K$
rename-annotation	$\nu^a : K \times 2^G \rightarrow K$

(b) Manipulating Annotations

add-scalar-measure	$\epsilon : K \times 2^G \rightarrow K$
add-aggregate-measure	$\alpha : K \times 2^D \times 2^G \rightarrow K$
remove-measure	$\rho^m : K \times 2^M \rightarrow K$
rename-measure	$\nu^m : K \times 2^G \rightarrow K$

(c) Manipulating Measures

copy-mea-as-dim	$\overset{m \rightarrow d}{\kappa} : K \times 2^G \rightarrow K$	copy-dim-as-mea	$\overset{d \rightarrow m}{\kappa} : K \times 2^G \rightarrow K$
copy-ann-as-mea	$\overset{a \rightarrow m}{\kappa} : K \times 2^G \rightarrow K$	copy-mea-as-ann	$\overset{m \rightarrow a}{\kappa} : K \times 2^G \rightarrow K$

(d) Copying Attributes

select	$\sigma : K \times P \rightarrow K$	select-n	$\lambda : K \times N \rightarrow K$
union	$\cup : K \times K \rightarrow K$	intersection	$\cap : K \times K \rightarrow K$
difference	$- : K \times K \rightarrow K$	cartesian-product	$\times : K \times K \rightarrow K$
natural-join	$\bowtie : K \times K \rightarrow K$	union-join	$\overset{\cup}{\bowtie} : K \times K \rightarrow K$
theta-join	$\bowtie : K \times K \times P \rightarrow K$	select-kcube-on-ann	$\mathsf{If} : K \times K \times P \rightarrow K$

(e) Manipulating Cells

map^1	$\mu : V \times L \times E \rightarrow L$	map^2	$\mu^2 : V \times V \times L \times L \times E \rightarrow L$
map^{2x}	$\overset{\times}{\mu} : V \times V \times L \times L \times E \rightarrow L$	reduce	$\Delta : V \times V \times L \times E \rightarrow L$
filter	$\phi : V \times L \times E \rightarrow L$	flatten	$\downdownarrows : L \rightarrow L$
iterate	$\circlearrowright : V \times P \times N \times L \times E \rightarrow L$		

(f) Higher-Order

partition-into-n	$\pi^n : K \times N \rightarrow L$	partition-by-dim	$\pi^d : K \times 2^D \rightarrow L$
gen-subsets	$\bigtriangledown : K \times 2^G \rightarrow L$	gen-supersets	$\bigtriangleup : K \times K \times 2^G \times 2^F \rightarrow L$

(g) Expanding a K-Cube into a K-Cube-List

Fig. 4. K-Algebraic Operations (1)

relation-to-kcube	$\boxplus : R \times 2^D \times 2^G \to K$	kcube-to-relation	$\square : K \to R$
kcube-to-kcube-list	$[\,] : K \to L$	kcube-list-to-kcube	$\boxdot : L \to K$

(a) Bridging Between Layers

Fig. 5. K-Algebraic Operations (2)

values are replicated from those of an existing dimension. In contrast, the last category (in Fig. 4(e)) comprises operations that make direct reference to the cells of K-cube, and this explains the analogies with classical database-algebraic operations (e.g., select, select-n, and natural-join). This is indicative of the fact that the K-algebra can be seen to operate on (possibly aggregated) data (as in standard database algebras) but, in addition to that, it wraps that data with metadata of interest for the identification of descriptive and predictive models of the data.

The other carrier structure employed in the K-algebra is the *K-cube-list*, which collects K-cubes in (possibly nested) lists. To see the intended structure, note firstly that Fig. 4(f) lists operations that either iterate over a K-cube-list or are higher-order functions that transform a K-cube-list in some way (e.g., by mapping, by filtering, by flattening, or by folding). Fig. 4(g) lists operations that expand a single K-cube into a K-cube-list, either by partitioning it (e.g., partition-into-N, partition-by-dim) or by replacing it by its super- or subsets (e.g., gen-supersets, gen-subsets) in a controlled manner. Finally, all the operations so far could be said to be *intra-carrier operations*. However, the K-algebra is a layered algebra in the sense that closure is ensured by *inter-carrier operations* that bridge across layers.

