fcaR, Formal Concept Analysis with R

by Pablo Cordero, Manuel Enciso, Domingo López-Rodríguez, and Ángel Mora

Abstract Formal concept analysis (FCA) is a solid mathematical framework to manage information based on logic and lattice theory. It defines two explicit representations of the knowledge present in a dataset as concepts and implications. This paper describes an R package called **fcaR** that implements FCA's core notions and techniques. Additionally, it implements the extension of FCA to fuzzy datasets and a simplification logic to develop automated reasoning tools. This package is the first to implement FCA techniques in R. Therefore, emphasis has been put on defining classes and methods that could be reusable and extensible by the community. Furthermore, the package incorporates an interface with the **arules** package, probably the most used package regarding association rules, closely related to FCA. Finally, we show an application of the use of the package to design a recommender system based on logic for diagnosis in neurological pathologies.

1 Introduction

The main goal of knowledge retrieval and knowledge discovery systems is to extract hidden patterns, trends, behaviours, or rules to solve large-impact real-world problems. Usually, these problems present heterogeneous data sources and are at the core of more general decision processes.

A fundamental principle is that the extracted knowledge should provide some understanding of the analyzed data. As computational systems get in critical and sensitive areas such as medicine, the justice system or financial markets, the knowledge becomes much more relevant to make predictions or recommendations or to detect common interest groups or leaders. However, in many cases, the inability of humans to understand the extracted patterns seems problematic.

To represent and retrieve knowledge from datasets, it has become more important to use formal methods based on logic tools. The formal representation of knowledge and the use of logic tools are more suitable for providing understandable answers. Therefore, it can help avoid the lack of interpretability and explainability of the results.

In particular, formal concept analysis (FCA) (Wille, 1982; Ganter and Wille, 1999) is a well-founded mathematical tool, based on lattice theory and logic, which can retrieve and store the knowledge in the form of concepts (analogous to closed itemsets in transactional databases) and implications (association rules with confidence 1). From this perspective, FCA constitutes a framework that complements and extends the study of exact and approximate association rules.

The origins of FCA were devoted to the study of binary datasets (formal contexts) where variables are called attributes. A relevant extension of FCA uses fuzzy sets (Belohlávek and Vychodil, 2016, 2017) to model real-world problems since datasets may contain imprecise, graded or vague information that is not adequately represented as binary values. The fuzzy extension can also model problems with numerical and categorical attributes since these can be scaled to a truth value describing the degree of fulfilment of the attribute.

Some authors have considered the use of FCA in machine learning. Kuznetsov (2004) relates FCA to some mathematical models of machine learning. Ignatov (2017) summarizes the main topics in machine learning and data mining where FCA has been applied: frequent itemset mining and association rules to make classification and clustering. A closer approach appears in Trabelsi et al. (2017), where a method for supervised classification based on FCA is used. The authors extract rules based on concepts generated previously from data. These rules are used to compute the closure of a set of attributes to obtain a classification rule.

From a dataset, FCA can establish maximal clusters, named concepts, between objects and attributes. Each cluster consists of objects having common properties (attributes), which are only fulfilled for these objects. The hierarchy between the concepts and relationships between the attributes (rules or implications) are computed with the same computational cost in FCA.

Among all the techniques used in other areas to extract knowledge, we emphasize using rules for its theoretical and practical interest. The notion of if-then rules, with different names, appears in several areas (databases, machine learning, data mining, formal concept analysis) as a relationship between attributes, called items, properties or atomic symbols regarding the domain. Nowadays, the number of rules extracted even from medium-sized datasets is enormous in all these areas. Therefore, the intelligent manipulation of these rules to reason with them is a hot topic to be explored.

In this direction, Cordero et al. (2002) introduced a logic, named simplification logic for functional dependencies (SL_{FD}), firmly based on a simplification rule, which allows us to narrow the functional dependency set by removing redundant attributes. Although the semantic of implications or if-then

	P1	P2	P3	P4
O1	0	½	½	½
O2	½	1	0	1
O3	½	1	0	1
O4	0	½	1	½

Table 1: A sample formal context. The attributes are P1 to P4 and the objects are named O1 to O4.

rules in other areas are different, the logic can be used too.

Using directly SL_{FD}, some automated deduction methods directly based on this inference system have been developed for classical systems and fuzzy systems (Mora et al., 2003; Cordero et al., 2012; Mora et al., 2012; Rodríguez Lorenzo et al., 2014, 2015).

In the fuzzy framework, several approaches to the definition of fuzzy implications (functional dependencies, rules) are proposed in the literature, see Jezková et al. (2017) for a survey. Our work considers that the definition of graded implication proposed by Belohlávek and Vychodil (2016, 2017) generalizes all the previous definitions. Furthermore, for this general definition of graded implications, an axiomatic system named FASL (fuzzy attribute simplification logic) was developed by Belohlávek et al. (2016), becoming a helpful reasoning tool. Note that FASL is a generalization of SL_{FD} to the fuzzy framework.

The core of our proposal is to provide a user-friendly computational interface to the principal operators and methods of fuzzy FCA, including the mentioned logic tools. This interface is easy to extend to new functionalities and incorporate new methods, such as minimal generators or the computation of different implication bases quickly. The operators and methods implemented are designed to work in the general fuzzy setting, but they are also applicable in the classical binary case.

Thus, the focus of our proposal, the **fcaR** package, is to provide easy access to formal methods to extract all the implicit knowledge in a dataset in the form of concepts and implications, working natively with fuzzy sets and fuzzy implications. Our goal is to provide a unified computational framework for the theoretically-oriented FCA users to develop, test, and compare new methods and knowledge extraction strategies.

Other objectives of this work include presenting an FCA-based tool for knowledge discovery accessible to other scientific communities, allowing for the development of new packages in other fields using FCA techniques, especially in the construction of recommendation systems.

The work is organized as follows: Section Background on FCA begins with a brief look of FCA and Simplification Logic. Section Related works presents other software libraries that implement FCA or related paradigms' core notions. In Section Package design, an explanation of the data structures, classes and constructor methods is covered. Section Formal concept analysis with fcaR shows how to use the package, describing the implemented FCA methods and the use of the simplification logic. In Section Usage example. Fuzzy diagnostic system, a real application of the package in developing a recommender system is illustrated. Finally, some conclusions and future works are presented in Section Conclusions and future work.

2 Background on FCA

In this section, we present the basic notions in the FCA framework using a running example (see Table 1). Note that the package incorporates the main methods in FCA that appear in this summary. Since the formal study of FCA is not the main scope of this work, we recommend the reference (Ganter and Obiedkov, 2016) for further details of this framework.

A *formal context* is a triple (G, M, I), where G is a set of objects, M is a set of attributes and I is a fuzzy relation between objects and attributes, where $I(x, y) \in [0, 1]$ means the truth value to which object x possesses attribute y, indicating $I(x, y) = 0$ the absence of attribute or property y in object x.

The meaning of each entry in the table is the extent to which an object possesses the attribute in the corresponding column. In the example shown in Table 1, the object named O4 fully possesses attribute P3 and possesses P2 and P4 only to degree 50%.

In the remaining of this paper, we will use the notation $^d/a$ to indicate the presence of attribute a with degree d.

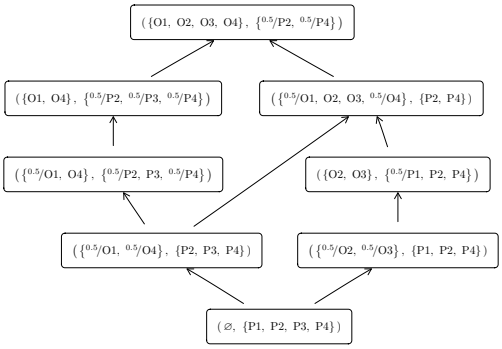

Figure 1: Concept lattice for the context in Table 1. Arrows indicate the direction of the order relationship between concepts.

Derivation operators

Given a fuzzy set of objects $\mathcal{S}$, we can compute its *intent* as the set of attributes that are shared by all objects in $\mathcal{S}$. Analogously, we define the *extent* of a set $\mathcal{T}$ of attributes as the set of objects which have *all* the attributes in $\mathcal{T}$.

In the above example, for $S = \{\text{O1, O2}\}$, we have $\text{intent}(S) = \{^{0.5}/\text{P2, } ^{0.5}/\text{P4}\}$ because $I(\text{O1, P2}) = 0.5$, $I(\text{O1, P4}) = 0.5$, $I(\text{O2, P2}) = 1$ and $I(\text{O2, P4}) = 1$.

The operator ϕ defined by $\phi(T) = \text{intent}(\text{extent}(T))$ is a closure operator and a set of attributes $\mathcal{T}$ is called *closed* if $\mathcal{T} = \phi(\mathcal{T})$. In our example, $\phi(\{^{0.5}/\text{P1}\}) = \{^{0.5}/\text{P1, P2, P4}\}$, meaning that *every object* that has P1 with degree at least 0.5, also has all the attributes {P2, P4}. When $\mathcal{T}$ is closed, the pair $(\text{extent}(\mathcal{T}), \mathcal{T})$ is called a *concept*.

