Universidad de Granada

Data Science and Big Data Processing in R: Representations and Software

Tesis Doctoral

Lala Septem Riza

Granada, Julio de 2015

Programa de Doctorado en Tecnologías de la Información y la Comunicación

Departamento de Ciencias de la Computación e Inteligencia Artificial

Editorial: Universidad de Granada. Tesis Doctorales
Autora: Llala Septem Riza
ISBN: 978-84-9125-215-3
URI:http://hdl.handle.net/10481/40689

Universidad de Granada

Data Science and Big Data Processing in R: Representations and Software

MEMORIA QUE PRESENTA

Lala Septem Riza

PARA OPTAR AL GRADO DE DOCTOR EN
TECNOLOGÍAS DE LA INFORMACÍON Y LA COMUNICACIÓN

Julio de 2015

DIRECTORES

José Manuel Benítez Sánchez
Francisco Herrera Triguero

Departamento de Ciencias de la Computación
e Inteligencia Artificial

El doctorando Lala Septem Riza y los directores de la tesis los doctores D. José Manuel Benítez Sánchez y D. Francisco Herrera Triguero garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección de los directores de la tesis y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Granada, Julio de 2015

El Doctorando

Fdo: Lala Septem Riza

El Director

Fdo: D. José Manuel Benítez Sánchez

El Director

Fdo: D. Francisco Herrera Triguero

Acknowledgments

First and foremost I want to thank my supervisors: José Manuel Benítez Sánchez and Francisco Herrera Triguero. I really appreciate all their contributions of time, ideas, and supports to make my Ph.D. experience productive and challenging. I wish to thank various people for their contribution to this research: Christoph Bergmeir, Andrzej Janusz, Dominik Ślęzak, and Bartosz Krawczyk. I would also like to thank my committee members in Computer Science and Artificial Intelligence, Universidad de Granada, for comments and suggestions in my defense.

I gratefully acknowledge to the Dept. of Computer Science, Universitas Pendidikan Indonesia, for supporting me to pursue the Ph.D. program, and to the Directorate General of Higher Education of Indonesia, for providing a Ph.D. scholarship.

I am especially grateful for all members of "FuzzyCoffee": Fran, Manu, Diana, Pablo, Sergio, Sara, Rafa, Dani, Jorge, Alexandro, Julio, etc., for interesting discussions while sipping a cup of coffee and tea. I also want to acknowledge the "EAP 2010" group that always motivates me to accomplish Ph.D. To all friends that, thank you very much for everything, especially for the friendship.

And, last but not least, I would like to thank my family for love and encouragement. You are everything.

Man jadda wajada, wa man zara'a hasada, wa man yajtahid yanjah,
Alhamdulillah
— Muchas gracias —

Granada, 1 Syawal 1436

Resumen

El principal objetivo de esta tesis es el desarrollo de implementaciones en software de alta calidad y fácil uso de distintos paquetes para implementar representaciones y algoritmos para modelado de sistemas y análisis de datos. Por haberse convertido en un estándar *de facto*, la plataforma obligada para la implementación es el lenguaje R. Los paquetes referidos consideran distintas técnicas basadas en sistemas difusos, *Rough Sets*, y *Fuzzy Rough Sets*. Además, se presenta un formato de representación universal para sistemas basados en reglas difusas. Finalmente, se aborda la implementación de *random forests* y *random ferns* para procesamiento en Big Data. De acuerdo con estos objetivos, los resultados de la investigación son los siguientes:

1. El paquete "frbs": Se trata de un paquete en R que implementa representaciones de los principales tipos de sistemas basados en reglas difusas así como una selección de algoritmos de aprendizaje automático para construirlos. El paquete se centra en las aplicaciones para problemas de clasificación y regresión. También incluye un mecanismo para construir sistemas difusos directamente por expertos humanos. Está disponible en CRAN: `http://cran.r-project.org/package=frbs` y en el sitio web del proyecto: `http://sci2s.ugr.es/dicits/software/FRBS`.

2. El paquete "RoughSets": Se trata de un paquete en R que implementa algoritmos basados en *Rough Sets* y *Fuzzy Rough Sets* para la representación de conocimiento y análisis de datos. Incluye herramientas específicas para manejo de valores perdidos, discretización, selección de características, y selección de instancias. Está disponible en CRAN: `http://cran.r-project.org/package=RoughSets` y en el sitio web del proyecto: `http://sci2s.ugr.es/dicits/software/RoughSets`.

3. frbsPMML: Es un formato para la representación universal de de sistemas basados en reglas difusas que se basa en el *Predictive Model Markup Language*. Asociado a este formato, se implementan dos bibliotecas para manejar la representación: una extensión del paquete "frbs" y el paquete en Java "frbsJpmml".

4. El paquete "SparkFernTreeR": Es un paquete en R que implementa *random forests* y *random ferns* para el procesamiento de Big Data. Este paquete se desarrolla sobre las plataformas: Apache Hadoop y Apache Spark.

Summary

The main objective of this thesis is the development of high quality and easy to use software modules for represent, create and manage system models and data analysis. Since it has become a *de facto* standard, R is the platform of choice. The mentioned packages consider the techniques based on fuzzy systems, rough sets, and fuzzy rough sets. In addition, a universal representation framework for fuzzy rule-based systems is introduced. Finally, the implementation of random forests and random ferns for tackling Big Data is discussed. According to these objectives, the following are results of the research:

1. The "frbs" package: It is an R package implementing the most relevant types of fuzzy rule-based systems along with a selection of machine -learning algorithms to build them. The package focuses on classification and regression tasks. It also includes a mechanism to allow the construction of a model by human experts. It is available in CRAN: `http://cran.r-project.org/package=frbs` and in the project website: `http://sci2s.ugr.es/dicits/software/FRBS`.

2. The "RoughSets" package: It is an R package implementing algorithms based on rough set theory and fuzzy rough set theory for knowledge representation and data analysis. In includes tools for managing missing values, discretization, feature selection, and instance selection, for both classification and regression tasks. It is available in CRAN: `http://cran.r-project.org/package=RoughSets` and in the project website: `http://sci2s.ugr.es/dicits/software/RoughSets`.

3. frbsPMML: It is a universal representation framework for fuzzy rule-based systems based on the Predictive Model Markup Language. Furthermore, two software libraries to manage the representation are im-

plemented: an extension of the "frbs"package and the Java package "frbsJpmml".

4. The "SparkFernTreeR" package: It is an R package implementing random forests and random ferns for dealing with Big Data processing. This package is developed on top of the Big Data frameworks: Apache Hadoop and Apache Spark.

Contents

List of Figures

List of Tables

Table of Acronyms

FRBSs	—	Fuzzy Rule-Based Systems 10
RST	—	Rough Set Theory 10
FRST	—	Fuzzy Rough Set Theory 11
RFs	—	Random Forests 11
RFe	—	Random Ferns 11
CRAN	—	The Comprehensive R Archive Network 11
DTs	—	Decision Trees 12
PMML	—	The Predictive Model Markup Language 13
GPL	—	General Public License 27
XML	—	Extensible Markup Language 35
IEEE	—	The Institute of Electrical and Electronics Engineers ... 35
TSK	—	Takagi Sugeno Kang 45
FRBCSs	—	Fuzzy Rule-Based Classification Systems 47
WM	—	Wang and Mendel's Technique 49
FRBCS.CHI	—	FRBCS Using Chi's method 49
FRBCS.W	—	FRBCS Using Ishibuchi's Method with Weight Factor .. 49
FNN	—	Fuzzy Neural Networks 50
ANN	—	Artificial Neural Networks 50
ANFIS	—	The Adaptive-Network-Based Fuzzy Inference System .. 50
HYFIS	—	The Hybrid Neural Fuzzy Inference System 50
GFS	—	Genetic Fuzzy Systems 50
GFS.THRIFT	—	GFS Based on Thrift's Method 50
GFS.FR.MOGUL	—	GFS Based on the MOGUL Methodology 50
GFS.GCCL	—	Ishibuchi's Method Based on Genetic Cooperative Competitive Learning 50
FH.GBML	—	Ishibuchi's Method Based on Hybridization of GFS.GCCL and Pittsburgh 50
SLAVE	—	Structural Learning Algorithm on Vague Environment . 51
SBC	—	Subtractive Clustering 51
DENFIS	—	Dynamic Evolving Neural Fuzzy Inference System 51
FIR.DM	—	Fuzzy Inference Rules with Descent Method 51
FS.HGD	—	FRBS Using Heuristics and the Gradient Descent Method 51
RDD	—	Resilient Distributed Datasets 76

Introducción

A. Planteamiento

Hoy en día, se generan datos a un ritmo imparable en prácticamente todos los ámbitos de la actividad humana. Los datos se producen en gran cantidad, con una alta velocidad con formatos complejos y desde múltiples fuentes. Por ejemplo, se están generando datos digitalizados desde las administraciones públicas, Internet de las Cosas (IoT, por ejemplo, GPS), teléfonos móviles, PDAs, medios sociales, sensores, aplicaciones de negocios, webs públicas, etc. Por otra parte, además del aspecto de la dimensión de los datos, la mayoría de los datos contienen información incierta, ruidosa, incompleta e irrelevante. Como las dos caras de una moneda, este fenómeno ofrece dos lados opuestos. En primer lugar, las herramientas y algoritmos disponibles tienen dificultades para manejar dichos datos de manera eficiente. Por otro lado, esta situación puede explotarse exitosamente si somos capaces de extraer conocimiento a partir de todos esos datos. Estas circunstancias están atrayendo a muchos investigadores y profesionales de todo el mundo, interesados en desarrollar metodologías sistemáticas que permitan manejar los datos de manera eficiente. Recientemente, el término "ciencia de datos" (*Data Science*) se utiliza ampliamente para designar en este campo.

La Ciencia de Datos es el estudio que se centra en la extracción de conocimiento a partir de los datos [66]. Para extraer conocimiento útil, los datos se procesan de forma sistemática en varios pasos, tales como su recopilación, preparación, análisis, visualización, administración y almacenamiento de grandes cantidades de información [266]. Los científicos de datos utilizan técnicas y teorías de distintos ámbitos del conocimiento. Por ejemplo, aprendizaje automático, que es el campo de estudio científico que se centra en los algoritmos que son capaces de aprender de los datos [160],

y es un componente importante para llevar a cabo el análisis de datos. Del aprendizaje automático se usan muchos algoritmos para abordar distintas etapas de la ciencia de datos, tales como el pre-procesamiento, modelado, etc. Otros elementos relevantes en la ciencia de datos son las herramientas informáticas y tecnologías, estadísticas, algoritmos de optimización, sistemas de gestión de bases de datos y lenguajes de programación.

En aprendizaje automático, podemos encontrar muchos métodos para la construcción de modelos predictivos. Puesto que los datos no siempre son fiables, los enfoques deben ser robustos en el manejo de la información imperfecta, imprecisa, con ruido y redundante. Entre las diversas propuestas científicas, hay dos teorías que destacan en este ámbito: la Teoría de Conjuntos Difusos [306] y la Teoría de *Rough Sets* (RST) [213]. Ambas se incluyen en el ámbito de la Inteligencia Computacional [72], que es una subrama de la Inteligencia Artificial que estudia los mecanismos de adaptación con inspiración lingüística o biológica para desarrollar sistemas con comportamiento inteligente en entornos complejos y cambiantes.