The layers are as follows. The *extensional layer* represents the extensional data from which K-cubes are initially obtained and to which K-cubes are eventually deployed. The carrier is that of the relational model; the operations stem from an relational algebra extended with grouping and aggregation [5]. The *K-cube layer* comprises those operations that manipulate K-cubes one-at-a-time. The carrier is the K-cube. The *K-cube-list layer* provides higher-order and operations that manipulate lists of K-cubes. The carrier is the K-cube-list. Thus, the operations in Fig. 5(a) allow values to move between layers. This is necessary for deriving summaries from, and deploying models in, the extensional data layer, as well as for moving from K-cubes to K-cube-lists.

The case study in Section 3 concretely illustrates all the most important operations introduced in Figs. 4 and 5.

3 A Concrete Example: Decision-Tree Induction

This section applies the K-algebra to a concrete example of building an ID3-like classifier. Its subsections provide a quick overview of top-down decision tree building algorithms, an overview of the algorithmic strategy for constructing the classifier in K-algebraic fashion, and how the K-algebraic concepts employed correspond to the original algorithmic template for the kind of classifiers addressed.

3.1 Top-Down Decision Tree Inducers

A decision tree for the weather dataset was introduced in Fig. 2. Decision-tree models are the result of a separate-and-conquer approach and classify instances into classes. The attribute play constitutes the class one wants to predict based on the other attributes, in the given example. The branches of the tree define partitions of data and label the instances that traverse them with the class values at their leaves. For example, when applied to the weather dataset, it determines that all instances in which outlook has the value overcast, the class value (i.e., play) be yes.

A classical top-down decision tree algorithm is *ID3* [14]. It constructs a decision tree from a training dataset by recursively partitioning the data, using the attributes but the class, until it is discriminating enough with respect to the class values. Ascertaining the latter requirement relies on a measure of the purity of a partition (such as the *information gain ratio* employed in ID3-like techniques or the *Gini index* used in *CART* [1]) which essentially reflects the variability of the class distribution. The algorithm proceeds in a depth-first fashion, using a greedy heuristic strategy to decide on the best splits at each node of the tree. A high-level procedural specification of ID3 is shown in Algorithm 1.

Algorithm: ID3(T, D, b, A)

T: the tree being built
D: the current partition of data
b: the current branch of the tree
A: the set of attributes to possibly split D further

if D *is pure or* $A = \emptyset$ **then**
    Build class node for D under b
else
    Using the information gain, determine the best attribute $a \in A$ to split D
    Create a node n for a and add it to T
    Create and add to T a branch b_i for each value v in $dom(a)$
    for *each branch* b_i **do**
        ID3(T, D_i, b_i, A) `# where` D_i `is the subset of` D `determined by` b_i

Algorithm 1: High-Level Procedural Specification of ID3

3.2 An Overview of the Algorithmic Strategy

In this subsection, the K-algebraic approach to an ID3-like classifier is presented. The algorithmic strategy used for that consists of three phases, viz., *preparation*, *iteration* and *combination and deployment*. Due to lack of space, the description here is rather high-level and is only meant to illustrate how the key concepts of the algebra are concretely applied to a classical, challenging KDD task. The outcome is the construction of (a K-cube that is operationally equivalent to) a decision-tree model. Some of the most important operations employed are discussed in somewhat more detail.

Section 2 described the general algorithmic strategy for specifying KDD tasks in the K-algebra. For the decision tree inducer, that strategy takes the following concrete form: (1) the preparation phase constructs the most detailed K-cube (K_{base}), which other computations will elaborate upon, and the initial K-cube (K_{class}) to be input into the iterative part, which provides sufficient aggregate information with respect to the dimension (play) for further evaluation of interestingness; (2) the iterative part expands K_{class} into different specialisations and partitions, evaluates interestingness (based on the Gini index, in this case), and, at each step, accumulates those with top interestingness (i.e., smallest Gini index); finally, (3) the combination and deployment phase assembles back the resulting accumulated K-cubes into one that is the desired (classification) model (K_{model}), and deploys it at the extensional layer using the appropriate bridge operation.

Phase 1: Preparation Algorithm 2 gives the K-algebraic expressions for the preparation phase. Its purpose and intermediate steps are as follows.