In general, a concept (A, B), where A is a set of objects, and B is a set of attributes, means that the only attributes shared by all objects in A are those in B, and the only objects having all attributes in B are those in A. This property makes (A, B) a *maximal rectangular cluster* in the dataset, with a strong dependence between the objects in A and the attributes in B. In the formal context represented in Table 1, the pair $(\{\text{O2, O3}\}, \{^{0.5}/\text{P1, P2, P4}\})$ is a concept, because $\text{extent}(\{^{0.5}/\text{P1, P2, P4}\}) = \{\text{O2, O3}\}$ and $\text{intent}(\{\text{O2, O3}\}) = \{^{0.5}/\text{P1, P2, P4}\}$, i.e. a fixpoint is achieved.

In FCA, using these derivation operators, two operations can be used to reduce a formal context:

- *Clarification*, which is the removal of duplicated rows (objects) and columns (attributes) of the formal context since duplicates do not contribute knowledge to the context.
- *Reduction*, which removes attributes that can be expressed as the closure of other attributes.

These two operations remove redundancies in the formal context without affecting the knowledge contained in it. Many of the subsequent operations in FCA have high computational complexity, and clarifying and reducing a formal context may reduce the computational time of posterior operations.

The concept lattice

The *concept lattice* of a formal context is the set of all concepts, with the partial order $\leq$ defined as follows: for two concepts (A_1, B_1) and (A_2, B_2), we say that $(A_1, B_1) \leq (A_2, B_2)$ if and only if $A_1 \subseteq A_2 \iff B_2 \subseteq B_1$. For instance, $(\{^{0.5}/\text{O2, } ^{0.5}/\text{O3}\}, \{\text{P1, P2, P4}\}) \leq (\{\text{O2, O3}\}, \{^{0.5}/\text{P1, P2, P4}\})$, since their intents verify $\{^{0.5}/\text{P1, P2, P4}\} \subseteq \{\text{P1, P2, P4}\}$.

This order (or precedence) relationship induces a hierarchy of concepts, which can be graphically represented in the Hasse diagram (Birkhoff, 1940) of the partially ordered set of concepts. In Figure 1, we find the Hasse diagram corresponding to our running example.

From the concept lattice and the order relationship, we can define notions as *subconcept*, *superconcept*, *infimum* and *supremum* of a set of concepts.

Notably, there are the *irreducible elements*, for the infimum (meet) or the supremum (join) operators, which, in the lattice, are shown as those elements with only one arrow departing or arriving at them, respectively. These elements are essential since one can reconstruct the whole lattice by operating with these elements.

The *standard context* for a given formal context $\mathbb{K}$ is another formal context $(\mathcal{J}, \mathcal{M}, \leq)$, where $\mathcal{J}$ and $\mathcal{M}$ are the sets of join- and meet-irreducible elements of $\mathbb{K}$ and $\leq$ is the partial order defined in the concept lattice. This standard context has a concept lattice isomorphic to that of $\mathbb{K}$.

Implications and logic

The knowledge stored in a formal context can also be represented as a set of implications, which are expressions of the form $A \Rightarrow B$ where A and B are sets of attributes or items, indicating that, for every object in which the set of attributes A is present, also B is present. This interpretation is similar to the one defined in data mining/machine learning over the so-named association rules. The confidence (a well-known estimator of the rules' quality) has value 1 in all the implications.

For instance $\{^{0.5}/\text{P1}\} \Rightarrow \{\text{P4}\}$ is a valid implication in the previous example, having the following interpretation: when the attribute P1 has degree at least 0.5 then we have P4 with degree 1.

The *Duquenne-Guigues basis* of implications (Guigues and Duquenne, 1986) is a set of valid implications from which all other valid implications can be deduced. The Duquenne-Guigues basis in our example is given by:

$$\begin{array}{lrcl} 1: & \varnothing & \Rightarrow & \{^{0.5}/\text{P2}, {}^{0.5}/\text{P4}\} \\ 2: & \{^{0.5}/\text{P2}, \text{P4}\} & \Rightarrow & \{\text{P2}\} \\ 3: & \{\text{P2}, {}^{0.5}/\text{P4}\} & \Rightarrow & \{\text{P4}\} \\ 4: & \{\text{P2}, {}^{0.5}/\text{P3}, \text{P4}\} & \Rightarrow & \{\text{P3}\} \\ 5: & \{^{0.5}/\text{P1}, {}^{0.5}/\text{P2}, {}^{0.5}/\text{P4}\} & \Rightarrow & \{\text{P2}, \text{P4}\} \\ 6: & \{^{0.5}/\text{P1}, \text{P2}, \text{P3}, \text{P4}\} & \Rightarrow & \{\text{P1}\} \end{array}$$

In Cordero et al. (2002), the simplification logic, denoted as SL_{FD}, was introduced as a method to manipulate implications (functional dependencies or if-then rules), removing redundancies or computing closures of attributes. This logic is equivalent to Armstrong's Axioms (Armstrong, 1974), which are well known from the 80s in databases, artificial intelligence, formal concept analysis, and others. The axiomatic system of this logic considers reflexivity as the axiom scheme

$$\texttt{[Ref]} \quad \frac{A \supseteq B}{A \Rightarrow B}$$

together with the following inference rules called fragmentation, composition and simplification, respectively, which are equivalent to the classical Armstrong's axioms of augmentation and, more importantly, transitivity.

$$\texttt{[Frag]}\ \frac{A \Rightarrow B \cup C}{A \Rightarrow B} \qquad \texttt{[Comp]}\ \frac{A \Rightarrow B,\ C \Rightarrow D}{A \cup C \Rightarrow B \cup D} \qquad \texttt{[Simp]}\ \frac{A \Rightarrow B,\ C \Rightarrow D}{A \cup (C \setminus B) \Rightarrow (D \setminus B)}$$

The main advantage of SL_{FD} with respect to Armstrong's Axioms is that the inference rules may be considered as equivalence rules, (see the work by Mora et al. (2012) for further details and proofs), that is, given a set of implications Σ, the application of the equivalences transforms it into an equivalent set. In the package presented in this paper, we develop the following equivalences:

1. Fragmentation Equivalency **[FrEq]**: $\{A \Rightarrow B\} \equiv \{A \Rightarrow B \setminus A\}$.
2. Composition Equivalency **[CoEq]**: $\{A \Rightarrow B, A \Rightarrow C\} \equiv \{A \Rightarrow B \cup C\}$.
3. Simplification Equivalency **[SiEq]**: If $A \subseteq C$, then
$$\{A \Rightarrow B, C \Rightarrow D\} \equiv \{A \Rightarrow B, A \cup (C \setminus B) \Rightarrow D \setminus B\}$$
4. Right-Simplification Equivalency **[rSiEq]**: If $A \subseteq D$, then
$$\{A \Rightarrow B, C \Rightarrow B \cup D\} \equiv \{A \Rightarrow B, C \Rightarrow D\}$$

Usually, many areas, the implications have always atomic attributes on the right-hand side. We emphasize that this logic can manage *aggregated* implications, i.e. the implications' consequents do not have to be singletons. This represents an increase of the logic efficiency.

This logic removes attribute redundancies in some of the implications in the Duquenne-Guigues basis presented before. Particularly, the implications with numbers 2, 3, 4, 5 and 6 are simplified to:

$$\begin{array}{lrcl} 2: & \{\text{P4}\} & \Rightarrow & \{\text{P2}\} \\ 3: & \{\text{P2}\} & \Rightarrow & \{\text{P4}\} \\ 4: & \{^{0.5}/\text{P3}, \text{P4}\} & \Rightarrow & \{\text{P3}\} \\ 5: & \{^{0.5}/\text{P1}\} & \Rightarrow & \{\text{P4}\} \\ 6: & \{^{0.5}/\text{P1}, \text{P3}\} & \Rightarrow & \{\text{P1}\} \end{array}$$

One of the primary uses of a set of implications is computing the closure of a set of attributes, the maximal fuzzy set that we can arrive at from these attributes using the given implications.

The importance of computing the closure from implications is because implications can be managed using logic tools. They formally describe the knowledge existing in the dataset; thus, the user can forget about the original *dataset*. In this sense, it is similar to supervised Machine Learning techniques.

The algorithm to compute the closure (Mora et al., 2012) is based on the classical CLOSURE algorithm (Maier, 1983; Ganter and Obiedkov, 2016). After each pass over the set of implications, instead of simply removing the implications used in the step, the simplification rule substitutes the implications by others equivalent but simpler, with fewer attributes. In the binary case, the reduced set of implications does not reference any attribute in the computed closure. A detailed description of the procedure is presented in Algorithm 1.