Los sistemas difusos constituyen una extensión de la teoría de conjuntos clásica surgida para modelar y representar el conocimiento humano. El concepto conjunto difuso introduce un grado de pertenencia que en vez de plantear sólo dos posibles valoraciones, miembros o no miembros, un objeto puede ser clasificado en una categoría determinada, con un grado que puede variar continuamente entre cero y uno. Por otra parte, el empleo de los grados de pertenencia permite emplear los conjuntos difusos para representar de forma efectiva el lenguaje natural, aquejado de vaguedad e imprecisión [307]. Sobre la base de los sistemas difusos, los sistemas basados en reglas difusas (SBRDs), son representaciones del conocimiento experto humano en términos de un conjunto de reglas difusas. En los SBRDs, en lugar de utilizar valores numéricos, una regla contiene una expresión en forma "Si A entonces B", donde A y B son conjuntos difusos. Un beneficio importante de SBRDs es que las representaciones basadas en conjuntos difusos son mucho más fáciles de ser interpretadas por humanos que las reglas clásicas (es decir, reglas numéricas).

RST fue introducida por Pawlak en 1982 [213] como una metodología para el análisis de datos basado en la aproximación de los conceptos en sistemas de información. Se centra en torno a la noción de discernibilidad: la capacidad de distinguir entre objetos, basada en los valores de sus atributos. Dada una relación de indiscernibilidad, podemos construir las aproximaciones inferior y superior de los conceptos. Los objetos incluidos

en la aproximación más baja se pueden clasificar con certeza como miembros del concepto. En contraste, la aproximación superior contiene objetos posiblemente pertenecientes al concepto. RST se ha generalizado de muchas maneras para abordar diversos problemas. En particular, en 1990, Dubois y Prade [68] combinaron los conceptos de vaguedad expresada por los grados de pertenencia de los conjuntos difusos [306] e indiscernibilidad en RST para obtener la teoría de *Fuzzy Rough Sets* (FRST). FRST permite una pertenencia parcial de un objeto a las aproximaciones inferior y superior, y por otra parte, la igualdad aproximada entre los objetos se puede modelar mediante relaciones difusas de indiscernibilidad.

Otra herramienta que se requiere en la ciencia de datos es un lenguaje de programación para desarrollar bibliotecas de software. Por lo general, los científicos de datos trabajan en plataformas interactivas que permiten realizar de forma relativamente fácil tareas gráficas, informes y análisis. MATLAB, Python, R, SAS y SPSS son lenguajes de programación que cumplen con estos requisitos, por lo que se pueden utilizar en el análisis de datos. Por supuesto, cada uno de ellos tiene sus ventajas y desventajas. Todos tienen comunidades amplias de usuarios que las emplean, pero en cuanto a popularidad y uso una de ellas se ha convertido en el estándar *de facto* para el análisis de datos: R. Entre las principales ventajas de R destacan que es un software de código abierto —Python también es de código abierto— y tiene una comunidad sólida que produce y mantienen una gran cantidad de paquetes, que están disponibles en el Comprehensive R Archive Network (CRAN). La popularidad de R entre los analistas de datos se constata a través de una encuestarealizada por KDnuggets [152] que muestra que R alcanza el primer lugar como el lenguaje de programación utilizado para análisis/minería de datos/ciencia de datos durante los últimos años.

Por otra parte, como indicábamos anteriormente los datos recopilados crecen a un ritmo vertiginoso. Este fenómeno ha dado lugar al término "Big Data." En 2012, la agencia Gartner [29] indicó: "Big Data es gran volumen, alta velocidad y alta variedad de activos de información que exigen, formas innovadoras y rentables de procesamiento de la información para mejorar la comprensión y la toma de decisiones". Hay dos cuestiones relacionadas con Big Data. En primer lugar, actualmente las computadoras y los algoritmos no puede manejar grandes conjuntos de datos de manera eficiente. En segundo lugar, los sistemas de gestión de almacenamiento actuales se enfrentan a esos mismos problemas. Han surgido multitud de plataformas para plantear soluciones frente al almacenamiento y gestión de datos en Big Data, pero entre todas ellas destacan dos: Apache Hadoop

[255, 199] y Apache Spark [308, 150]. Mientras Hadoop ofrece tres componentes principales: un sistema distribuido de archivos, un modelo de programación llamado MapReduce, y un sistemas de gestión de recursos, Apache Spark ofrece un modelo computacional basado en el concepto central de conjuntos de datos distribuidos resilientes (RDD). Además, hay un paquete en R que integra R [124] con Apache Spark y que permite utilizar mandatos R de manera distribuida. El paquete se llama "SparkR"(`http://amplab-extras.github.io/SparkR-pkg/`).

Como se mencionó anteriormente, en la era de Big Data, necesitamos extender algoritmos convencionales para que puedan ser utilizados sobre las plataformas de Big Data. Entre los diversos algoritmos conocidos existen dos que se pueden extender de forma natural. Son *random forest* (RFS) [33] y *random ferns* (RFE) [212]. El primero es un conjunto de métodos para la construcción de un *ensemble* de árboles de decisión (DTS) [35] que se generan a partir de *bootstrap* y características aleatorias. El segundo, RFe, es un método que emplea *bootstrap* y clasificación Naive Bayes [212]. En otras palabras, el método reemplaza DTs con ferns no jerárquicos basados en la fórmula de Bayes.

Por otra parte, a pesar de que los algoritmos de aprendizaje automático, tales como los basados en SBRD, RST, FRST, DTs, RF y RFe están disponibles, su uso en situaciones prácticas no es un proceso sencillo. Las bibliotecas de software para ciencia de datos son necesarias debido a las siguientes razones:

1. *Dificultad de implementación.* Si bien en cada propuesta nueva de algoritmos y métodos éstos deben ser detallados de forma suficiente como para permitir que un usuario interesado en implementarlas pueda hacerlo, no siempre es tan fácil. Puede ocurrir que los usuarios no tengan la habilidad o conocimiento necesario para lograrlo. O tal vez que los autores no incluyeran detalles suficientes para producir una implementación fiel. Una implementación en software es una representación específica de los métodos en una forma lista para ser usada.

2. *Audiencia más amplia.* Una implementación de un método en código fuente proporciona una forma alternativa de entender realmente los algoritmos. Esto se complementa con la documentación que debe acompañar a cada paquete de software. Además, esto permitirá el acceso a los métodos a una audiencia más amplia de usuarios. Por lo tanto, las bibliotecas de software como las que se proponen en esta investigación ayudarían al usuario a realizar un análisis de datos con las técnicas abordadas.

3. *Investigación reproducible.* Con el fin de verificar los resultados publicados por los investigadores es necesario reproducir los experimentos que se publican. Si el mismo software utilizado para el estudio experimental está disponible, la verificación se puede realizar con más precisión y más rápidamente.

4. *Falta de una representación portable.* Mientras que los modelos producidos con las técnicas consideradas pueden ser desplegados en distintas plataformas, la utilización en un conjunto más amplio de plataformas podría ser mejorada si se definiera y desarrollase un formato de representación universal. Este formato estándar es una necesidad si se desea compartir y comunicar los modelos con otras personas, como por ejemplo, otro grupo de investigadores u otros usuarios de diferentes departamentos de una organización.

5. *Licencia de código abierto.* El conocimiento y los resultados producidos con fondos públicos deben estar universalmente disponibles. Esto no es sólo una buena idea, sino que también es un requisito para los programas de investigación de la mayoría de los países y de las organizaciones públicas que sufragan la investigación. Cuando se trata de software, se han publicado múltiples tipos de licencias "open-source". La mayoría de ellos permiten un uso libre que, básicamente, implica a los usuarios a utilizar y extender el software publicado bajo esta licencia.

De acuerdo con los párrafos previos, se requiere una investigación centrada en la implementación e integración de los algoritmos anteriores en bibliotecas software de alta calidad y fáciles de usar. Por tanto, los investigadores y los profesionales que los usen pueden hacer frente a sus tareas de ciencia de datos más eficazmente.

B. Objetivos

En general, esta investigación tiene como objetivo implementar representaciones y algoritmos de técnicas de análisis de datos en bibliotecas de software y diseñar un estándar de representación universal para algunos modelos. Este objetivo general se puede desglosar en los siguientes:

1. Desarrollar paquetes en R para abordar tareas de análisis de datos

(concretamente, pre-procesamiento de datos, clasificación y regresión) basados en métodos de inteligencia computacional. El objetivo se circunscribe a dos técnicas principales:

   (a) Sistemas basados en reglas difusas. Se pretenden desarrollar representaciones y algoritmos de aprendizaje automático para SBRDs. Estos sistemas se dirigirán a la resolución de problemas de clasificación y regresión. Además de los procesos automáticos, se ofrecerá funcionalidad para la construcción de SBRDs por expertos humanos.

   (b) *Rough Sets* (RST) y *Fuzzy Rugh Sets* (FRST). Se pretende desarrollar un paquete extenso con los principales algoritmos basados en RST y FRST utilizados para el pre-procesamiento de datos (es decir, discretización, selección de características, y la selección de instancias), clasificación y regresión.

2. Para diseñar un estándar de representación universal para SBRDs basado en el *Predictive Model Markup Language* (PMML). Se complementará con paquetes para manejar y explotar la representación en distintas plataformas.

3. Para desarrollar un paquete en R para el procesamiento de Big Data que implemente dos algoritmos conocidos: RF y RFe. El paquete debe ser interoperable con Apache Hadoop y Apache Spark.

C. Resumen

Para detallar cómo se han alcanzado los objetivos propuestos, esta tesis se estructura en cinco capítulos principales, uno adicional con las principales conclusiones y notas finales y, finalmente, tres anexos. A continuación, se presenta un resumen de los respectivos capítulos.

En el capítulo 1 se incluye una introducción a cada tema relacionado con la tesis y el estado de la técnica. En primer lugar, se explican los conceptos de Ciencia de Datos. Un tema que está fuertemente relacionado con la ciencia de datos, el Aprendizaje Automático, se presenta en la Sección 1.2. Después de eso, se detallan el lenguaje R y su ecosistema, PMML, sistemas difusos, RST y FRST, DTs y RFs, RFe, procesamiento de Big Data y sus plataformas.

El capítulo 2 pretende responder al Objetivo 1(a). El resultado obtenido es el paquete de R "frbs," basado en SBRDs, para abordar los problemas de clasificación y regresión. Este capítulo comienza con una introducción al paquete. Posteriormente, se indica la arquitectura del paquete y los detalles de implementación. Después se ilustra su uso. Le siguen estudios experimentales sobre su eficacia y una comparación con otras bibliotecas de software. Por último, se presenta de forma concisa un resumen del capítulo.

frbsPMML, que es una propuesta de formato estándar para representación universal de SBRDs basado en PMML. Se detalla en el capítulo 3, cuyo objetivo es abordar el Objetivo 2. El capítulo se estructura en cuatro secciones. Después de la introducción de frbsPMML, se presentan en detalle las especificaciones de la representación de SBRDs. Además, se presentan dos módulos que proporcionan los motores de explotación en R y Java. Las características y los beneficios de frbsPMML junto con un resumen del capítulo se discuten en las secciones finales.

El capítulo 4 detalla el trabajo realizado para alcanzar el Objetivo 1(b). El resultado de este trabajo es el paquete "RoughSets". Es un paquete de R basado en RST y FRST para abordar las siguientes tareas: discretización, selección de características, selección de instancias, clasificación y regresión. Básicamente, la estructura de este capítulo es similar a la del Capítulo 2. Contiene una introducción, la arquitectura del paquete y su implementación, ejemplos de uso, y una comparación con otros paquetes. Por último, se presenta un breve resumen del capítulo.