Algorithm: **K-Algebraic Expressions for Preparation**

// obtain the most detailed K-cube from the extensional data

$K_{base} \leftarrow \boxplus_{\langle outlook, humidity, windy, play\rangle, \langle af:\, Count()\rangle}(weatherR)$

$L_{base} \leftarrow [K_{base}]$

// obtain the K-cube with frequency distribution for the dimension play (the class)

$K_1 \leftarrow \alpha_{\langle play\rangle, \langle af: Sum(af)\rangle}(K_{base})$

$K_{class} \leftarrow \rhd^{\langle\rangle}_{*\setminus\langle play\rangle}(K_1)$

$L_{class} \leftarrow [K_{class}]$

Algorithm 2: The Preparation Phase

Firstly, the most detailed K-cube K_{base} (Fig. 6(a)) is obtained by means of the corresponding bridge operation relation-to-K-cube (denoted by $\boxplus$). This is an essential component of the algebraic strategy, as the process typically evolves in an less-to-more-specific fashion. *Specialisations* of K-cubes are obtained at each step, which requires the existence of a more detailed K-cube from which information is recovered. For instance, take a K-cube with dimensions (outlook, play) and a measure of absolute frequency (af). In order to specialise a K-cube into (outlook, humidity, play), it is not possible to de-aggregate the information from the former (i.e., af), unless it is recovered from another K-cube with the information in non-aggregated form. This explains why operations such as specialise and gen-supersets require two K-cubes as arguments: the one to specialise and the one from which information is to be recovered. This K-cube is then converted into in a singleton list L_{base} with the operation K-cube-to-K-cube-list (denoted by $[\ldots]$), for the benefit of list-processing operations in the iterative phase.

This phase also comprises the construction of the initial K-cube to be pumped into the iterative part. In lines 3 to 4, the K-cube K_{class} (Fig. 6(b)) is constructed to yield the class distribution, i.e., the absolute frequency of the dimension play. It is obtained by the combination of an add-aggregate-measure (denoted by α), which computes the aggregate values on partitions of cells determined by the distinct values of play, and a remove-dimension, which removes all the dimensions but play from the newly aggregated K-cube. The K-cube K_{class} is the starting point for the iterative part, which will process specialisations of it with a view towards minimising the Gini index for combinations of the class with other dimensions. The initial list of K-cubes to be processed in the iterative part is assigned to L_{class}, as shown in line 5.

outlook	humidity	windy	play	af
overcast	high	false	yes	1
overcast	high	true	yes	1
overcast	normal	false	yes	1
overcast	normal	true	yes	1
rainy	high	false	yes	1
rainy	high	true	no	1
rainy	normal	false	yes	2
rainy	normal	true	no	1
sunny	high	false	no	2
sunny	high	true	no	1
sunny	normal	false	yes	1
sunny	normal	true	yes	1

(a) K_{base}

play	af
no	5
yes	9

(b) K_{class}

Fig. 6. Example Outcome Preparation

Phase 2: Iteration Most of the effort in computing the classifier lies in the iterative phase of this K-algebraic approach. Algorithm 3 presents the corresponding K-algebraic expressions, whose output represented by L_{iter} feeds phase 3.

The iterate operation (denoted by $\circlearrowright$) characterises this phase. It takes as input the initial K-cube-list obtained from the preparation phase (L_{class}), and also the criteria to decide on when to continue accumulating K-cubes or leave the iteration. It is assumed that when a K-cube has the annotation gini evaluated to 0, it shall be accumulated and not specialised further. As an example, the maximum number of iterations is set to 10. When a pass of the iteration is completed, the accumulated K-cubes are assigned to L_{iter}, which is the input to the combination and deployment phase. Fig. 7 shows the intermediate results in the 3-pass iteration that leads to the desired result. (K-cube subscripts suggest which dimensions and subsets of cells led to its derivation.)