Algorithm 1: SL_{FD} closure

Input : X: attribute set; Γ: set of implications
Output: X^+: the closure of X with respect to Γ; Γ': the simplified set of implications

$\Gamma' := \Gamma \cup \{\varnothing \Rightarrow X\}$
$X_{\text{new}} := X$
$X_{\text{old}} := X$
repeat
  Replace $\{\varnothing \Rightarrow X_{\text{old}}\}$ with $\{\varnothing \Rightarrow X_{\text{new}}\}$ in Γ'
  $X_{\text{old}} = X_{\text{new}}$
  for *each* $A \Rightarrow B \in \Gamma' \setminus \{\varnothing \Rightarrow X_{\text{new}}\}$ **do**
    if $A \subseteq X_{\text{new}}$ **then**
      Replace $\{\varnothing \Rightarrow X_{\text{new}}\}$ with $\{\varnothing \Rightarrow X_{\text{new}} \cup B\}$
      $X_{\text{new}} := X_{\text{new}} \cup B$
    end
    if $B \subseteq X_{\text{new}}$ **then**
      Remove $A \Rightarrow B$ from Γ'
    end
    if $A \cap X_{\text{new}} \neq \varnothing$ *or* $B \cap X_{\text{new}} \neq \varnothing$ **then**
      Replace $A \Rightarrow B$ with $A \setminus X_{\text{new}} \Rightarrow B \setminus X_{\text{new}}$
    end
  end
until $X_{\text{old}} = X_{\text{new}}$
return X^+ *and* Γ'

For instance, the closure of $S = \{^{0.5}/\text{P2}\}$ with this algorithm is $S^+ = \{^{0.5}/\text{P2}, {}^{0.5}/\text{P4}\}$ (this means that all objects that have all the attributes in S, also have those in S^+). The simplification logic leads us to a reduced set of implications Γ':

1:	{P4}	⇒	{P2}
2:	{P2}	⇒	{P4}
3:	{P2, $^{0.5}$/P3, P4}	⇒	{P3}
4:	{$^{0.5}$/P1}	⇒	{P2, P4}
5:	{$^{0.5}$/P1, P2, P3, P4}	⇒	{P1}

One can interpret these implications as the knowledge in the original implications if the attributes in S are not considered (formally, this is equivalent to suppose that $\varnothing \Rightarrow S$ is true). In the example, if, in addition to having $\{^{0.5}/\text{P2}\}$, we have {P1} with degree 0.5, we can infer that {P2} and {P4} are fully present, by implication number 4.

3 Related works

The package presented in this paper has been developed in the R language. In recent years, R has lived a *remarkable revolution* caused in part by an increasing user community from many different scientific fields. This has led to the development of reproducible research tools, e.g., **rmarkdown** (Xie et al., 2018), and tools to connect and interact with other programming languages, with packages like **Rcpp** (Eddelbuettel and François, 2011) or **reticulate** (Ushey et al., 2020). Besides, the development of a game-changer programming paradigm with *tidy* principles has provided a significant impact on R's usability, see the **tidyverse** of Wickham et al. (2019).

These facts have transformed R from a purely statistical language into one of the most popular languages in data science, machine learning, data mining or visualization. In R, there are multiple packages to perform data mining and machine learning tasks. However, only a few of them are

	ToscanaJ	ConExp	conexp-clj	Galicia	GALACTIC	arules	RKEEL	frbs	fcaR
Programming language	Java	Java	Clojure	Java	Python	R	R	R	R
Context operations		×	×	×	×				×
Context visualization		×	×	×	×				×
Lattice computation		×	×	×	×				×
Lattice operations	×	×	×	×	×				×
Lattice visualization	×	×	×	×	×				×
Implication basis computation		×	×		×				×
Association rules computation		×	×	×	×	×	×	(1)	(2)
Closure operators			×		×				×
Use of logic tools									×
Native fuzzy sets			(3)		(3)				×

Table 2: Functionality comparison among different software libraries for FCA. (1) The rules can be used only for classification or regression; (2) **fcaR** is integrated with **arules** to import and export exact association rules; (3) some packages do not use natively fuzzy sets as representation, but use scaled or discretized contexts.

focused on formal methods that can model and extract knowledge in the form of implications and rules on fuzzy sets and operate on them using logic tools.

The **arules** package by Hahsler et al. (2005) provides the infrastructure for representing, manipulating and analyzing transaction data and patterns (frequent itemsets and association rules) in *binary* settings. The **frbs** package (Riza et al., 2015) presents an implementation of various learning algorithms based on fuzzy rule-based systems (FRBSs) for dealing with classification and regression tasks. The **RKEEL** package (Moyano and Sanchez Ramos, 2016) is an R interface to KEEL, a popular Java software for a large number of different knowledge data discovery tasks. It can extract association rules from numerical datasets, but variables are discretized or categorized first.

It must be noted that none of the previous approaches uses logic tools to infer knowledge from the extracted rules and implications. Also, the support for fuzzy sets is minimal, and none of them implements the core notions of FCA.

Since the **arules** package has become a *de facto* standard, one of the critical points in our proposal's design has been to get **fcaR** easily integrated with **arules**.

In other programming languages, several libraries are implementing FCA methods. Some of the most used libraries are focused on computing the concept lattice and provide a graphical interface to operate with the formal context and the concept lattice: ToscanaJ (Becker and Correia, 2005) and Galicia (Valtchev et al., 2003). Other tools, such as Concept Explorer (ConExp) (Yevtushenko, 2000), with its new versions ConExp NG and ConExp FX, besides their graphical capabilities, implement the core FCA algorithms and operations: context editing, building concept lattices, finding bases of implications and association rules, for instance.

The two most recent and fully-featured libraries are conexp-clj (Hanika and Hirth, 2019) and GALACTIC (Demko and Bertet, 2020), focusing on their extensibility and addition of multiple new algorithms, including the implementation of closure operators and methods based on implications.

In Table 2, we present a comparison of the main features present in each of these libraries.

In conclusion, **fcaR** can be considered among the general-purpose FCA tools with a more comprehensive feature list, integrating the core notions of FCA and logic tools.

4 Package design

In this section, we present the design principles and implementation information about the **fcaR** package: the use of object-oriented programming, the need to be integrated with the **arules** package, and its use in reproducible research.

Package availability

This package is available in CRAN and can be installed using `install.packages('fcaR')`. Also, the development version with new features and bugfixes can be installed from GitHub using `remotes::install_github('Malaga-FCA-group/fcaR')`.

All the documentation in the form of a **pkgdown** site can be found in https://malaga-fca-group.github.io/fcaR/.

Class name	Use
`"Set"`	A basic class to store a fuzzy set using sparse matrices
`"Concept"`	A pair of sets (extent, intent) forming a concept for a given formal context
`"ConceptLattice"`	A set of concepts with their hierarchical relationship. It provides methods to compute notable elements, sublattices and plot the lattice graph
`"ImplicationSet"`	A set of implications, with functions to apply logic and compute closure of attribute sets
`"FormalContext"`	It stores a formal context, given by a table, and provides functions to use derivation operators, simplify the context, compute the concept lattice and the Duquenne-Guigues basis of implications

Table 3: Main classes found in **fcaR**.

Data structures

Now, we present an overview of the data structures implemented in **fcaR** and the design principles on which the development of the package has been based. The main points are the **R6** (Chang, 2020) object-oriented programming (OOP) paradigm and the extensive use of sparse matrices for internal storage of objects.

The **R6** OOP paradigm is being increasingly used in the R ecosystem since its ability to encapsulate data structures and methods related to each class in a single and straightforward interface exposed to the user. This reason and its extensibility have made it the choice to implement the data structures in this package.

The core of the **fcaR** package provides data structures that allow the user to work seamlessly with formal contexts and sets of concepts and implications. The main classes are presented in Table 3. Let us briefly describe the internal representation of the main object classes.

Formal contexts

A formal context (G, M, I) can be represented by a two-way table I indicating the relationship between objects and attributes. This table is expressed in matrix form in R, with rows corresponding to objects and columns corresponding to attributes.

In the classical setting, as occurred in the **arules** package, a formal context may represent a collection of *itemsets* which conform to a *transactions database*. In such a setting, the matrix is binary, and each of its entries represents the presence (1) or absence (0) of an item in a particular itemset. A value of 1 in the matrix is interpreted as the object fully possessing the associated attribute.

In this package, one of the main features is dealing with fuzzy or graded attributes. Therefore I is not a binary matrix anymore, but a numerical matrix with entries in the interval $[0, 1]$. We emphasize that our package can deal with binary datasets as a particular case of the fuzzy case. Most of the methods implemented work for binary and fuzzy Formal Concept Analysis.

The **R6** class `"FormalContext"` stores the relationship matrix (or *incidence* matrix, if binary) I in a compressed sparse format, given by class `"dgCMatrix"` from package **Matrix** (Bates and Maechler, 2021). The reason to store I as a sparse matrix is that, in practice, most situations will occur with many objects, and only a few attributes present for each object. This allows for greater computational efficiency.

The main methods applicable to an object of the `"FormalContext"` class are presented in Table 4.