En el capítulo 5, se presenta un paquete de R utilizado para el procesamiento de Big Data, que completa el tercer objetivo. Se centra en la implementación de algoritmos basados en RF y RFe. Este capítulo contiene las siguientes secciones: una introducción, la arquitectura e implementación de los paquetes y los ejemplos que muestran cómo utilizar el paquete. Por último, la sección de resumen concluye el capítulo.

En el sexto capítulo se incluyen las observaciones finales, que se estructuran en: resúmenes generales, las publicaciones asociadas, y el trabajo futuro.

Finalmente, los apéndices relacionados con "FRBS," "RoughSets," y "SparkFernTreeR" se presentan para proporcionar materiales detallados relacionados con la investigación. En estos apéndices, se incluyen algunos datos relacionados con el proceso de ingeniería del software seguido durante el desarrollo del trabajo descrito en esta memoria. Después de eso, en la parte final de la tesis aparece toda la bibliografía consultada durante la investi-

gación.

Introduction

A. Problem Statement

Nowadays, the flood of data can no longer be dammed so that it inundates every corner of our daily activities. In other words, data have been produced in a high amount, a high speed with complex formats and from various sources. For example, we have been generating data from digitized government administrations, Internet of Things (IoT, e.g., GPS), mobile phone, PDA, social media, sensors, business applications, public webs, etc. Furthermore, besides the aspect of data dimension, most data contain uncertain, noisy, incomplete, and irrelevant information. As head and tail on a coin, this phenomenon offers two opposing sides. First, it brings emerging problems since available tools and algorithms have difficulties to handle such data efficiently. However, at the same time, this condition offers great advantages if we are able to extract knowledge from data, e.g., to make a better decision, predict a future action, describe a current situation, etc. These problems and advantages have been attracting many researchers and practitioners around the world to provide systematic methodologies dealing with data efficiently. Recently, the term "data science" has become widely used for expressing the research.

Data science is the study that focuses on knowledge extraction from data [66]. In this study, to extract useful knowledge, data are systematically processed in several steps, such as data collection, preparation, analysis, visualization, management, and preservation of large collections of information [266]. To accomplish an objective perfectly, data scientists involve techniques and theories contributed by many fields. For example, machine learning, which is the field of scientific study focusing on algorithms that are able to learn from data [160], is a major component to perform data

analysis. It promises a huge number of learning algorithms that can be used for tackling various stages in data science, such as data pre-processing, data analysis, etc. Other factors that should be considered in data science are computing tools and technologies, statistics, optimization algorithms, database management systems, and programming languages.

In machine learning, one can find many sophisticated methods for constructing predictive models. Since data are not always reliable, approaches should be robust in handling imperfect, vague, noisy, and redundant information. In the literature, there are two concepts promising these capabilities: fuzzy systems [306] and rough set theory (RST) [213]. These concepts are usually included in the term Computational Intelligence [72], which is a sub-branch of Artificial Intelligence that studies adaptive mechanisms to enable or facilitate intelligent behavior in complex and changing environments.

Fuzzy systems propose an extension of the classical set theory for modeling and representing human knowledge. The concept introduces a degree of membership which means that instead of just having two qualifications: member or non-member, an object can be classified into a certain category with a degree between zero and one. Therefore, fuzzy sets can be viewed as a generalization of the classical sets. Furthermore, employing the degrees of membership makes fuzzy set theory effectively works on natural languages containing vagueness and imprecision [307]. Based on fuzzy systems, fuzzy rule-based systems (FRBSs), which are methodologies to represent human expert in a set of fuzzy rules, are introduced. In FRBSs, instead of involving numerical values, a fuzzy rule contains an expression in the form "IF A THEN B" where A and B are fuzzy sets. A main benefit of FRBSs is that the representations based on fuzzy sets are much easier to be interpreted by human than classical rules (i.e., numerical rules).

RST was introduced by Pawlak in 1982 [213] as a methodology for data analysis based on the approximation of concepts in information systems. It revolves around the notion of discernibility: the ability to distinguish between objects, based on their attribute values. Given an indiscernibility relation, we can construct lower and upper approximations of concepts. Objects included in the lower approximation can be classified with certainty as members of the concept. In contrast, the upper approximation contains objects possibly belonging to the concept. RST has been generalized in many ways to tackle various problems. In particular, in 1990, Dubois and Prade [68] combined concepts of vagueness expressed by membership degrees in

fuzzy sets [306] and indiscernibility in RST to obtain fuzzy rough set theory (FRST). FRST allows partial membership of an object to the lower and upper approximations, and moreover, approximate equality between objects can be modeled by means of fuzzy indiscernibility relations.

Another tool that is required in data science is a sufficient programming language in order to build a software libraries. Usually data scientists work on interactive platforms so that plotting, summarizing, and analyzing can be done in a relatively easy manner. MATLAB, Python, R, SAS, and SPSS are programming languages met with these requirements, so they can be used in data analysis. Of course, they promise some advantages along with drawbacks at the same time; it depends on the tasks, objectives, and situations at hand. The main advantages that are specific to R are that it is an open source software — Python is also open source — and has a solid community that contributes a huge number of packages, which are available at the Comprehensive R Archive Network (CRAN). This statement is emphasized by a survey conducted by KDnuggets [152] showing that R has been taking the first place for the programming language used for an analytics/data mining/data science for the last years.

Furthermore, the flooding of data is not stopping any time soon; data go on growing exponentially. Thus, the term Big Data was introduced by computer scientists several years ago. In 2012, Gartner [29] says: "Big Data is high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making." There are two issues related to Big Data. First, current computers and algorithm cannot handle massive datasets efficiently. Second, current storage management systems face the same problems. There are two famous frameworks available currently: Apache Hadoop [255, 199] and Apache Spark [308, 150]. While Apache Hadoop offers three main components: a distributed file system, the programming model called MapReduce, and resource management systems, Apache Spark offers on in-memory computational model based on the central concept of Resilient Distributed Datasets. Additionally, there is an R package that integrates R [124] with Apache Spark and enables native R commands in a distributed fashion, named "SparkR" (`http://amplab-extras.github.io/SparkR-pkg/`).

As mentioned above, in the Big Data era, we need to extend conventional algorithms so that they can be utilized on top of the Big Data frameworks. Two methods that can be naturally extended are random forests (RFs) [33] and random ferns (RFe) [212]. The first one is a set of methods constructing

a model by assembling decision trees (DTs) [35] that are generated from bootstrap samples and randomized features. Moreover, a DT is an algorithm that constructs a tree by performing a recursive partition of the instance space. Then, RFe is a method employing bootstrap sampling and Naive Bayesian classification [212]. In order words, the method replace DTs with non-hierarchical ferns based on Bayes's formula.

Furthermore, even though machine-learning algorithms, such as those based on FRBS, RST, FRST, DTs, RFs, and RFe have been available, using them in practical situations is not a straightforward process. Software libraries for Data Science is a necessity because of the following reasons:

1. *Difficulty of implementation.* While new proposal for algorithms and methods should be detailed with enough detail to allow an interested reader to implement them, it is not always that easy. May the readers do not have the skill or knowledge necessary to accomplish it. Or maybe that the authors did not include enough detail to produce a faithful implementation. An implemented software is a specific representation of the methods in a way ready to use.

2. *Wider audience.* An implementation of a method in source code provides an alternative way to actually understand the algorithms. This complemented with the documentation that should accompany every piece of software. This will enable the access to the methods to a wider audience of users. Thus, software libraries like the one proposed in this research would help user perform data analysis with addressed techniques.

3. *Reproducible research.* In order to verify the results published by researchers it is necessary to reproduce the experiments they report. If the very same software used for the experimental study is available, the verification can be done more accurately and faster.

4. *Lack of portable representations.* While the models produced with the techniques considered could be deployed in a number of different platforms, the used along those platforms could be boosted if a universal representation format were defined and developed. This standard format is a must if one wants to share the models and communicate them with other people, just like other group of researchers or users in different departments of an organization.

5. *Open-source license.* The knowledge and result produced out of public

funds should be universally available. This is not only a good idea, but also a requirement by most countries' and public organizations' research programs. When it comes to software a number of different "open-source" license have been published. Most of them allow for a free usage that basically entails users to use and extend the software published under them.

According to the above explanation, a research focusing on implementing and integrating the above algorithms into high quality and easy to use software libraries is required. Thus researchers and practitioners that use them can cope with their data science tasks efficiently.

B. Objectives

In general speaking, this research aims to implement several well-known concepts and associated algorithms to deal with data analysis in software libraries and to design a universal representation framework for some models. This general objective can be described in detail as follows:

1. To develop R packages used for tackling tasks in data analysis (i.e., data pre-processing, classification, and regression) based on computational intelligence methods. The following are two concepts considered:

   (a) Fuzzy rule-based systems (FRBSs). The research aims to implement models and several machine-learning algorithms based on FRBSs for dealing with classification and regression. Furthermore, a functionality for constructing an FRBS model by human experts is required to be implemented.

   (b) Rough set theory (RST) and fuzzy rough set theory (FRST). This objective is to implement several algorithms based on RST and FRST used for data preprocessing (i.e., discretization, feature selection, and instance selection), classification, and regression.

2. To design a universal representation framework of FRBS models based on the predictive model markup language (PMML). It should be multi platforms. To achieve this, prediction engines complying with the representation would be developed.

3. To develop an R package for Big Data processing that implements two well-known algorithms: RFs and RFe. The package should be interoperable with Apache Hadoop and Apache Spark.

C. Structure of the Document

To describe how the proposed objectives have been reached, this thesis is organized in five main chapters, another one with concluding remarks, and three appendices. In the following, a summary of each chapter is presented.

In Chapter 1, an introduction to each related topic and state of the art are discussed. Firstly, we explain concepts of data science. A topic that is strongly related to data science, which is machine learning, is presented in Section 1.2. After that, the R language and its ecosystem, PMML, fuzzy systems, RST and FRST, DTs and RFs, RFe, Big Data processing and its platforms are discussed.

Chapter 2 aims to answer Objective 1(a). The obtained result is the R package "frbs", based on FRBSs, for dealing with classification and regression problems. In this chapter, an introduction to the package is firstly explained, and the package architecture and implementation details follow. The usage of the package, experimental studies, and a comparison with other software libraries are explained as well. Finally, it concisely presents a summary of the chapter.

frbsPMML, which is a universal representation framework for FRBSs based on PMML, is presented in Chapter 3. The chapter is intended to fulfill Objective 2. There are four sections included in the chapter. After introducing frbsPMML, specifications of the representation of FRBS models are presented in detail. Furthermore, we present two modules providing prediction engines that comply with the standard framework. Features and benefits of frbsPMML along with a summary of the chapter are discusses in the final sections.

Chapter 4 attempts to carry out Objective 1(b) by presenting the "RoughSets" package. It is an R package based on RST and FRST to cope the following tasks: discretization, feature selection, instance selection, classification, and regression. Basically, the structure of this chapter is similar to that of Chapter 2. It contains an introduction, the package architecture and its implementation, examples of usage, and a comparison with other

packages. Finally, a brief summary of the chapter is presented.

In Chapter 5, an R package used for Big Data processing is presented for completing the third objective. It focuses on implementing algorithms based on RFs and RFe. This chapter contains the following sections: an introduction, the package architecture and implementation and examples showing how to use the package. Then the summary section concludes the chapter.