K-cubes represent different ways of partitioning the data. The aim of this iterative process is to find those that best discriminate the desired class. The Gini index is a measure of how good each K-cube is in this regard, and is associated with the K-cube in the form of an annotation. All K-cubes processed in this

Algorithm: **K-Algebraic Expressions for Iteration**

// plug in the iterate operation the parameters for the termination condition
// and the initial list to be processed

$L_{iter} \leftarrow \circlearrowright^{L_{current}}_{gini=0,\ 10} (L_{class})($

// generate supersets, of dimensionality one degree higher, of each K-cube

$L_1 \leftarrow \overset{\times}{\mu}^{k_1,k_2}_{\bigtriangledown^{\langle\rangle}_{\langle af:\ Sum(af)\rangle}(k_1,k_2)}(L_{current}, L_{base})$

// evaluate interestingness

$L_2 \leftarrow \mu^{k}_{interest(k)}(L_1)$

// reduce nested lists to choose best of each

$L_3 \leftarrow \mathit{\Delta}^{k_1,k_2}_{\mathrm{If}_{k_1.gini<k_2.gini}(k_1,k_2)}(L_2)$

// partition by dimensions

$L_4 \leftarrow \mu^{k}_{\pi^{d}_{*\setminus\langle play\rangle}(k)}(L_1)$

// evaluate interestingness

$L_5 \leftarrow \mu^{k}_{interest(k)}(L_4)$

// flatten K-cube-list

$L_6 \leftarrow\downdownarrows (L_5)$

)

Algorithm 3: The Iteration Phase

0th Iteration
$L_{current} = [\ K_{class}\ ]$ $L_{iter} = [\ ]$
1st Iteration
$L_2 = [\ [K_{out}, K_{hum}, K_{windy}]\ ]$ $L_3 = [\ K_{out}\ ]$ $L_5 = [\ [\ K_{outra}, K_{outov}, K_{outsu}\ ]\ ]$ $L_6 = [\ K_{outra}, K_{outov}, K_{outsu}\ ]$ $L_{current} = [\ K_{outra}, K_{outsu}\ ]$ $L_{iter} = [\ K_{outov}\ ]$
2nd Iteration
$L_2 = [\ [\ K_{outraHum}, K_{outraWindy}\ ], [\ K_{outsuHum}, K_{outsuWindy}\ ]\ ]$ $L_3 = [\ K_{ourraWindy}, K_{outsuHum}\ ]$ $L_5 = [\ [\ K_{outraWindyf}, K_{outraWindyf}\ ], [\ K_{outsuHumhi}, K_{outsuHumno}\ ]\ ]$ $L_6 = [\ K_{outraWindyf}, K_{outraWindyf}, K_{outsuHumhi}, K_{outsuHumno}\ ]$ $L_{current} = [\ ]$ $L_{iter} = [\ K_{outov}, K_{outraWindyf}, K_{outraWindyf}, K_{outsuHumhi}, K_{outsuHumno}\ ]$

Fig. 7. Iteration Phase: Intermediate Results

phase have the dimension play, since we are interested in obtaining statistical information of its interaction with other dimensions.

Each iterative step starts by obtaining specialisations of the K-cubes being processed. From the initial K-cube with only the dimension play, immediate specialisations (with dimensionality one degree higher) are derived, using the gen-supersets ($\bigtriangledown$) operation (line 2). These are evaluated, and the most promising one, i.e., the one with lowest gini, is kept, by applying a combination of reduce ($\mathit{\Delta}$) and select-kcube-on-ann (If) operations. The details of the computation of the gini index for the current K-cube-list are abstracted away by a K-algebra equation $interesting(K)$, which is evaluated in lines 3 and 6. Candidate specialisations are obtained for each K-cube until it has reached the desired interestingness, at which point it is saved and removed from further consideration. For example, in the first iteration, L_2 contains the specialisations [(outlook, play), (humidity, play), (windy, play)]. Each specialisation is evaluated in terms of how discriminative they are with regard to the values of play. The one with lowest gini, in this case, the one with outlook, is kept and expanded further until the termination criteria is reached. Notice that, as a K-cube is an aggregated representation of a partition of data, each of its immediate specialisations correspond to sub-partitions of the original one, corresponding to what a decision tree builder must do in practise.