The features of the **R6** paradigm allow us to build another critical aspect of a `"FormalContext"`: it stores both the associated `"ConceptLattice"` and the Duquenne-Guigues basis as an `"ImplicationSet"`, which can be accessed using `fc$concepts` and `fc$implications`, respectively. Initially, these instances are empty (no concepts nor implications are included), and they are filled with the corresponding data as needed.

Concept lattices

A `"ConceptLattice"` object is usually built from a `"FormalContext"` or by using some specific methods on an existing `"ConceptLattice"`. Thus, although it has a constructor method, the user will not likely use it to create an instance of this class.

Internally, the `"ConceptLattice"` class stores three sparse matrices: two for the extents and intents in matrix form (rows are objects/attributes and columns represent concepts) and a sparse matrix indicating the order relationship between concepts. The element (i, j) of this matrix is 1 if and only if

Method name	Purpose
`new(I)`	Constructor for the `"FormalContext"` class. `I` can be a `"matrix"`, a `"data.frame"` or a filename from where to import the context
`intent(S)`	Computes the intent of a set `S` of objects
`extent(S)`	Computes the extent of a set `S` of attributes
`closure(S)`	Computes the closure of a set `S` of attributes
`clarify()`	Performs clarification of a `"FormalContext"`
`reduce()`	Performs reduction of a `"FormalContext"`
`standardize()`	Builds the standard context $(\mathcal{J}, \mathcal{M}, \leq)$ for the given `"FormalContext"`
`find_concepts()`	Builds the concept lattice following the NextClosure algorithm for fuzzy formal contexts
`find_implications()`	Uses the modified NextClosure for fuzzy formal contexts to compute the concept lattice and the Duquenne-Guigues basis of implications
`to_transactions()`	Converts the formal context to an object of class `"transactions"` from the **arules** package

Table 4: Main methods of the `"FormalContext"` class.

the i-th concept is a subconcept of the j-th concept, otherwise it is 0.

The main methods of objects of the `"ConceptLattice"` class are summarized in Table 5.

Method name	Purpose
`new(A,B)`	Constructor for the `"ConceptLattice"` class. `A` and `B` are the extents and intents of the concepts
`intents()`	Retrieves the intents of all the concepts
`extents()`	Retrieves the extents of all the concepts
`sublattice(C)`	Computes the smallest sublattice that includes all concepts in the concept set `C`
`meet_irreducibles()`, `join_irreducibles()`	Compute the meet-irreducible and join-irreducible elements of the lattice
`decompose(C)`	Decomposes the concept `C` as the supremum of meet-irreducible concepts
`supremum(C)`, `infimum(C)`	Compute the supremum or infimum of a set `C` of concepts
`subconcepts(C)`, `superconcepts(C)`	Compute the subconcepts and superconcepts of a concept `C`
`lower_neighbours(C)`, `upper_neighbours(C)`	For a given concept `C`, compute its lower and upper neighbours in the lattice

Table 5: Main methods of the `"ConceptLattice"` class.

Implication sets

There are two ways in which an `"ImplicationSet"` can be created: by finding the implications of a given `"FormalContext"` or by importing a `"rules"` object from the **arules** package.

In both cases, internally, an `"ImplicationSet"` object is composed of two sparse matrices, `lhs_matrix` and `rhs_matrix`, representing the left-hand and right-hand sides (LHS and RHS, respectively) of the computed implications. As it is the default in **Matrix**, these matrices are stored column-wise, such that column j represents the j-th implication in the set, and row i stands for the i-th attribute.

The main methods applicable to an object of class `"ImplicationSet"` are described in Table 6.

Method name	Purpose
`new(A,B)`	Constructor of the `"ImplicationSet"` class. `A` and `B` represent the left-hand and right-hand sides of the implications
`add(P)`	Concatenates the implications in the current set with those in `P`
`closure(S)`	Computes the closure of the attribute set `S` with respect to the current implication set, applying the SL_{FD} logic
`recommend(S,attr)`	Computes the closure of `S` and filters it to show only the attributes in `attr`
`apply_rules(rules)`	Transforms the `"ImplicationSet"` into another using equivalence rules
`to_basis()`	Transforms the `"ImplicationSet"` to an equivalent basis of implications
`filter(lhs,rhs)`	Filters the `"ImplicationSet"` to retrieve only implications with specific attributes in the left-hand or in the right-hand sides
`to_arules()`	Converts a binary `"ImplicationSet"` to the `"rules"` class of **arules**

Table 6: Main methods of the `"ImplicationSet"` class.

Interfacing with arules

One of the main design objectives in this package is interoperability with the **arules** package since it can be considered a standard in the field.

The constructor methods for the `"FormalContext"` and `"ImplicationSet"` classes allow as inputs objects of types `"transactions` and `"rules"`, respectively, from the **arules** package.

A `"transactions"` object in **arules** can be represented by a binary formal context. Each transaction can be translated into one object as defined in FCA. In contrast, different transaction items are mapped to binary variables according to whether the item is present or not in the transaction. Any object of type `"transactions"` can be used to initialize a `"FormalContext"`, assigning the labels of the items present in the *transaction database* to the attributes' names in the context, and using the `"itemMatrix"` as the internal representation of the binary incidence matrix. Also, **fcaR** can export the `"FormalContext"` to the `"transactions"` format, by using the `to_transactions()` method of a binary `"FormalContext"`.

Analogously, a `"rules"` object can be used to create a `"ImplicationSet"`. The LHS and RHS of the `"rules"` object are mapped to the `lhs_matrix` and `rhs_matrix` of the `"ImplicationSet"`. In addition, the method `to_arules()` can be used to convert any binary `"ImplicationSet"` to the `"rules"` format.

These interfaces allow the users who usually work in other areas with **arules** to use FCA methods and algorithms for transactions databases, as an extension of what can do in **arules**, and then, if necessary, to convert back to the **arules** format.

Use for reproducible research

Another of the design goals of the **fcaR** package is that it could be used in research and used in publications. All the implemented classes in **fcaR** provide a `to_latex()` function so that concepts, implications and the table of the formal context can be easily exported to LaTeX. Also, the `"FormalContext"` and `"ConceptLattice"` classes have a `plot()` method that allows to graphically represent that object types. This `plot()` function allows to export the graphs in LaTeX format to be included directly in PDF reports with publication quality (see the Figures in this work).

All this document has been written using the Rmarkdown format using these functions to generate the tables and listings of concepts and implications. The only required LaTeX packages are `amssymb` (for some mathematical symbols), `longtable` and `caption` (for the listings of concepts and implications) and `tikz` (for plotting formal contexts and concept lattices).

5 Formal concept analysis with fcaR

In this Section, we will show how to perform the basic operations mentioned in Section Background on FCA. We will use the same formal context shown in Table 1.

Let us suppose that the variable 'I' stores the matrix corresponding to that formal context. To create a new `"FormalContext"` with that matrix, the command is `fc <- FormalContext$new(I)`. Now, the object `fc` stores the formal context and enables us to perform the corresponding operations on it.

For example, we can access the attribute and object names with `fc$attributes` and `fc$objects`, respectively. We can even plot a heatmap (see Fig. 2) to show the sparsity of the context, very useful for large contexts.

Derivation operators

The methods that implement the derivation operators are named after them: `intent()`, `extent()` and `closure()`. They can be applied on objects of type `"Set"`, representing fuzzy sets of objects or attributes:

```
> S <- Set$new(fc$objects, O1 = 1, O2 = 1)
> S
{O1, O2}
> fc$intent(S)
{P2 [0.5], P4 [0.5]}
> T <- Set$new(fc$attributes, P1 = 1, P3 = 1)
> T
{P1, P3}
> fc$extent(T)
{}
```

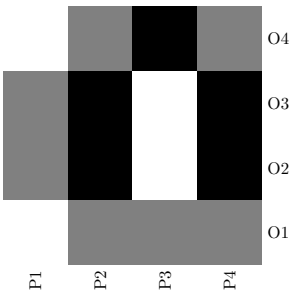

Figure 2: The heatmap representing the formal context shown in Table 1, obtained by doing `fc$plot()`. A grey scale is used, where white indicates the value 0 for an object-attribute pair, while black indicates the value 1, with the greys representing the intermediate values. This type of graph aids visual inspection of the context, being able to check its sparsity, density or possible patterns.

```
> fc$closure(T)
{P1, P2, P3, P4}
```

In addition, we can perform *clarification* on the formal context, by using `fc$clarify()`, giving:

```
FormalContext with 3 objects and 3 attributes.
           P1   P3  [P2, P4]
       O1  0   0.5     0.5
       O4  0    1      0.5
 [O2, O3] 0.5   0       1
```

The duplicated rows and columns in the formal context have been collapsed, and the corresponding attributes and objects' names are grouped together between brackets, e.g., [P2, P4].