The sixth chapter includes the concluding remarks, that are structured in: general summaries, the associated publications, and future work.

Finally, the appendices related to "frbs," "RoughSets," and "SparkFern-TreeR" are presented to provide detailed materials related to the research. In these appendices, several data related to the software engineering process followed during the development of the research are included. After that, all the consulted references are listed in the end of the thesis.

Chapter 1

State of the Art

This chapter reviews the main topics involved in the research. Firstly, we introduce two important aspects that are related to each other: data science and machine learning. After that the R programming language and its ecosystem is presented. Moreover, an explanation on how to develop a package in R is also depicted. Next, we briefly introduce PMML and its benefits for providing interoperability. For machine-learning methods, we discuss several algorithms included in FRBS, RST, FRST, DTs, RFs, and RFe that will be implemented in this research. Finally, a brief background and platforms used for the Big Data processing are presented.

1.1 A Gentle Introduction to Data Science

A well known aphorism says: "Who ever control information and knowledge, will actually rule the world." Recently, this proverb is closer to the reality, especially from the business perspective, one can realize this by having a look at the following examples. Google become the most popular website after providing the search-engine application for 1.17 billion people. By applying sophisticated algorithms for recognizing patterns of friendship relationships, Facebook and Linkedln are the biggest players providing social media. Amazon gains many benefits from retail-business transactions, not only from the transactions but also from knowledge that is extracted from customer database and their behaviors. Therefore, it can be seen clearly that analyzing data is required to be ahead of the competence. Unfortunately,

it is not always an easy work because of some reasons. For example, data can be represented in many formats (e.g., structured and unstructured), generated from various sources (e.g., sensors, mobile phones, webs, etc.), and contain noisy, redundancy, irrelevance, and uncertainty. Therefore, a systematical study should be done to extract knowledge and useful information from data so that we can make a better decision, predict future events, identify and describe unseen patterns, etc. Data science aims to tackle these objectives.

Several definitions of data science have been coined by researchers in several disciplines. In the study [113], data science is not only a synthetic concept to unify statistics, data analysis, and their related methods but also comprises its results. Basically, it includes three steps: design for data, collection of data, and analysis on data. Data science can be also defined as an emerging area of work concerned with the collection, preparation, analysis, visualization, management, and preservation of large collections of information [266]. Furthermore, this study mentions that there are the four A's of data: data architecture, data acquisition, data analysis, and data archiving. More general definition provided by [66] states that data science is the study that focuses on extraction knowledge from data. Furthermore, an introduction to data science by considering the history perspective can be found in [227].

In detail, the study in [156] describes the activities of data scientists as follows:

- Data collection:
    1. Data engineering platform: building a system required for collecting data from multiple sources continuously.
    2. Telemetry injection: inserting instrumentation code to gather software execution and usage profiles.
    3. Experimentation platform: developing a system for supporting experimentation with various software.
- Data analysis:
    1. Data merging and cleaning: focusing on joining data from different sources, dealing with missing values, and incomplete data.
    2. Sampling: selecting a subset of available data by considering behavior of complete data.

3. Data shaping including selecting and creating features: transforming original data into a user-defined format.
4. Defining sensible metrics: building measurements that are sensible to data consumers.
5. Defining ground truths: defining class labels and scenarios of anomalies.
6. Building predictive models: building a model used for prediction by employing machine learning, data mining, statistics, etc.
7. Hypothesis testing: determining a hypothesis and doing a test of it based on statistical methods.

- Use and dissemination:

1. Operationalization of predictive models: integrating predictive models into software products and systems by performing right models at a right condition.
2. Building automatic systems: defining automated actions and triggers for different situations of predictions.
3. Translating insight and models to business values: delivering the value of insights and predictive models using domain-specific terms to end users.

According to Conway [51] as in Figure 1.1, data science is an interdisciplinary field that requires hacking skills (i.e., programming), math and statistics knowledge, and substantive expertise in a field of science. Furthermore, machine learning, which will be explained in the next section, is a major component that provides important contributions for data science. Other factors that should be considered in data science are computation tools and technologies, optimization algorithms, database management systems, etc. Furthermore, a capability to understand the context of problems at the hand is also required by data scientists.

1.2 Machine Learning

This section briefly explains about concepts of learning in general. Then, definitions of machine learning and its applications are presented.

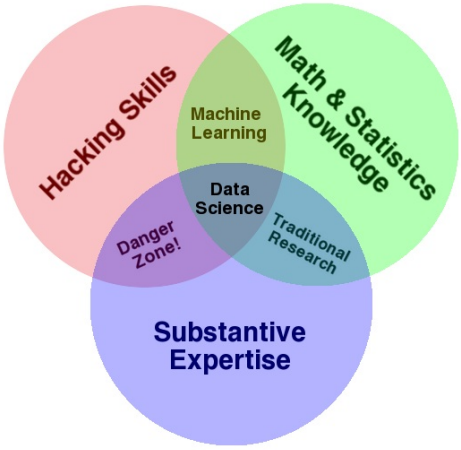


Figure 1.1: The data science venn diagram [51].

1.2.1 Learning

For many years, the learning process in the organisms, such as humans and animals, has been attracting many researchers, especially psychologists. It can be understood because we have been doing it since our childhood. For example, scientists study how we can speak, walk, swim, write, and etc., which results out of our learning from people around us. Furthermore, the ability of learning is not exclusive of humans, but animals and other organisms also have it. With good training, a dog can help humans to do some jobs, such as to pick up newspapers, to follow some instructions, etc. These daily life activities challenge scientists to understand what learning is.

There exists two well-known definitions of learning: behaviorism and constructivism. These theories differently explain the main components and processes involved in learning. The first one defines learning as changes in the behavior of an organism that are the result of regularities in the environment [256, 257]. A simple example describing the definition is on dog training. A dog will be fed if it can do certain activities, such as sit and jump, otherwise we do not give it anything. Therefore, the behaviorism theory is quite related to reinforcement learning containing two components: reward and punishment. The second perspective is called constructivism and refers to a view on learning focused on how organisms actively construct knowledge

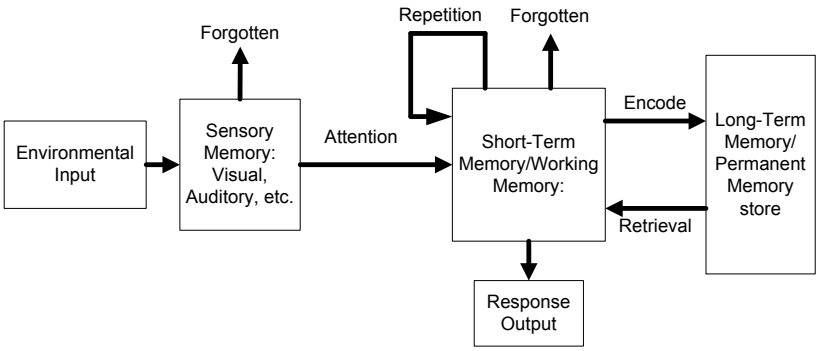


Figure 1.2: Information processing model [17].

out of experiences [223]. In other words, learning is a process to generate and construct knowledge from an interaction between their experience and minds. According to this theory, there are two processes involved, which are assimilation and accommodation. Assimilation means to incorporate a new experience with existing knowledge without changing its global structure. This may happen when the experience is aligned with the knowledge. For example, a preschool child who already understands the concept of fish might initially label any animal living in the water with the term "fish," even for a whale or a dolphin. In contrast, accommodation refers to the process of integration and adaptation of new experiences by utilizing them into existing knowledge effectively. A child who initially generalizes the concept of fish, which is any animals living in the water, eventually revise the concept by adding other characteristics of fish, as e.g., any animals living in the water having gills.

Another interesting concept related to the learning process is information processing theory. It is a theory addressing how humans process, store, structure, and respond to information they receive through their senses [201]. A component that becomes the main concern on information processing model is a memory. From among available models, Atkinson and Shiffren's model [17] of the role of memory is the most popular in an information processing system as illustrated in Figure 1.2.

It can be seen that sensory memory provides initial screening and processing of incoming stimulus for very brief periods of time, usually on the order of 1/2 to 3 seconds. The amount of information held at any given time in sensory memory is limited to five to seven discrete elements, such as voice, color, and human faces. After stimuli enter sensory memory, they are either

forwarded to working memory or deleted from the system. It is a term that is used to refer to a multi-component temporary memory system in which information is assigned meaning, linked to other information, and essential mental operations such as inferences are performed. Duration of retention of working memory is 5 up to 15 seconds unless some type of mental rehearsal occurs. The amount of information stored in the memory is limited to 7 up to 9 units of information. The last memory is long-term memory, which is a memory used for placing all previous perceptions, knowledge, and information learned by an individual. Unlike sensory and working memory, long-term memory is not constrained by capacity or duration of attention limitations.

1.2.2 Machine Learning

Basically, learning process in organisms, which was explained above, is mimicked for learning on machine. By duplicating this process, we expect that the machine is able to perform a complicated job as we do through the learning process, such as pattern recognition, decision making, reasoning, etc. Until now these tasks represent open challenges to scientists even though many efforts have been proposed to reach closer to the goal. For example, after the human chess champion Garry Kasparov won in 1996 against an IBM computer called IBM Deep Blue, on May 11, 1997, the upgraded version of Deep Blue, involving reasoning and other techniques, beat the world chess champion after six-game match: two wins for IBM, one for the champion and three draws [117]. Another task showing that machine learning still falls behind human ability is on pattern recognition. For example, the study in [167] attempts to recognize a pattern (e.g., face, cat, etc) by employing a cluster containing 1000 machines (16,000 cores) for three days in the training step.

Before proceeding further, it would be better to discuss what machine learning is. It can be defined as the field of scientific study focusing on algorithms that are able to learn from data [160]. According to the definition, it can be seen that since we are working on machines, there are several components on learning that are different from organisms. Firstly, on machines instead of working with mind, we use algorithms as the main component on learning. Furthermore, they are not aimed as fixed procedures, but designed to extract knowledge using induction from data automatically. Another important aspect that needs to be prepared is data. In other words, machine

learning only considers a specific input dataset that is already prepared as parameters for dealing with an associated task.

Another well-known definition of learning proposed by Tom M. Mitchell is that "A computer program is said to learn from experience $\boldsymbol{E}$ with respect to some class of tasks $\boldsymbol{T}$ and performance measure $\boldsymbol{P}$, if its performance at tasks in $\boldsymbol{T}$, as measured by $\boldsymbol{P}$, improves with experience $\boldsymbol{E}$." [195] It means that learning involves three features related to each other: the class of tasks, the measure of performance, and the source of experience. For example, on fingerprint recognition we have the following components:

- Task $\boldsymbol{T}$: recognizing and classifying fingerprint
- Performance measure $\boldsymbol{P}$: percent of fingerprint correctly classified.
- Training experience $\boldsymbol{E}$: a database of fingerprint with given classifications.

It should be noted that algorithms will repeatedly improve performance by considering training experience $\boldsymbol{E}$ on the task $\boldsymbol{T}$ until the desired performance is reached.

Furthermore, in order to accomplish given tasks, a comprehensive strategy needs to be developed so that problem solving can be achieved effectively. Generally, there are two big steps on machine learning: training and prediction steps. Sometimes these processes can be conducted at the same time on one module, but we often recognize them as two separated stages.