Once the best candidate is retained, the next step is to explore further the partitions of data defined by each cell of a K-cube. For example, from the K-cube with outlook, distinct K-cubes for the subset of cells corresponding to outlook:rainy, outlook:overcast and outlook:sunny are obtained, using the partition-by-dimensions (π^d) in line 5. These have their interestingness evaluated in line 6. Subsequently, the functionality of the iterative operation guarantees that the pure ones are accumulated and the remaining ones are fed back for further refinement. The iteration ends when the K-cube-list $L_{current}$ becomes empty (or the maximum number of passes is reached).

outlook	humidity	windy	play
sunny	normal	false	yes
sunny	normal	true	yes
sunny	high	false	no
sunny	high	true	no
overcast	normal	false	yes
overcast	normal	true	yes
overcast	high	false	yes
overcast	high	true	yes
rainy	normal	false	yes
rainy	normal	true	no
rainy	high	false	yes
rainy	high	true	no
gini: 0.0000			

Fig. 8. K_{m4}: The Final Model

Phase 3 - Model Combination and Deployment The K-cube-list resulting from phase 2 collects distinct k-cubes, which were judged to be interesting in the iterative process. They provide aggregate perspectives on (non-overlapping) partitions of the available data to help determine the class values assigned to each of the latter. The purpose of this final phase is twofold: (1) to assemble these collections of K-cubes back into a single K-cube representing the computed classification model, and (2) to deploy the model in the extensional layer by means of the appropriate bridge operation, which labels the dataset with the corresponding class values given by the cells of the K-cube model. The algebraic specification is given in Algorithm 4.

The first step is to merge the collection in L_{iter} into a single k-cube using union-join, which takes as input two K-cubes and merges them into a single schema-compatible (.e., with the same set of dimensions, measures and annotations) one. Since the operation is binary and on a list, the higher-order reduce (Δ) operations is used, and the result is assigned to L_7. L_7 is then converted into the single K-cube representing the classification model and shown in Fig. 8. Subsequently, the steps in line 3 and 4 (which correspond to an induction leap) output the final model K_{m4} (which can be seen to be equivalent to the tree in Fig. 2). The dimensional values of play are replicated as measures in line 3 using the copy-dim-as-mea ($\overset{d \to m}{\kappa}$) operation. Then, determining which class value is representative per partition is handled by the generalise ($\triangleright$) operation in line 4, which removes the dimension play. As the K-cube shrinks in terms of its dimensionality, each partition of cells (determined by the remaining dimensions) is generalised (i.e., a representative cell is chosen) according to the maximum value of rf, as shown by the input of the operation (Max(rf)). In the deployment stage, firstly the model is converted into a relation using the kcube-to-relation ($\square$) bridge operation. Then, in the extensional layer, the model can be applied to any unlabelled dataset (e.g., R_{test}) using the relational natural-join to yield the predictions in $R_{predicted}$ ($weatherR$ is a relation representing the weather dataset introduced earlier).

Algorithm: **K-Algebraic Expressions for Combination and Deployment**

// combine K-cubes into a model and obtain class

1 $L_7 \leftarrow \Delta^{k_1,k_2}_{\substack{\cup \\ k_1 \bowtie k_2}} (L_{iter})$

2 $K_{m1} \leftarrow \boxdot(L_7)$

3 $K_{m2} \leftarrow \overset{d \to m}{\kappa}_{\langle play_m :\, play \rangle}(K_{m1})$

4 $K_{m3} \leftarrow \triangleright^{\langle Max(rf) \rangle}_{play}(K_{m2})$

5 $K_{m4} \leftarrow \nu^{m}_{\langle play :\, play_m \rangle}(K_{m3})$

// deploy model

6 $R_{model} \leftarrow \square(K_{m4})$

7 $R_{test} \leftarrow \pi_{* \setminus \{play\}}(weatherR)$

8 $R_{predicted} \leftarrow R_{model} \bowtie R_{model}$

Algorithm 4: The Combination and Deployment Phase

4 Related Work

There have been proposals for KDD algebras, e.g., the 3W algebra [12] and the proposal in [6]. Both are based on constraint database systems. They manipulate data and models together using a single carrier, but data mining is seen as a black box, with a single operation as a wrapper for a specific algorithm. Some of the work [13, 10] on integrating knowledge discovery and database research has focused, on a narrow range of tasks, typically association rules, or else confines itself to the proposal of a surface language, which fails to throw light on what computations can be defined with it, either in abstract or concretely. Some work [9, 2] covers a wider range of tasks, but proceeds in a tool set mode rather than being based on general principles, and, as such fail to provide formal framework for the integration of the two research areas. Finally, researchers [17, 15] have been attracted to the idea of using user-defined functions for implementing data mining algorithms that are strongly coupled to database query engines, but again only algorithms have been attempted and there is a distinct lack of higher-level abstractions.