Concept lattice

The command to compute the concept lattice for a `"FormalContext"` `fc` is `fc$find_concepts()`. The lattice is stored in `fc$concepts`, which is of the `"ConceptLattice"` class.

```
> fc$concepts
A set of 8 concepts:
1: ({O1, O2, O3, O4}, {P2 [0.5], P4 [0.5]})
2: ({O1, O4}, {P2 [0.5], P3 [0.5], P4 [0.5]})
3: ({O1 [0.5], O4}, {P2 [0.5], P3, P4 [0.5]})
4: ({O1 [0.5], O2, O3, O4 [0.5]}, {P2, P4})
5: ({O1 [0.5], O4 [0.5]}, {P2, P3, P4})
6: ({O2, O3}, {P1 [0.5], P2, P4})
7: ({O2 [0.5], O3 [0.5]}, {P1, P2, P4})
8: ({}, {P1, P2, P3, P4})
```

In order to know the *cardinality* of the set of concepts (that is, the number of concepts), we can use `fc$concepts$size()`, which gives 8 in this case. The complete list of concepts can be printed with `fc$concepts$print()`, or simply `fc$concepts`. Also, they can be translated to LaTeX using the `to_latex()` method, as mentioned before.

The typical subsetting operation in R with brackets is implemented to select specific concepts from the lattice, giving their indexes or a boolean vector indicating which concepts to keep. The same rules for subsetting as in R base apply:

```
> fc$concepts[c(1:3, 5, 8)]
A set of 5 concepts:
1: ({O1, O2, O3, O4}, {P2 [0.5], P4 [0.5]})
2: ({O1, O4}, {P2 [0.5], P3 [0.5], P4 [0.5]})
3: ({O1 [0.5], O4}, {P2 [0.5], P3, P4 [0.5]})
4: ({O1 [0.5], O4 [0.5]}, {P2, P3, P4})
5: ({}, {P1, P2, P3, P4})
```

In addition, the user can compute concepts' *support* (the proportion of objects whose set of attributes contains the intent of a given concept) by means of `fc$concepts$support()`.

```
> fc$concepts$support()
[1] 1.00 0.50 0.25 0.50 0.00 0.50 0.00 0.00
```

Sublattices

When the concept lattice is too large, it can be useful to work with a sublattice of the complete lattice on certain occasions. To this end, we use the `sublattice()` function.

For instance, to build the sublattice generated by the concepts $\left(\left\{^{0.5}/\text{O1}, ^{0.5}/\text{O4}\right\}, \{\text{P2, P3, P4}\}\right)$ and $\left(\{\text{O2, O3}\}, \left\{^{0.5}/\text{P1, P2, P4}\right\}\right)$, which had indexes 5 and 6 in the list of concepts, we can do:

```
> fc$concepts$sublattice(5:6)
A set of 4 concepts:
1: ({O1 [0.5], O2, O3, O4 [0.5]}, {P2, P4})
2: ({O1 [0.5], O4 [0.5]}, {P2, P3, P4})
3: ({O2, O3}, {P1 [0.5], P2, P4})
4: ({}, {P1, P2, P3, P4})
```

Some interesting sublattices appear when we consider only concepts fulfilling a given condition (e.g., a minimum support), using a command such as `fc$concepts$sublattice(fc$concepts$support() > 0.5)`.

Subconcepts, superconcepts, infimum and supremum

For a given concept (A, B), we can find all its subconcepts ((C_i, D_i) such that $(C_i, D_i) \leq (A, B)$) by using the `subconcepts()` functions. Analogously, the `superconcepts()` function can be used to compute the set of (C_i, D_i) such that $(A, B) \leq (C_i, D_i)$.

For instance, if we take a sample concept $L = \left(\left\{^{0.5}/\text{O1, O2, O3, }^{0.5}/\text{O4}\right\}, \{\text{P2, P4}\}\right)$, we can compute its subconcepts and superconcepts using:

```
> fc$concepts$subconcepts(L)
A set of 5 concepts:
1: ({O1 [0.5], O2, O3, O4 [0.5]}, {P2, P4})
2: ({O1 [0.5], O4 [0.5]}, {P2, P3, P4})
3: ({O2, O3}, {P1 [0.5], P2, P4})
4: ({O2 [0.5], O3 [0.5]}, {P1, P2, P4})
5: ({}, {P1, P2, P3, P4})
> fc$concepts$superconcepts(L)
A set of 2 concepts:
1: ({O1, O2, O3, O4}, {P2 [0.5], P4 [0.5]})
2: ({O1 [0.5], O2, O3, O4 [0.5]}, {P2, P4})
```

Also, we can define the infimum and the supremum of a set of concepts as the greatest common subconcept and the lowest common superconcept of all the given concepts, respectively. For a list of concepts, we can compute its infimum and supremum using the `supremum()` and `infimum()` methods of the `"ConceptLattice"` object. Additionally, irreducible elements in the lattice, with respect to join (supremum) and meet (infimum), can be computed for a given concept lattice with the methods `join_irreducibles()` and `meet_irreducibles()`.

```
> fc$concepts$meet_irreducibles()
A set of 5 concepts:
1: ({O1, O4}, {P2 [0.5], P3 [0.5], P4 [0.5]})
2: ({O1 [0.5], O4}, {P2 [0.5], P3, P4 [0.5]})
3: ({O1 [0.5], O2, O3, O4 [0.5]}, {P2, P4})
4: ({O2, O3}, {P1 [0.5], P2, P4})
5: ({O2 [0.5], O3 [0.5]}, {P1, P2, P4})
> fc$concepts$join_irreducibles()
A set of 5 concepts:
1: ({O1, O4}, {P2 [0.5], P3 [0.5], P4 [0.5]})
2: ({O1 [0.5], O4}, {P2 [0.5], P3, P4 [0.5]})
3: ({O1 [0.5], O4 [0.5]}, {P2, P3, P4})
4: ({O2, O3}, {P1 [0.5], P2, P4})
5: ({O2 [0.5], O3 [0.5]}, {P1, P2, P4})
```

These irreducible elements are essential since they constitute the basic elements with which the entire concept lattice can be reconstructed.

	M1	M2	M3	M4	M5
J1	×				
J2	×	×			
J3	×	×	×		
J4			×	×	
J5			×	×	×

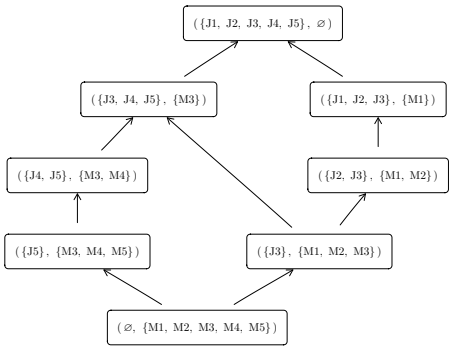

Table 7: Left: Standard context associated to the context in Table 1; M1 to M5 refer to the meet-irreducible elements of the formal context, and J1 to J5, to the join-irreducible ones. Right: concept lattice of the standard context, isomorphic to the one in Figure 1.

The standard context

The standard context has a concept lattice that is isomorphic to the one of the original context, so it encapsulates the same knowledge. With **fcaR** one can directly compute the standard context by using the `standardize()` function, for example, using `fc_std <- fc$standardize()` to save the new context to another variable.

This is a method that applies to a `"FormalContext"` object, but is closely related to its associated `"ConceptLattice"`, since the objects and attributes of the newly created standard context are the meet- and join- irreducible elements of the concept lattice.

The standard context for the example in Table 1 and its corresponding lattice are shown in Table 7.

Implications and logic

As mentioned earlier, a core problem in FCA is the computation of implications that hold in a formal context. The iterative fuzzy version of NEXTCLOSURE has been implemented in C and made accessible from **fcaR** thanks to an interface using **Rcpp**. As we have said, our package can manage classical implications from binary datasets. In order to build the Duquenne-Guigues basis of implications, by using the NEXTCLOSURE algorithm, the command `fc$find_implications()` is executed, and the result is stored in `fc$implications`.

The set of implications is stored as an object of class `"ImplicationSet"`, which can be inspected with the `print()` method or simply with `fc$implications`. In order to get a subset of the implications by providing its indexes, the standard R subsetting with '`[`' will create another `"ImplicationSet"` with exactly the rules required. For instance, `fc$implications[1:3]` gives:

```
Implication set with 3 implications.
Rule 1: {} -> {P2 [0.5], P4 [0.5]}
Rule 2: {P2 [0.5], P4} -> {P2}
Rule 3: {P2, P4 [0.5]} -> {P4}
```

These implications can be converted to LaTeX using the `to_latex()` method, giving:

$$\begin{array}{lrcl} 1: & \varnothing & \Rightarrow & \{^{0.5}/\mathrm{P2},\ ^{0.5}/\mathrm{P4}\} \\ 2: & \{^{0.5}/\mathrm{P2},\ \mathrm{P4}\} & \Rightarrow & \{\mathrm{P2}\} \\ 3: & \{\mathrm{P2},\ ^{0.5}/\mathrm{P4}\} & \Rightarrow & \{\mathrm{P4}\} \end{array}$$

On the other hand, one can `filter()` the implications to retrieve just those with specific attributes in the LHS and RHS. As before, the method returns a new `"ImplicationSet"` object. For example, to get the implications with attributes P1 and P2 in the LHS and attribute P4 in the RHS, the user can execute `fc$implications$filter(lhs = c('P1', 'P2'), rhs = 'P4')`.