The training step aims to extract knowledge by conducting learning from data. Therefore, it is also called the learning stage. Figure 1.3 shows components involved in the learning step adopted from [2]. It can be seen that the training dataset ($\boldsymbol{D}$) contains a vector of the input variable $\mathbf{x}$ and y_i representing the output variable[1]. The input-output examples in the training data define an unknown function f mapping the input space $\boldsymbol{X}$ to the output space $\boldsymbol{Y}$, so that $y_i = f(\mathbf{x}_i)$. The next component involved is learning methods. These algorithms mainly search the suitable hypothesis function g, which is a good guess of the unknown target function f, that is obtained from a set of hypotheses $\boldsymbol{H}$. According to the explanation, we can say that learning is actually a process to approximate the function f based on the training data. Moreover, because available training data are limited and searching all members of the hypothesis set is impossible, learning can be

[1] Noted that we consider supervised learning in this case

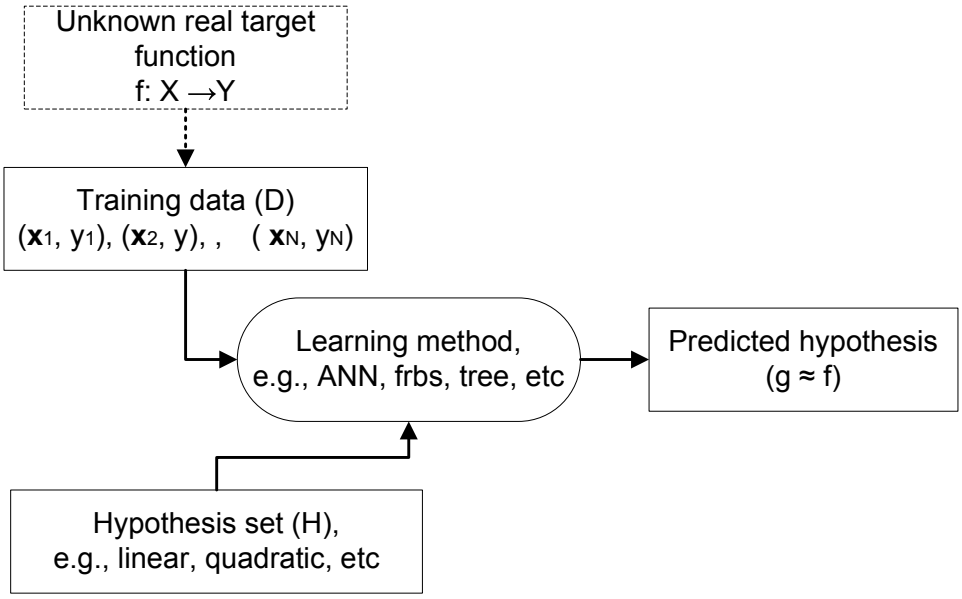


Figure 1.3: Basic components on the learning step [2].

seen as generalization as well. In addition to the predicted function g, we can define it as a model/knowledge represented by other formats, such as based on rules, cluster center, instances, etc.

After obtaining the model from the previous step, we perform the prediction stage on new data to get predicted values. Figure 1.4 shows that the testing data T are predicted by a prediction engine with considering the obtained model. Usually we perform the prediction step for the two following objectives: fitting model and prediction. The first one is aimed to validate and improve our model by using a part of the training data as testing, whereas on the later we use new data whose output values are unknown.

By looking at the two basic processes explained above, machine learning is effectively used and it will produce a sophisticated result when the following criteria can be fulfilled:

- Representative data are available. It is an important aspect required when working with machine learning. Without representative data, it will be impossible to produce a good model. Even though in practical situations, it is also difficult to obtain perfect data as we wish. Sometimes available data are quite limited, but other cases data overflow

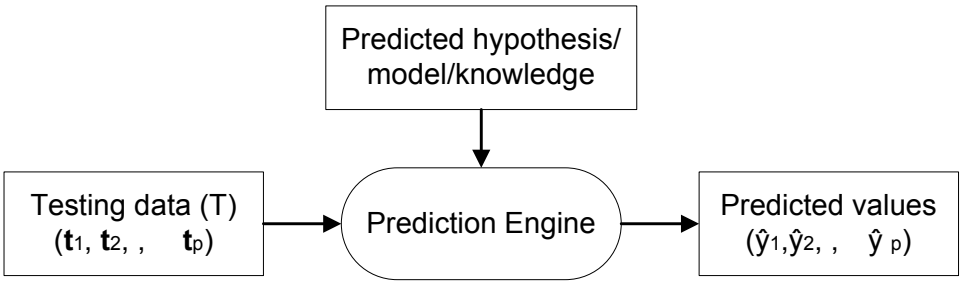


Figure 1.4: Basic components on the prediction step.

so that we are confused what we are looking for.

- Patterns exist on training data. Since basically a learning algorithm discovers and exploits patterns within training data, the existence of them is a must. It is represented by a function mapping input variables to the output or by other representations. Therefore, it is reasonable that machine learning does not produce a good model when data are generated in a random way.

- A mathematical formula is difficult to be constructed. Machine learning attempts to obtain a relationship on the input-output examples though from mathematical perspective the functions is difficult and complicated to be found. In contrast, if we have data that are already known their hypothesis function, performing machine learning becomes wasting the time. Of course, it still produces a result even though not so accurate as calculation using the mathematical equation.

1.2.3 Application Areas

This section briefly discusses applications of machine learning. Some examples of its implementations are provided on each type.

Supervised Learning

Definition. *The supervised learning task consists of constructing a model that maps input values to the output ones on available training data.*

It is clear here that we need to supply training data containing input and output values. Moreover, the observations are often known as instances/examples, the explanatory variables are termed features or input ones (usually grouped into a feature vector), and the output is called the response variable.

There are two types of problems that can be solved through supervised learning:

- Classification: In this task the available training dataset contains a categorical output variable. In other words, the task is to learn a mapping from inputs x to outputs y, where $y \in \{1, \ldots, C\}$, with C being the number of classes. For instance, let us consider the human face recognition problem. In this case the training dataset is a collection of images of human faces and their respective ids. Other cases can be found in the literature e.g., credit risk domain [59], fingerprint matching [106], the classification of cloud in satellite imagery [146], and forecast the recruitment of seven fish species of North East Atlantic [76].

- Regression: It is just like classification except the output variable takes real values. Here are some examples of real-world regression problems: a local forecasting of daily global horizontal irradiation (GHI) [16], prediction oil price [85], and financial time series data, e.g., IBM stock daily close price data [41].

Unsupervised Learning

Definition. *The unsupervised learning task consists of constructing a model from training data that do not contain output values.*

A model is built by deducing structures present or finding patterns in the input variables. Techniques in unsupervised learning are association rule identification and clustering. The former leads to the discovery of associations and correlations among available variables. For instance, in analysis of customer buying habits one can discover that customers who buy milk, will likely buy bread on the same trip to the supermarket as well. Clustering is a process of grouping a set of objects so that members of a cluster are more like each other than they are like members of a different cluster. The following are some real cases of clustering: the design and implementation of crime detection and criminal identification for Indian cities [274], early

detection of lung cancer risk [7], and analysis of cellular phenotypes in large imaging data sets [311].

Semi-supervised Learning

Definition. *The semi-supervised learning task consists of constructing a model from training data where a small set of input-output examples are available and the remainder only includes inputs.*

It can be seen that basically it is a combination between the supervised and unsupervised learning. In this case, the data set $\mathbf{x} = (x_i)_{i\in[n]}$ can be divided into two parts: the examples $\mathbf{x}_l = (x_1,\ldots,x_l)$, for which labels $\mathbf{y}_l = (y_1,\ldots,y_l)$ are provided, and the examples $\mathbf{x}_u = (x_{l+1},\ldots,x_{l+u})$, the labels of which are not known [44]. The goal of the task is to define predicted values (e.g., labels/classes) for the unlabeled examples. Applications of the semi-supervised learning are as follows: text classification with universum algorithm [173], face recognition [88], and improvement the quality of magnetic resonance brain tissue segmentation [224].

Reinforcement Learning

Definition. *The reinforcement learning task consists of constructing a model by providing stimulus on an environment to which the model must respond and react.*

In this case, the training data do not contain the target output, but instead contains some possible output along with a measure of how good that output is. In contrast to supervised learning where the training examples were of the form (*input*, *correct output*), the examples in reinforcement learning are of the form: (*input*, *some output*, *grade for this output*) [2]. We can find implementations of reinforcement learning as follows: e.g., the card game [19], the RoboCup simulated soccer [270], and the area of transportation and traffic engineering and specifically in intelligent transport systems [1].

1.3 R Language and Ecosystem

1.3.1 An Introduction to R

R is a programming language and software environment that provides a powerful interactive environment for scientific computing, data analysis, visualization, modeling, machine learning, high performance computing, statistics, etc [124]. It was developed by R. Ihaka and R. Gentleman and released under the Free Software Foundation's General Public License (GPL) license. It can be run in different hardware platforms and with different operating systems. Currently it can be installed on Linux, Mac OS X, Solaris, and MS Windows. Now, it is currently maintained and developed by the R Development Core Team. Furthermore, regarding a survey conducted by KDnuggets [152], R takes the first place for the programming language used for an analytics/data mining/data science in 2015 as shown in Figure 1.5.

From the programming language perspective, the R language has some characteristics and offers advantages, as follows [296]:

- It can be used in iterative and batch modes. For example, the integrated development environment (IDE) RStudio provides a sophisticated editor for developing R in the interactive way. In the batch mode, we can run R with input from *infile* and send output (*stdout/stderr*) to another file by using the command line environment.

- It provides complete data structures to ease data processing. For example, *list* is an R object that allows to include heterogeneous data types (e.g., *matrix*, *vector*, *list*, etc). For representing data in the table format, there exists two types: *matrix* and *data.frame*, where the first is used to collect objects having the same structure, whereas we can include different types in *data.frame*. Furthermore, R provides many strategies for manipulating data, e.g., subsetting, merging, and concatenating.

- It supports procedural, functional, and object-oriented programming languages. Especially for functional programming, R provides the following primitive functions: e.g., *Map()*, *Reduce()*, and the **apply()* family. Moreover, there are three techniques for working on the object-oriented programming, which are *S3*, *S4*, and reference classes.