5 Conclusions

This paper has presented two main contributions: (1) the K-algebra and (2) a specification of decision-tree induction in the K-algebra. The latter is, to the best of our knowledge, the first logical-algebraic specification of a KDD task that requires more expressiveness than available in classical database algebras. This shows that the K-algebra opens the way for the expression of challenging KDD tasks in purely algebraic form. In particular, the K-algebra can express the iterative aspects of the KDD process and, through its layered design, covers a wider range of tasks in the KDD process than previous work. More importantly, the K-algebra is a database algebra insofar as it is designed to operate on collections (and hence, flows). It should be noted that the K-Algebra has been designed to model discovery tasks in the symbolic tradition and, hence, the question as to whether it can be used to model tasks that are expressive in, say, connectionist of genetically-inspired approaches (e.g., neural networks, genetic algorithms) has not yet been investigated. The goal of this proposal is to address the issues identified, viz., lack of compositionality, lack of generality and dependency on user expertise. The K-algebra response to lack of compositionality is to exhibit closure through its layered design. Its response to lack of generality is to be founded on generic algorithmic elements (viz., expand, compress, evaluate, accumulate, filter out and combine). Its response to dependency on user expertise is to be formulated and formalised at a logical, abstract level upon which optimisation algorithms can, in future work, be developed. There also remains the important task of trying to capture more kinds of discovery approaches[2] and investigate empirically the practicality of the K-Algebra, especially the development of tools that may be necessary for its usability aspect.

References

1. L. Breiman, J. H. Friedman, R. Olshen, , and C. J. Stone. *Classification and Regression Trees.* CRC Press, 1984.

[2] No claims are made regarding the expressiveness that results from the set of operations proposed other than that it is, as pointed out, sufficient to express ID3 which, in an algebraic setting, is a novel contribution.

2. M. Corporation. Ole db for data mining specification, v1.0, July 2000.
3. U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From Data Mining to Knowledge Discovery: An Overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI/MIT Press, 1996.
4. L. Fegaras and D. Maier. Optimizing Object Queries Using an Effective Calculus. *ACM Transactions on Database Systems*, 25(4), December 2000.
5. H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database System Implementation*. Prentice-Hall, 2000.
6. I. Geist and K. Sattler. Towards Data Mining Operators in Database Systems: Algebra and Implementation. In *Proc. of 2nd Int. Workshop on Databases, Documents, and Information Fusion*, 2002.
7. L. Gerber and A. A. A. Fernandes. The K-Algebra for Knowledge Discovery in Databases. Technical report, University of Manchester, Department of Computer Science, March 2004.
8. R. H. Güting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, 4(2):243–286, 1995.
9. J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A Data Mining Query Language for Relational Databases. SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery, 1996.
10. T. Imielinski and A. Virmani. MSQL: A Query Language for Database Mining. *Data Mining and Knowledge Discovery*, 3(4):373–408, 1999.
11. H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and K. Thompson. TAX: A Tree Algebra for XML. In *Database Programming Languages, 8th International Workshop*, volume 2397 of *LNCS*, pages 149–164, 2002.
12. T. Johnson, L. V. S. Lakshmanan, and R. T. Ng. The 3W Model and Algebra for Unified Data Mining. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 21–32, 2000.
13. R. Meo, G. Psaila, and S. Ceri. An Extension to SQL for Mining Assocation Rules. *Knowledge Discovery and Association Rules*, 2(2):195–224, 1998.
14. J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.
15. S. Sarawagi, S. Thomas, and R. Agrawal. Integrating Mining with Relational Database Systems: Alternatives and Implications. Technical Report RJ 10107 (91923), IBM Almaden Research Center, San Jose, California, March 1998.
16. P. A. Tucker, D. Maier, T. Sheard, and L. Fegaras. Exploiting Punctuation Semantics in Continuous Data Streams. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):555–568, May/June 2003.
17. H. Wang and C. Zaniolo. ATLaS: A Native Extension of SQL for Data Mining. In *Proceedings of the Third SIAM International Conference on Data Mining*, 2003.