Some quantities can be computed from a set of implications:

- *Cardinality*, that is, the number of implications, with `fc$implications$cardinality()`.
- *Size*, which is the cardinality of the LHS and RHS of *each* implication, interpreted as fuzzy sets (thus non-integer sizes can appear): `fc$implications$size()`.
- *Support*, the proportion of objects in the formal context whose set of attributes is a *superset* of the LHS of each implication: `fc$implications$support()`.

Application of the simplification logic

In **fcaR**, the simplification logic has been implemented and made accessible from an `"ImplicationSet"` object through method `apply_rules`.

The list of equivalence rules applicable to an `"ImplicationSet"` is stored in a `registry` object from the **registry** package by Meyer (2019).

This registry is called `equivalencesRegistry`, and one can inspect its contents by using:

```
> equivalencesRegistry$get_entry_names()
[1] "Composition"          "Generalization"       "Reduction"
[4] "Simplification"       "Right Simplification" "Reorder"
```

These names correspond to the methods added to the registry by default and are used to index those methods. Every method is accompanied by a description, so that we can see its definition:

```
> equivalencesRegistry$get_entry('Composition')
     method Composition
        fun <<function>>
description A -> B and A -> C equivalent to A -> BC
```

We can even use abbreviated names to refer to the method. For instance, we can use 'comp' instead of 'Composition' in the above command to obtain the information about the composition rule.

The registry's use enables the user to extend the functionality provided by the **fcaR** package. One can define and implement new equivalences rules and add them to the registry, and make them available to the `apply_rules()` method.

In order to add a new equivalence rule, we use the following:

```
> equivalencesRegistry$set_entry(method = 'Method name',
+                                fun = method_function,
+                                description = 'Method description')
```

where `method_function()` is the R function that computes the equivalences, and has the signature `function(LHS,RHS,attributes)`, where `LHS` and `RHS` are the sparse matrices defining the left-hand and right-hand sides of the implications, `attributes` is the character vector of attribute names, and returns the modified `LHS` and `RHS`. This mechanism has been used in the package to implement additional equivalence rules.

The user can decide which rules to remove redundancies will be applied and in which order:

```
> fc$implications$apply_rules(rules = c('reduction',
+                                       'comp',
+                                       'gener',
+                                       'simpl'))
```

These methods can be applied to binary and fuzzy implications.

Computation of closure and recommendations

One of the primary uses of a set of implications extracted from a formal context is computing the closure of a set of attributes, that is, the fuzzy set obtained by applying the given implications on the set of attributes.

The `closure()` method returns both the closure and the reduced `"ImplicationSet"` if `reduce = TRUE` and only the closure if `reduce = FALSE` (default). For instance, we can create a fuzzy set where the attribute `P2` is present with:

```
> A <- Set$new(attributes = fc$attributes, P2 = 1)
```

Then, we can compute its closure and the reduced set of implications doing:

```
> fc$implications$closure(A, reduce = TRUE)
$closure
{P2, P4}

$implications
Implication set with 2 implications.
Rule 1: {P3 [0.5]} -> {P3}
Rule 2: {P1 [0.5], P3} -> {P1}
```

6 Usage example. Fuzzy diagnostic system

In this section, a complete example of the use of **fcaR** on real-world problems is presented: Designing a diagnostic system from a formal context with (fuzzy) medical data.

The dataset for this section is provided and documented in the package.

In this example, the aim is to build an automated system using the **fcaR** package to perform medical diagnosis. We have focused on neurological pathologies since, in recent years, an increasing number of initiatives have appeared to share, curate, and study specific, prevalent brain pathologies. Among these pathologies, schizophrenia is of the highest interest, and public, curated repositories have been released.

The data source is SchizConnect (Wang et al., 2016), an online data repository integrating and mediating data from other schizophrenia-related databases, such as COBRE (Aine et al., 2017), which collect neuroimaging, psychological, neurological and clinical information. SchizConnect allows retrieving data about the patients that fulfil some conditions introduced as a query to the database. A subset of the COBRE dataset has been retrieved by querying SchizConnect for 105 patients with neurological and clinical symptoms. We also collected their corresponding diagnosis.

Among the clinical attributes in the dataset, one can find:

- *Calgary Depression Scale for Schizophrenia* (Addington et al., 1990), nine items (attributes) assessing the level of depression in schizophrenia, differentiating between positive and negative aspects of the disease.
- The *Simpson-Angus Scale* (Simpson and Angus, 1970), six items to evaluate Parkinsonism-like alterations related to schizophrenia in an individual.
- The *Structured Clinical Interview for DSM-III-R Personality Disorders* (First et al., 1997), with nine variables related to the presence of signs affecting personality.
- The diagnosis for each individual: it can be *schizophrenia strict* (abbreviated '`dx_ss`') or *other diagnosis* (abbreviated '`dx_other`', which includes schizoaffective and bipolar disorders). These diagnoses are mutually exclusive; thus, only one of them is assigned to each patient.

In summary, the dataset consists of the previous 30 attributes related to signs or symptoms and two attributes related to diagnosis. So the dataset has 105 objects (patients), and 32 attributes to explore. The symptom attributes are multi-valued.

For a given attribute (symptom), the available grades are *absent*, *minimal*, *mild*, *moderate*, *moderate severe*, *severe* and *extreme*. Thus, taking into account these scale levels, the attributes are coded as fuzzy and graded attributes with values 0, $1/6$, $1/3$, $1/2$, $2/3$, $5/6$ and 1, respectively. Since the symptom attributes are ordinal, for the sake of simplicity, they have been encoded as grades in the interval $[0, 1]$, just by mapping the lowest (*absent*) value to 0 and the greatest (*extreme*) to 1 and placing the remaining values equidistantly in the interval. In Ganter and Obiedkov (2016), other strategies of scaling for ordinal, nominal, or interval attributes are presented.

In **fcaR**, this dataset is exposed to the user with the name `cobre32`, and we can use it to create a fuzzy formal context (see Figure 3):

```
> fc <- FormalContext$new(cobre32)
```

Now, let us build a diagnosis system that employs the tacit knowledge present in the formal context. The objective is to define a function that takes a `"Set"` as input (with the clinical attributes of an individual) and returns the degree of the diagnosis attributes using the implications extracted from the formal context as an inference engine.

Next, we use the NEXTCLOSURE algorithm to extract implications and compute the set of concepts, using `fc$find_implications()`.

The concept lattice is quite big (14706 concepts); therefore, it cannot be plotted here for space and readability reasons. For this reason, we only plot a sublattice of small size in Figure 4.

There is an aggregate of 985 implications extracted. Let us compute the average cardinality of the LHS and the RHS of the extracted rules:

```
> colMeans(fc$implications$size())
     LHS      RHS
2.417597 1.954146
```

Note that our paradigm can deal with non-unit implications, that is, where there is more than one attribute in the RHS of the implication. This feature is an extension of what is usual in other paradigms, for example, in transactional databases.

We can use the *simplification logic* to remove redundancies and reduce the LHS and RHS size of the implications. The reason to do this is to decrease the computational cost of computing closures:

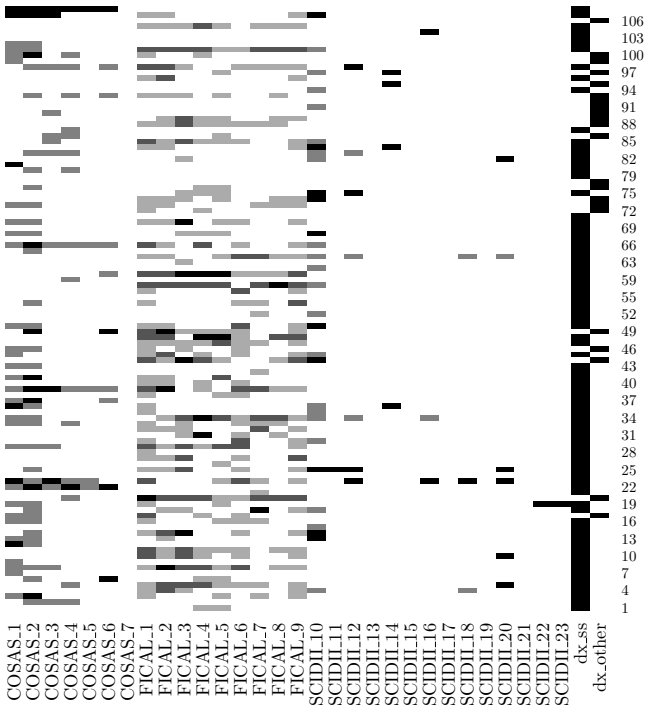

Figure 3: Formal context of the `cobre32` dataset. The default plot type allows us to identify patterns occurring in the attributes, such as the sparsity of the `SCIDII` variables. Additionally, we can see that the variables indicating diagnosis, `dx_ss` and `dx_other` in the last two columns, are binary and mutually exclusive.

```
> fc$implications$apply_rules(rules = c('simplification', 'rsimplification'))
> colMeans(fc$implications$size())
     LHS      RHS
1.998308 1.557191
```

We can see that the average cardinality of the LHS has been reduced from 2.418 to 1.998 and that the one of the RHS, from 1.954 to 1.557.