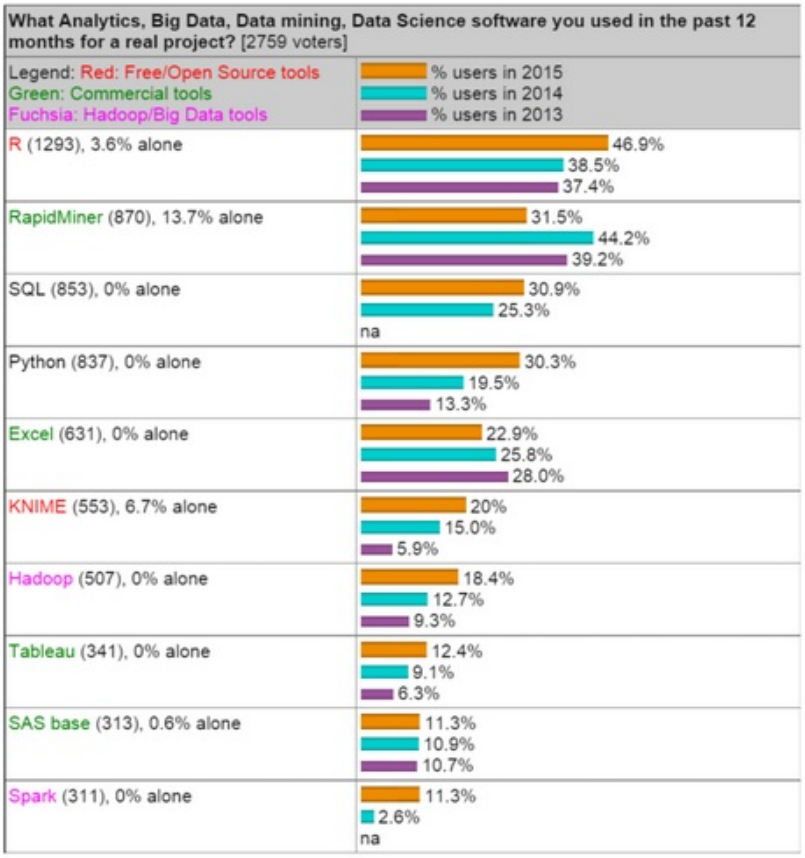


Figure 1.5: A survey of predictive analytics/data mining/data science software conducted by [152] in 2015.

- The performance of modules can be improved by using codes written in the C, C++, and Fortran languages.
- R provides complete and mature primitive functions and their documentations. The language is focused on statistical computing and graphics. For example, R provides *binom()*, *anova()*, and *summary()* that are used for calculating binomial distribution, ANOVA, and statistical summary (i.e., minimum, first quartile, median, etc), respectively.
- R has been growing exponentially. Recently, there are over 6000 packages for dealing with various tasks. These packages are built by many

active contributors from academics and industries.

- It is an open source and free language and runs on multiple platforms (i.e., Linux, Mac OS X, Solaris, and MS Windows). Because of these, it has been attracting many people to create a lot of community on Internet. For example, in order to get help from experts, we can go to the Rhelp mailing list (`https://stat.ethz.ch/mailman/listinfo/r-help`), stackoverflow (`http://stackoverflow.com/questions/tagged/r`), subject-specific mailing lists like R-SIG-mixed-models (`https://stat.ethz.ch/mailman/listinfo/r-sig-mixed-models`) and ggplot2 (`https://groups.google.com/forum/#!forum/ggplot2`).

Besides the benefits explained above, R offers a suitable ecosystem for scientists, engineers, and companies to contribute by developing packages, joining in forum discussions, etc. That is why R is constantly growing and now provides a wide variety of statistical and graphical techniques, as well as modules in many other areas, such as data mining, machine learning, pattern recognition, bioinformatics, and other fields.

1.3.2 The Comprehensive R Archive Network

Mostly, packages developed in the R framework are included in the following repositories: CRAN and the Bioconductor project. CRAN, which can be found at `http://cran.r-project.org/`, is maintained by the efforts of volunteers (the "CRAN team") and the resources of the R Foundation and the employers of those volunteers (WU Wien, TU Dortmund, U Oxford, AT&T Research). Then, Bioconductor (`http://www.bioconductor.org/`) is an open source, open development software project to provide R tools for the analysis and comprehension of high-throughput genomic data. The Bioconductor project started in 2001 and is overseen by a core team, based primarily at the Fred Hutchinson Cancer Research Center, and by other members coming from US and international institutions.

Now, there are over 6000 packages available in CRAN which are classified into more than 30 task views. For instance, the task view of "Machine Learning and Statistical Learning" contains more than 30 packages related to learning methods, e.g., neural networks, recursive partitioning, RFs, support vector machines, kernel methods, etc. Other task views that provides many useful packages for engineering and industries are time series analysis, high-performance and parallel computing, analysis of ecological and envi-

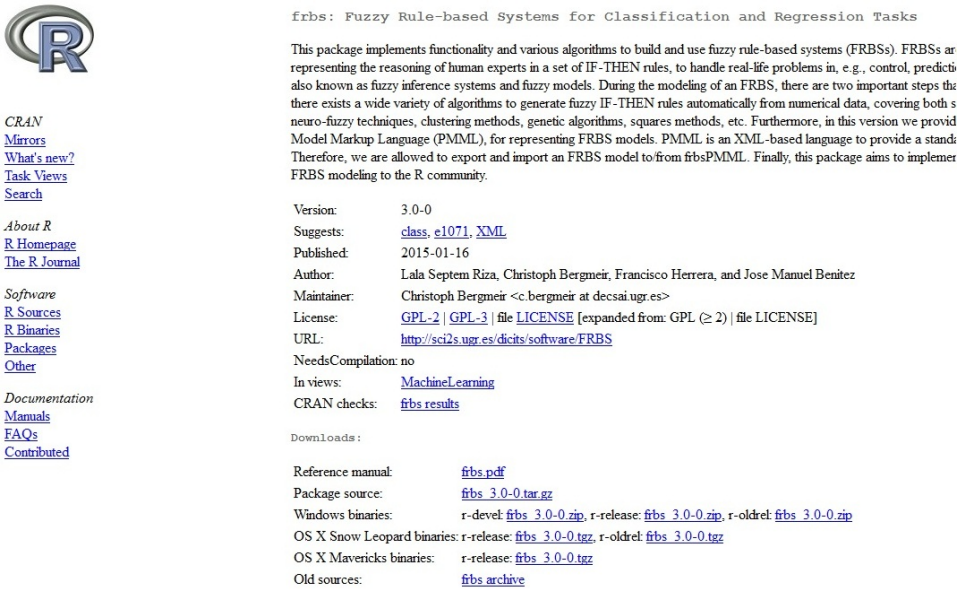


Figure 1.6: The display of the "frbs" package on CRAN.

ronmental data, and graphics displays and dynamic graphics and graphic devices and visualization. Furthermore, CRAN has a standard display of R packages on its website. For example, Figure 1.6 shows the display of the "frbs" package in CRAN. It can be seen that there are several parts on the CRAN page. Firstly, a title of the package is presented on the top; In this case, it is "frbs: FRBSs for Classification and Regression Tasks." Furthermore, we can also find: description, version, suggested packages, authors, maintainer, license, manual in the pdf format, and package source. From the user perspective R packages included in CRAN are easy to be installed in users' R environment, which is by calling the function *install.packages()*. CRAN is replicated through a network of mirrors around the globe.

The quality of many packages is backed through the following highly reputed academic journals: Journal of Statistical Software (`http://www.jstatsoft.org/`), the R Journal (`http://journal.r-project.org/`), and Bioinformatics (`http://bioinformatics.oxfordjournals.org/`). These journals have been publishing many articles related to R packages contributed in the repositories.

1.3.3 Development of R Packages

A key factor for the good quality of packages in R is that every package submitted into the repositories is checked both in automatic and manual fashions and must meet a standard quality. This section explains a work flow that should be taken into account by developers, who want their package to be included in CRAN.

An R package is a collection of functions that attempts to deal with problems based on a particular method or concept or set of them. One package usually involves many algorithms and techniques. The structure of a package is simply a directory that has the same name as the package and the following contents [234, 235, 236]:

- A file named *DESCRIPTION* with descriptions of the package, author, and license terms in a text format that is readable by computers and by people. Every package has to provide this file.
- A *man/* directory that contains documentation files of each functions included. These files are in the *.Rd* format and can be read by typing the *help()* function.
- An *R/* directory that comprises R code. So, it is obvious that this directory is important since all functions are stored in it as files with the extension *.R*.
- A *data/* directory storing datasets embedded in the package.
- A *src/* directory that stores the C, C++, and Fortran code. If we do not have the embedded code, this directory is not needed.

Furthermore, a good R package needs to present a complete and comprehensive manual. In general speaking, the manual contains several components as follows:

- Global descriptions about the packages on the first page such as *Maintainer*, *License*, *Title*, *Author*, *Description*, *Version*, etc.
- Short explanation corresponding to a function. Usually it contains *Description*, *Detail*, *Usage*, *Arguments*, *Value*, *References*, and *Examples*. The *Usage*, *Arguments*, and *Value* parts are used to explain a signature, input parameters, and output values of the associated function. In case we implement a particular algorithm proposed in articles,

references can be put on *Reference*. Lastly, the most important part is *Examples* containing executable unit tests. It is recommended to provide it for each function. Because every time we make a change on the code, this part will be executed to be unit testing on the building phase.

- Other information regarding an introduction to algorithms and concepts.

Currently the manual can be generated automatically by using the "Roxygen2" and "devtools" packages.

Processes of creating the R package can be illustrated as in Figure 1.7. First, a directory with the same name as the package name is created. It can be done by two ways, which are executing *package.skeleton()* and creating manually by users. After that, we need to create and write the *DESCRIPTION* file. All functions involved in the package with their documentations are saved in the *R/* sub folder with the extension *.R*. After writing all parts of the documentation, we generate *.Rd* files by using "Roxygen2" or "devtools". Checking, building, and testing need to be performed to ensure the package as intended perfectly. On this stage, we may do debugging and refining functions repeatedly. The final step before submission to CRAN, final checking by executing `R CMD check -as-cran` should be done. Actually, this step aims to check whether the package has met with a standard quality required by CRAN.

As we mentioned above that one remark showing how packages in R are kept in good quality is that every package submitted into the repository is checked manually and must meet a standard quality. The following are some items that will be checked by `R CMD check -as-cran`:

- checking for the file *DESCRIPTION*,
- checking package subdirectories
- checking whether the package can be installed,
- checking R files for syntax errors,
- checking R code for possible problems,
- checking Rd files,
- checking for missing documentation entries,