With the simplified implication set, we can build a recommender system by simply wrapping the `recommend()` method inside a function:

```
> diagnose <- function(S) {
+
+   fc$implications$recommend(S = S,
+                             attribute_filter =
+                               c('dx_ss', 'dx_other'))
+
+ }
```

This function can be applied to `"Set"`s that have the same attributes as those of the formal context. The `attribute_filter` argument specifies which attributes are of interest, in our case, the diagnosis attributes.

Let us generate some sets of attributes and get the recommendation (diagnosis) for each one:

```
> S1 <- Set$new(attributes = fc$attributes,
+               COSAS_1 = 1/2, COSAS_2 = 1, COSAS_3 = 1/2,
+               COSAS_4 = 1/6, COSAS_5 = 1/2, COSAS_6 = 1)
> diagnose(S1)
   dx_ss dx_other
       1        0
> S2 <- Set$new(attributes = fc$attributes,
+               COSAS_2 = 1, COSAS_6 = 1, FICAL_1 = 1/3, FICAL_3 = 1/3)
> diagnose(S2)
   dx_ss dx_other
```

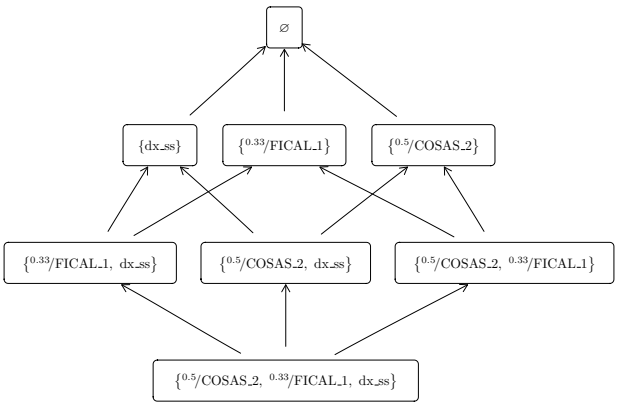

Figure 4: Hasse diagram for a sublattice of the `cobre32` formal context. Since the complete concept lattice is huge, with the Hasse diagram of a sublattice we can understand the relationship in a subset of the attributes. In this case, we can see how variables `COSAS_2` and `FICAL_1` are related to the variable `dx_ss` which indicates the schizophrenia diagnosis.

```
       0        0
> S3 <- Set$new(attributes = fc$attributes,
+               COSAS_4 = 2/3, FICAL_3 = 1/2, FICAL_5 = 1/2, FICAL_8 = 1/2)
> diagnose(S3)
   dx_ss dx_other
       0        1
```

These results mean that, for `S1`, the recommended diagnosis is schizophrenia strict, for `S2` there is not enough information, and the recommended diagnosis for `S3` is *other*, different from schizophrenia strict. For `S2`, maybe adding more attributes to it can help in obtaining a diagnosis.

One can inspect the reduced set of implications obtained after computing the closure for `S2`, simplify them and filter them to get the implications that can be applied if more attribute values are known for `S2`:

```
> cl <- fc$implications$closure(S2, reduce = TRUE)
> cl$implications$apply_rules(c('simp', 'rsimp', 'reorder'))
> cl$implications$filter(rhs = c('dx_ss', 'dx_other'),
+                        not_lhs = c('dx_ss', 'dx_other'), drop = TRUE)
Implication set with 12 implications.
Rule 1: {FICAL_5 [0.33]} -> {dx_other}
Rule 2: {FICAL_6 [0.33], FICAL_8 [0.33]} -> {dx_ss}
Rule 3: {SCIDII_18 [0.33]} -> {dx_ss}
Rule 4: {COSAS_1 [0.5], FICAL_8 [0.33]} -> {dx_ss}
Rule 5: {SCIDII_20 [0.33]} -> {dx_ss}
Rule 6: {SCIDII_16 [0.33]} -> {dx_ss}
Rule 7: {SCIDII_12 [0.33]} -> {dx_ss}
Rule 8: {FICAL_7 [0.33]} -> {dx_ss}
Rule 9: {FICAL_6 [0.33], SCIDII_10 [0.5]} -> {dx_ss}
Rule 10: {COSAS_3 [0.5]} -> {dx_ss}
Rule 11: {COSAS_1 [0.5]} -> {dx_ss}
Rule 12: {SCIDII_10 [0.5]} -> {dx_ss}
```

We can check that, for `S2`, if the presence of any symptom (of the LHS of the implications) is verified, then the implications above could directly tell us the diagnosis.

An extended version of the diagnosis system in this example has been presented in Cordero et al. (2020), where the **fcaR** package has been used to build a conversational recommender system based on fuzzy rules. In that work, the recommender system designed with the help of **fcaR** has obtained better results than those of the classical methods used in the area of recommendations.

7 Conclusions and future work

This work aims to present the first R package implementing the core methods in Formal Concept Analysis. This development aims to provide a helpful tool for the FCA community and other fields where knowledge discovery and retrieval play an essential role. Notably, the hybridization of FCA with other Machine Learning techniques and its application to data mining processes is well-known, making FCA an appealing tool for knowledge discovery.

This package provides **R6** classes and methods to manage datasets presented as formal contexts, use the derivation operators, work with the concepts (closed itemsets) or find and manage the basis of implications. Additionally, the **fcaR** package implements a logic to infer knowledge from sets of implications, allowing to compute closures and therefore, it can be used as the engine to build recommender systems and automated reasoning tools.

Another feature included in the package is providing graphical visualisations of the concept lattice that condenses the extracted knowledge.

The tool has been developed from two principles: *integration* with other packages and *extensibility*. First, the **arules** package is a *de facto* standard when working with transactional databases, closed itemsets and association rules in R. Since FCA's perspective can be seen as complementary to this, **fcaR** is able to import and export both the datasets and the implications from/to the analogous classes defined in **arules**. This way, the user of **arules** can benefit from the FCA tools in **fcaR**, including the simplification logic.

The extensibility of the **fcaR** is guaranteed by its design philosophy. First, the object-oriented programming paradigm allows extending the classes and methods to other datatypes, other kinds of formal contexts, or, for example, association rules. Second, using a registry for equivalence rules makes it easy to include new logic tools in the package's architecture without affecting existing code. Thus, users can extend the package's functionality by incorporating their methods and then testing and comparing them in a single framework.

Thus, **fcaR** implements a wide range of features. With the help of the included documentation and the comprehensive vignettes, any user can start analysing datasets with FCA tools quickly.

Regarding the low-level implementation, we have used sparse matrices as the primary internal data structure of the package since they represent a space- and cost-efficient storage. Also, when needed, the **fcaR** uses parallel computing and C routines to increase efficiency and tackle algorithmic bottlenecks.

As an example of use, we have used the implications extracted from a dataset to develop a recommender system to help diagnose schizoaffective disorders quickly.

The package is currently in stable development status. However, we devise future extensions in terms of new or improved algorithms or as new applications arise (in areas such as recommender systems, unsupervised learning or text mining).

We emphasize that **fcaR** is having a great acceptance. At the time of writing, it has now reached more than 20,000 downloads from CRAN.

Acknowledgments

This work has been partially supported by the projects TIN2017-89023-P (Spanish Ministry of Economy and Competitiveness, including FEDER funds), PGC2018-095869-B-I00 (Spanish Ministry of Science and Innovation), and the Junta de Andalucía project UMA2018-FEDERJA-001, co-funded by the European Regional Development Fund.

Bibliography

D. Addington, J. Addington, and B. Schissel. A depression rating scale for schizophrenics. *Schizophrenia research*, 3(4):247–251, 1990. [p359]

C. J. Aine, H. J. Bockholt, J. R. Bustillo, J. M. Cañive, A. Caprihan, C. Gasparovic, F. M. Hanlon, J. M. Houck, R. E. Jung, J. Lauriello, J. Liu, A. R. Mayer, N. I. Perrone-Bizzozero, S. Posse, J. M. Stephen, J. A. Turner, V. P. Clark, and V. D. Calhoun. Multimodal neuroimaging in schizophrenia: Description and dissemination. *Neuroinformatics*, 15(4):343–364, 2017. URL https://doi.org/10.1007/s12021-017-9338-9. [p359]

W. W. Armstrong. Dependency Structures of Data Base Relationships. In *IFIP Congress*, pages 580–583, 1974. [p349]

D. Bates and M. Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2021. URL https://CRAN.R-project.org/package=Matrix. R package version 1.3-3. [p352]

P. Becker and J. H. Correia. The ToscanaJ suite for implementing conceptual information systems. In *Formal Concept Analysis*, pages 324–348. Springer Berlin Heidelberg, 2005. URL https://doi.org/10.1007/11528784_17. [p351]

R. Belohlávek and V. Vychodil. Attribute dependencies for data with grades I. *International Journal of General Systems*, 45(7-8):864–888, 2016. URL https://doi.org/10.1080/03081079.2016.1205711. [p346, 347]

R. Belohlávek and V. Vychodil. Attribute dependencies for data with grades II. *International Journal of General Systems*, 46(1):66–92, 2017. doi: 10.1080/03081079.2016.1205712. URL https://doi.org/10.1080/03081079.2016.1205712. [p346, 347]

R. Belohlávek, P. Cordero, M. Enciso, A. Mora, and V. Vychodil. Automated prover for attribute dependencies in data with grades. *International Journal of Approximate Reasoning*, 70:51–67, 2016. URL https://doi.org/10.1016/j.ijar.2015.12.007. [p347]

G. Birkhoff. *Lattice Theory*, volume 25. American Mathematical Soc., 1940. [p348]

W. Chang. *R6: Encapsulated Classes with Reference Semantics*, 2020. URL https://CRAN.R-project.org/package=R6. R package version 2.5.0. [p352]

P. Cordero, M. Enciso, A. Mora, and I. P. de Guzmán. SL_{FD} logic: Elimination of data redundancy in knowledge representation. In *IBERAMIA*, volume 2527 of *LNCS*, pages 141–150. Springer, 2002. URL https://doi.org/10.1007/3-540-36131-6_15. [p346, 349]

P. Cordero, M. Enciso, A. Mora, and M. Ojeda-Aciego. Computing minimal generators from implications: a logic-guided approach. In *CLA 2012*, volume 972 of *CEUR W.Proc.*, pages 187–198. CEUR-WS.org, 2012. URL http://ceur-ws.org/Vol-972/paper16.pdf. [p347]

P. Cordero, M. Enciso, D. López, and A. Mora. A conversational recommender system for diagnosis using fuzzy rules. *Expert Systems with Applications*, 154:113449, 2020. ISSN 0957-4174. URL https://doi.org/10.1016/j.eswa.2020.113449. [p361]

C. Demko and K. Bertet. GALACTIC: Galois lattices, concept theory, implicational systems and closures. a set of python3 packages for studying formal concept analysis. *Theoretical Computer Science*, 845:1–20, 2020. [p351]

D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL https://doi.org/10.18637/jss.v040.i08. [p350]

M. B. First, R. L. Spitzer, M. Gibbon, and J. B. Williams. *User's Guide for the Structured Clinical Interview for DSM-IV Axis I Disorders SCID-I: Clinician Version*. American Psychiatric Pub, 1997. [p359]

B. Ganter and S. A. Obiedkov. *Conceptual Exploration*. Springer, 2016. ISBN 978-3-662-49290-1. URL https://doi.org/10.1007/978-3-662-49291-8. [p347, 350, 359]

B. Ganter and R. Wille. *Formal Concept Analysis - Mathematical Foundations*. Springer, 1999. ISBN 978-3-540-62771-5. URL https://doi.org/10.1007/978-3-642-59830-2. [p346]

J.-L. Guigues and V. Duquenne. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences humaines*, 95:5–18, 1986. [p349]

M. Hahsler, B. Grun, and K. Hornik. arules - a computational environment for mining association rules and frequent item sets. *Journal of Statistical Software*, 14:1–25, 2005. URL http://dx.doi.org/10.18637/jss.v014.i15. [p351]

T. Hanika and J. Hirth. Conexp-Clj - a research tool for fca. *ICFCA (Supplements)*, 2378:70–75, 2019. [p351]

D. I. Ignatov. Introduction to formal concept analysis and its applications in information retrieval and related fields. *CoRR*, abs/1703.02819, 2017. [p346]

L. Jezková, P. Cordero, and M. Enciso. Fuzzy functional dependencies: A comparative survey. *Fuzzy Sets and Systems*, 317:88–120, 2017. URL https://doi.org/10.1016/j.fss.2016.06.019. [p347]

S. O. Kuznetsov. Machine learning and formal concept analysis. In *ICFCA 2004*, volume 2961 of *LNCS*, pages 287–312. Springer, 2004. URL https://doi.org/10.1007/978-3-540-24651-0_25. [p346]

D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983. ISBN 0-914894-42-0. [p350]

D. Meyer. *registry: Infrastructure for R Package Registries*, 2019. URL https://CRAN.R-project.org/package=registry. R package version 0.5-1. [p358]

A. Mora, M. Enciso, P. Cordero, and I. P. de Guzmán. An efficient preprocessing transformation for functional dependencies sets based on the substitution paradigm. In *CAEPIA 2003*, volume 3040 of *LNCS*, pages 136–146. Springer, 2003. URL https://doi.org/10.1007/978-3-540-25945-9_14. [p347]

A. Mora, P. Cordero, M. Enciso, I. Fortes, and G. Aguilera. Closure via functional dependence simplification. *International Journal of Computer Mathematics*, 89(4):510–526, 2012. URL https://doi.org/10.1080/00207160.2011.644275. [p347, 349, 350]

J. M. Moyano and L. Sanchez Ramos. RKEEL: using KEEL in R code. *FUZZ-IEEE 2016*, pages 257–264, July 2016. URL http://dx.doi.org/10.1109/FUZZ-IEEE.2016.7737695. [p351]

L. S. Riza, C. Bergmeir, F. Herrera, and J. M. Benítez. frbs: Fuzzy rule-based systems for classification and regression in R. *Journal of Statistical Software*, 65(6):1–30, 2015. URL https://doi.org/10.18637/jss.v065.i06. [p351]

E. Rodríguez Lorenzo, K. Bertet, P. Cordero, M. Enciso, and A. Mora. The direct-optimal basis via reductions. In *CLA 2014*, volume 1252 of *CEUR W.Proc.*, pages 145–156. CEUR-WS.org, 2014. URL http://ceur-ws.org/Vol-1252/cla2014_submission_18.pdf. [p347]

E. Rodríguez Lorenzo, K. V. Adaricheva, P. Cordero, M. Enciso, and A. Mora. From an implicational system to its corresponding d-basis. In *CLA 2015*, volume 1466 of *CEUR W.Proc.*, pages 217–228. CEUR-WS.org, 2015. URL http://ceur-ws.org/Vol-1466/paper18.pdf. [p347]

G. Simpson and J. Angus. A rating scale for extrapyramidal side effects. *Acta Psychiatrica Scandinavica*, 45(S212):11–19, 1970. [p359]

M. Trabelsi, N. Meddouri, and M. Maddouri. A new feature selection method for nominal classifier based on formal concept analysis. In *KES 2017*, volume 112 of *Procedia Computer Science*, pages 186–194. Elsevier, 2017. URL https://doi.org/10.1016/j.procs.2017.08.227. [p346]

K. Ushey, J. Allaire, and Y. Tang. *reticulate: Interface to Python*, 2020. URL https://CRAN.R-project.org/package=reticulate. R package version 1.18. [p350]

P. Valtchev, D. Grosser, C. Roume, and M. R. Hacene. Galicia: an open platform for lattices. In *ICCS'03*, pages 241–254. Citeseer, 2003. [p351]

L. Wang, K. I. Alpert, V. D. Calhoun, D. J. Cobia, D. B. Keator, M. D. King, A. Kogan, D. Landis, M. Tallis, M. D. Turner, S. G. Potkin, J. A. Turner, and J. L. Ambite. Schizconnect: Mediating neuroimaging databases on schizophrenia and related disorders for large-scale integration. *NeuroImage*, 124: 1155–1167, 2016. URL https://doi.org/10.1016/j.neuroimage.2015.06.065. [p359]

H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. URL https://doi.org/10.21105/joss.01686. [p350]

R. Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In *Ordered Sets*, pages 445–470. Springer, 1982. [p346]

Y. Xie, J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida, 2018. ISBN 9781138359338. [p350]

S. A. Yevtushenko. System of data analysis Concept Explorer. In *Proc. 7th National Conference on Artificial Intelligence (KII'00)*, pages 127–134, 2000. [p351]

Pablo Cordero
Universidad de Málaga
Departamento de Matemática Aplicada
ETSI Telecomunicaciones, Campus de Teatinos
http://webpersonal.uma.es/de/pcordero/My_personal_web/Wellcome.html

ORCiD: 0000-0002-5506-6467
pcordero@uma.es

Manuel Enciso
Universidad de Málaga
Departamento de Lenguajes y Ciencias de la Computación
ETSI Informática, Campus de Teatinos
http://lcc.uma.es/~enciso/index_EN.html
ORCiD: 0000-0002-0531-4055
enciso@uma.es

Domingo López-Rodríguez
Universidad de Málaga
Departamento de Matemática Aplicada
ETSI Telecomunicaciones, Campus de Teatinos
https://dominlopez.netlify.app
ORCiD: 0000-0002-0172-1585
dominlopez@uma.es

Ángel Mora
Universidad de Málaga
Departamento de Matemática Aplicada
ETSI Informática, Campus de Teatinos
https://amorabonilla.github.io/
ORCiD: 0000-0003-4548-8030
amora@uma.es