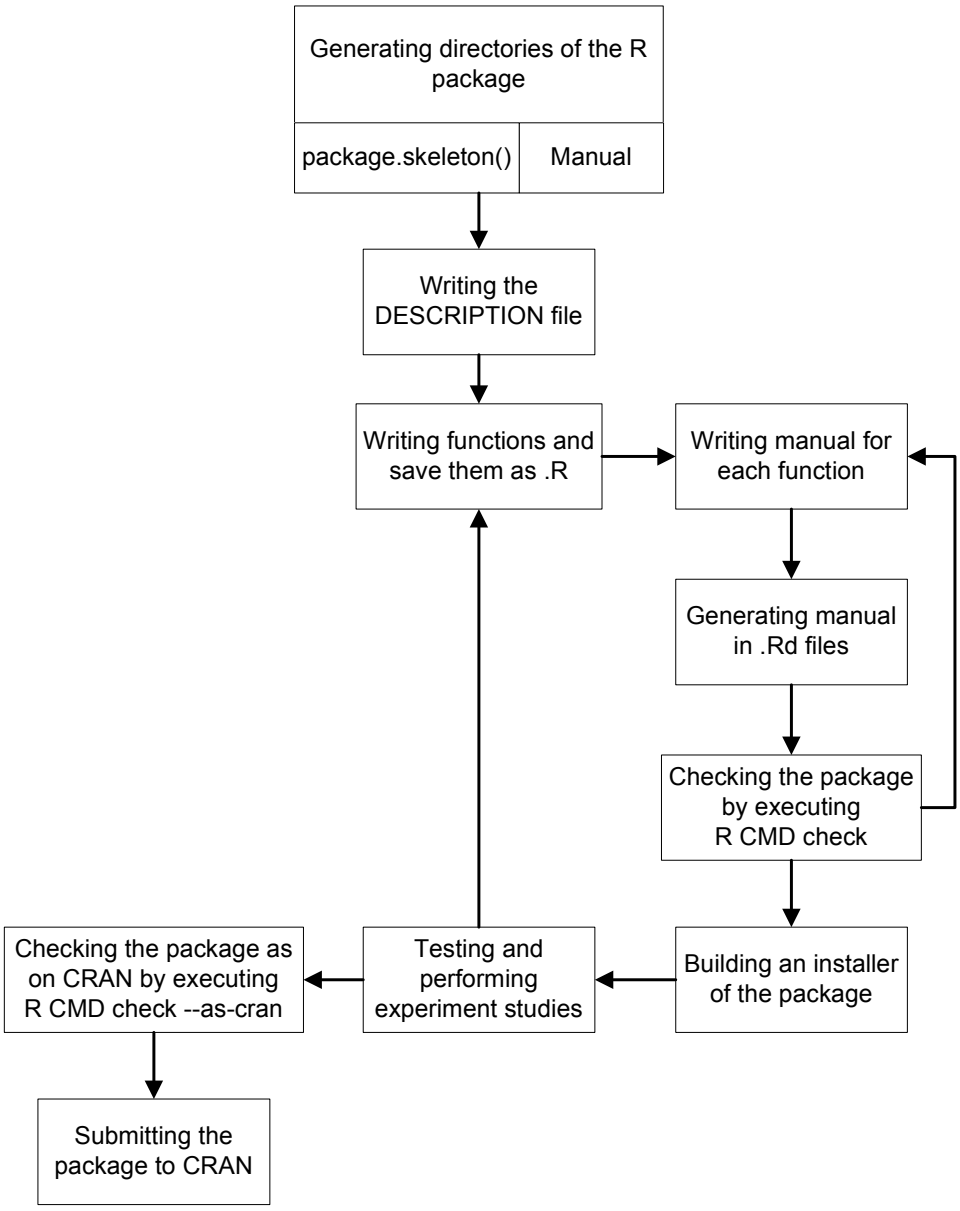


Figure 1.7: The processes of an R package submission.

- checking for code/documentation mismatches,
- checking examples,

- checking PDF version of manual.

CRAN requires us to obtain the values *OK*, which means there is no a part on the above items that gets *warnings* and *NOTE*. For specific problems, CRAN's maintainers can be also check a new R package manually. Therefore, to fulfill all requirements is mandatory.

1.4 Predictive Model Markup Language

1.4.1 Introduction

Due to the complexity of problems faced today, researchers and practitioners deal with them by proposing and using a wide variety of methods implemented in various software libraries. This phenomenon leads us to a situation where so many kinds of software are available for use. Next, when using various packages having different specification of input and output data, another problem arises, which is interoperability. As interoperability is an important issue not only in industry cases but also for academic purposes, this is a shortcoming that we address with our work.

For academic purposes, when proposing a new algorithm, it is important to perform an experimental study and then provide a comparison with other related approaches, to analyze the behaviour and performance of the new technique. One of the critical issues regarding this process is that it typically requires to understand and analyze different formats of models produced by various software libraries. Naturally, it is difficult to make a comprehensive comparison, e.g., according to the interpretability perspective. And even further processing steps involving the models, such as assembling and aggregation, are almost impossible. Therefore, we see that a universal representation framework is urgent to be designed and implemented, especially for the academic research community. Another advantage of the universal representation is that it promotes reproducible research [221] as research results can be archived, distributed, replicated, and reproduced easily in a standard format. In other words, multiple research groups using different platforms can share and analyze models.

In industry, interoperability is often very important and required, as users dedicated to model construction may often be located in another department as the users of the models, also using different computer programs in their

workflows. For example, an insurance company may have a department to generate models of a risk level. Then, there may be another department that is in charge of applying the model for prediction of the risk level of somebody according to given profiles. Therefore, in this case the obtained models would be distributed to many places. Furthermore, it is desirable for the resulting models to be easily understood and communicated. Again, from an industry perspective, a universal representation framework that satisfies these requirements is desirable.

The Institute of Electrical and Electronics Engineers (IEEE) defines interoperability as follows [125]: "The ability of two or more systems or components to exchange information and to use the information that has been exchanged." In other words, interoperability attempts to minimize any role of human to intervene of the models e.g., re-write, re-format, and transform. Thus, the efficiency can be achieved by researchers and practitioners. However, a challenge that should be addressed is that how we establish a common representation of models. The universal representation should be independent of programming languages and environment/platforms. Additionally, it has to have solid definitions and constraints in representing models so that ambiguities can be avoided.

PMML is an Extensible Markup Language (XML)- based language to provide a standard for describing models produced by data mining and machine learning algorithms [103]. It is developed by Data mining Group which is an independent, vendor led committee including companies such as IBM, SAS, Zementis, and Microsoft.

One main reason why PMML is specified in XML is that XML is a standard language defined in the XML 1.0 specification by the World Wide Web Consortium [299]. It provides a format that is both human- and machine-readable. There are many applications that can generate their formats into XML, such as Microsoft Office and LibreOffice. Additionally, to write a document based on XML, we need to consider definitions determined by a given schema, e.g., based on XML Schema [289]. It contains a basic grammar explaining the structure, content, and constraint of documents. Moreover, any new extensions made in the document have to be defined on XML schema. Therefore, it is reasonable that XML is used to be a standard representation framework.

Figure 1.8 shows a PMML workflow, together with some advantages of PMML in data analysis processes. The workflow generally involves a modelling, expert intervention, and deploying phase. In the modelling phase,

the final result is a model produced by learning methods according to given data. It may also involve data pre-processing and model validation. After the modelling, the model is exported to the PMML format, which is XML-based and human-readable. Even though interpretability of the model mainly depends on the type of learning methods used, PMML helps at this end with readability and transparency. Therefore, human experts can be relatively easy to read, understand, and even modify the model and adapt it better to real-world conditions. Anyway, since it is a text based representation it can be viewed and edited with just a text editor; no specific complex tool is required. In the final phase, we can also see several advantages. The model can be used in various predictor engines compliant with the PMML format to predict new data. In other words, with PMML it is easy to move the obtained models between various applications and platforms, so that it is easy to share them, e.g., across different departments. In addition, we note that prediction with new data in this phase is usually performed and repeated more frequently than the modelling in the first phase. Here, PMML helps to achieve a reproducible concept [221] since PMML provides a standard format that can be used anytime to predict new data by any compliant application.

A main contribution of PMML is to provide interoperable schemata of predictive models. Using PMML, we can easily perform these tasks as our models are documented in an XML-based language. Human experts can also update and modify the model on the files directly. Furthermore, the study in [102] shows that PMML has been deployed in cloud computing [40, 39]. Therefore, we can apply our models anywhere without worrying about details of applications and resources. Another benefit offered by PMML is that it can effectively express many models without depending on any programming languages (e.g., Java, Python, and C++) or platforms (e.g., Windows, Linux, and Mac).

1.4.2 Specifications

In PMML, currently, there are available 16 models documented as follows:

- Association rules: representing rules showing relations between attributes;
- Baseline models: specifying change detection models;

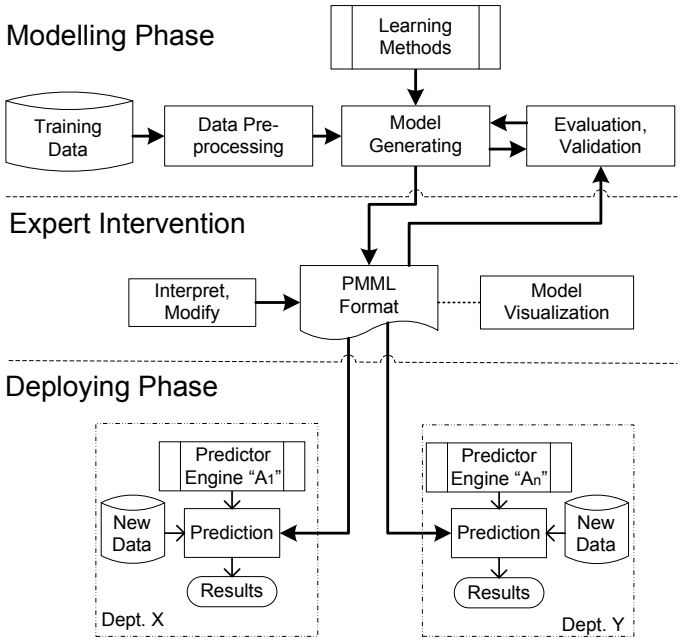


Figure 1.8: Workflow using PMML.

- Cluster models: representing a set of clusters;
- General regression: allowing a multitude of regression models;
- k-nearest neighbors: representing a model of instance-based learning algorithms;
- Naïve Bayes: representing a model based on simple probabilistic classifiers according to Bayes' theorem;
- Neural networks: describing models based on artificial neural networks;
- Regression model: determining the relationship between dependent and independent attributes;
- Ruleset models: representing rules based on DT models;
- Scorecard models: describing a model mapping a set of inputs to predict a target value;
- Sequence rules: containing a set of rules for various items;

- Text models: providing a model used for text operations, such as frequency of terms;
- Time series models: providing time series analysis, such as forecasting;
- Tree models: providing a model represented by a tree for classification;
- Support vector machine: representing a model based on the method for classification and regression.

Additionally, PMML also provides schemata for data pre- and post-processing. For example, PMML defines normalization, discretization, value mapping, aggregation, etc.

Since PMML is an XML-based language, the specification is defined by an XML Schema as recommended by the World Wide Web Consortium (W3C) [289]. The general schema and components of PMML can be seen in Listing 1.1. The PMML format is specified by the main tag *PMML* that contains some components. In the following, we describe the main components:

- *Header*: It contains general information about the PMML document, such as copyright information for the model, its description, application, and timestamp of generation.
- *DataDictionary*: It contains information related to fields or variables, such as number, names, types, and value ranges of variables.
- *MODEL-ELEMENT*: It is a main part of the PMML document that consists of models supported by PMML. In each model, there are several components embedded in the element, such as *MiningSchema* and *Output*. *MiningSchema* specifies outlier treatment, a missing value replacement policy, and missing value treatment, whereas *Output* shows a description of the output variable. For example, in a clustering model, we define a schema representing the cluster centers that are included in the *ClusteringModel* element.

Besides these components, there are some optional elements, such as *MiningBuildTask*, *TransformationDictionary*, and *Extension*.

```
<xs:element name="PMML">
  <xs:complexType>
    <xs:sequence>
```

```
      <xs:element ref="Header"/>
      <xs:element ref="MiningBuildTask" minOccurs="0"/>
      <xs:element ref="DataDictionary"/>
      <xs:element ref="TransformationDictionary" minOccurs="0"/>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="MODEL-ELEMENT"/>
      </xs:sequence>
      <xs:element ref="Extension" minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="version" type="xs:string"
        use="required"/>
  </xs:complexType>
</xs:element>

<xs:group name="MODEL-ELEMENT">
  <xs:choice>
    <xs:element ref="AssociationModel"/>
    ...
    <xs:element ref="TreeModel"/>
  </xs:choice>
</xs:group>
```

Listing 1.1: General XML Schema of PMML.

According to its functionalities, applications can be classified into two groups:

- PMML producer: It refers to a software that produces models, and exports/writes them to PMML format.
- PMML consumer: It refers to a software used for importing/reading and deploying PMML models to predict new data. In this software, there are procedures for validating and verifying the PMML format.

Nowadays, the PMML framework is implemented in several platforms. In the R environment, we can find the PMML-producer application "pmml" [103]. In order to generate models, the package executes several other packages available in R, such as "arules" for mining association rules, and "nnet"

for neural networks. Another package available within R, "pmmlTransformations", is an extension of "pmml" for supporting the data pre-processing in PMML such as normalization, discretization, and value mapping [135]. Next, the Konstanz Information Miner (KNIME), which is a platform for data integration, processing, analysis, and exploration [28], can be used both as a PMML producer and consumer [197]. SPSS provides a feature to import and export from/to PMML format [101]. The Waikato Environment for Knowledge Analysis (WEKA) allows to import PMML models based on regression, general regression, artificial neural networks, tree models, rule set models, and support vector machine models [108]. In order to provide further interoperability in delivering software solutions, PMML has been deployed in cloud computing using the Software-as-a-Service model [102]. For example, it is embedded in the ADAPA scoring engine on the Amazon Web Services. A detailed table showing all software systems that implement the PMML standard can be found at http://www.dmg.org/products.html.

1.5 Fuzzy Systems

1.5.1 Fuzzy Sets

When we were in school, a lot of lessons aimed at teaching us to think in accordance with the rules precisely. In the arithmetic lesson, for example, if we count the numbers by following the rules that have been learned, then we will definitely get results that are 100 percent correct and precise. Every time we repeat the calculations, we always get the same result. However, as people say: "the world is full of uncertainty and ambiguity" we easily find real-world problems that are not always possible to be solved with certainty. Let us take a very simple example to explain this fact. Suppose a basket containing 100 pieces of mango. Then, we asked 10 people to separate the fruits into two groups, fruits have been ripe and still raw, in different baskets. In the final results, we probably find that they choose different fruits in the baskets. However, we can understand the results because they may have different criteria for deciding which ones are ripe and raw, for example based on color, odor, hardness, dimensions, etc. Even on the same criteria, if we provide the scale 1 until 100, they may give different values on the same fruit. This example illustrates that the imprecision and uncertainty are very close and involved in our lives. Therefore, the procedure, which is tolerant for imprecision and uncertainty, is required to deal with our real-

world problems.

Fifty years ago, Zadeh published a seminal paper that proposes an extension of the classical set theory, Fuzzy Set Theory [306], for modeling and representing human knowledge. The concept introduces a degree of membership which means that instead of just having two qualifications: member or non-member, an object can be classified into a certain category with a degree between zero and one. A degree of zero and one refers to an object is not a member and a member of set whereas a value somewhere in between shows a partial degree of membership. For instance, instead of saying the age in term of the exact number, e.g., 32 year old, we can just say in the following terms: "child," "young," "adult," "mature," "old." It can be seen that the latter provide vague and less specific information, but it is more useful in more cases. Then, by considering fuzzy sets, we can define that 30 year old refers to "young" with the grade 0.2 and "mature" with the degree 0.8.

Mainly, the degree of membership of a given object is defined by the so-called membership functions. If the function of a fuzzy set A is denoted by μ_A, it is mapping original values to the unit interval $[0,1]$ as a grade: $\mu_A : X \rightarrow [0,1]$. Therefore, fuzzy sets can be viewed as a generalization of the classical sets that only involve two values: 0 and 1. Furthermore, employing the degrees of membership makes fuzzy set theory effectively works on natural languages containing vagueness and imprecision [307].

A degree of membership is commonly generated by the following functions: triangular, trapezoid, Gaussian, sigmoid, and generalized bell. They can be defined as follows:

- Triangular: It has the shape of a triangle and is described by the corner points (a,b,c). It can be written as

$$\mu_{T(a,b,c)}(x) = \begin{cases} 0, & \text{where } x \leq a \text{ or } x > c \\ \frac{x-a}{b-a}, & \text{where } a < x \leq b \\ \frac{x-c}{b-c}, & \text{where } b < x \leq c \end{cases}$$

- Trapezoid: It refers to the trapezoid function, which is defined by the

corner points (a,b,c,d) as follows:

$$\mu_{Tr(a,b,c,d)}(x) = \begin{cases} 0, & \text{where } x \leq a \text{ or } x > c \\ \frac{x-a}{b-a} & \text{where } a < x \leq b \\ 1, & \text{where } b < x \leq c \\ \frac{x-d}{c-d}, & \text{where } c < x \leq d \end{cases}$$

- Gaussian: Bell shaped and characterised by two parameters: mean c and deviation σ, with

$$\mu_{G(c,\tau)}(x) = \exp\left(-\frac{(x-c)^2}{\sigma^2}\right)$$

- Generalized bell: It is defined by the parameters (a,b,c):

$$\mu_{Gb(a,b,c)}(x) = \frac{1}{1+\left(\left(\frac{x-c}{a}\right)^2\right)^b}$$

- Sigmoid: It is defined by two parameters, γ and c:

$$\mu_{S(\gamma,c)}(x) = \frac{1}{a+\exp(-\gamma(x-c))}$$

Let us consider the previous example, so we can define membership functions of age as Figure 1.9. In this case, we split "age" into five fuzzy terms: "child," "young," "adult," "mature," "old," by using the trapezoid functions between 0 and 80 year old.

Basically, Figure 1.9 can also be expressed by the following functions on

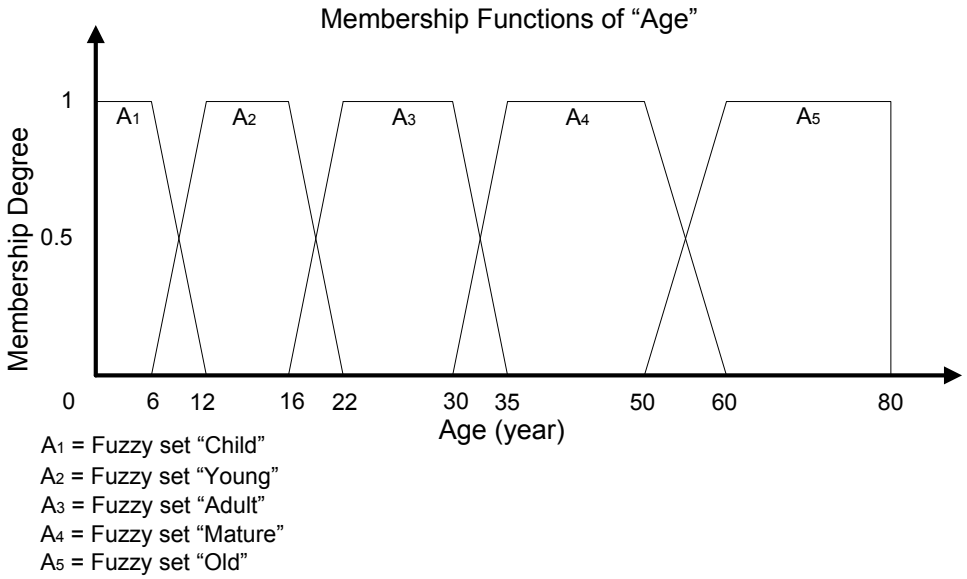


Figure 1.9: Membership functions of the “age” variable.

the interval $[0, 80]$:

$$\mu_{A_1} = \begin{cases} 1, & \text{where } x \le 6 \\ (12-x)/6, & \text{where } 6 < x < 12 \\ 0, & \text{where } x \ge 12 \end{cases}$$

$$\mu_{A_2} = \begin{cases} 0, & \text{where either } x \le 6 \text{ or } x \ge 22 \\ (x-6)/6, & \text{where } 6 < x < 12 \\ (22-x)/6, & \text{where } 16 < x < 22 \\ 1, & \text{where } 12 \le x \le 16 \end{cases}$$

$$\mu_{A_3} = \begin{cases} 0, & \text{where either } x \le 16 \text{ or } x \ge 35 \\ (x-16)/6, & \text{where } 16 < x < 22 \\ (35-x)/5, & \text{where } 30 < x < 35 \\ 1, & \text{where } 22 \le x \le 30 \end{cases}$$

$$\mu_{A_4} = \begin{cases} 0, & \text{where either } x \le 30 \text{ or } x \ge 60 \\ (x-30)/5, & \text{where } 30 < x < 35 \\ (60-x)/10, & \text{where } 50 < x < 60 \\ 1, & \text{where } 35 \le x \le 50 \end{cases}$$

$$\mu_{A_5} = \begin{cases} 0, & \text{where } x \le 50 \\ (x-50)/10, & \text{where } 50 < x < 60 \\ 1, & \text{where } x \ge 60 \end{cases}$$

So, it is clear that 32 year old refers to the term "adult" and the term "mature" with the same degree of 0.4 for both terms.

Furthermore, the three basic operations on crisp sets, i.e., the complement, intersection, and union, can be generalized to fuzzy sets in various fashions. The following are standard fuzzy set operations:

$$\begin{aligned} \bar{\mu}(x) &= 1 - \mu(x) \\ (\mu_{A_1} \cap \mu_{A_2})(x) &= \min[\mu_{A_1}(x), \mu_{A_2}(x)] \\ (\mu_{A_1} \cup \mu_{A_2})(x) &= \max[\mu_{A_1}(x), \mu_{A_2}(x)] \end{aligned} \tag{1.1}$$

A lot of monographs provide comprehensive explanations about fuzzy theory and its techniques, for example in [158, 220].

Several extensions based on the fuzzy set concepts have been proposed, such as FRBSs and FRSTs as presented below.

1.5.2 Fuzzy Rule-Based Systems

Fuzzy rules are an extension of classical rule-based systems (also known as production systems or expert systems). Basically, they are expressed in the form "IF A THEN B" where A and B are fuzzy sets. In other words, instead of involving numerical values, a fuzzy rule contains fuzzy sets. A and B are called the antecedent and consequent parts of the rule, respectively. For example, we have following variables and their linguistic values to classify a mango fruit quality:

dimension = {small, medium, large}

weight = {light, medium, heavy}

color intensity = {lighter, neutral, darker}

Based on a particular condition, we can define a fuzzy rule, as follows:

IF the dimension is medium and the weight is medium and
the color intensity is darker **THEN** the quality is good.

where the linguistic values of the output variable quality is "bad," "medium," and "good." It can be seen that the representations based on fuzzy sets are

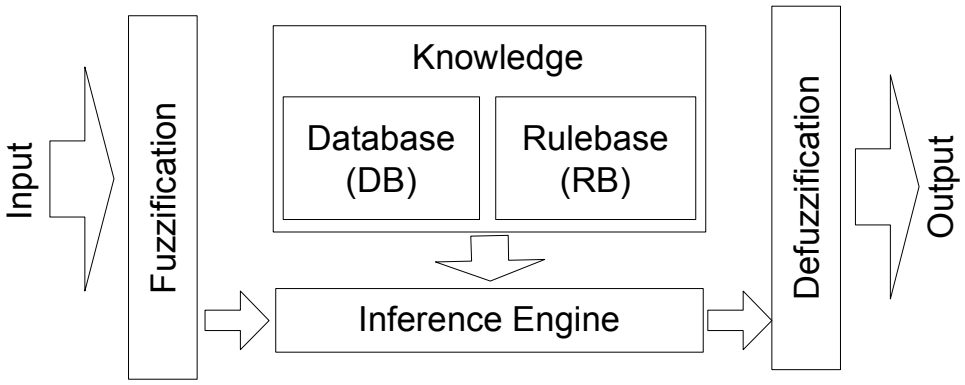


Figure 1.10: The components of the Mamdani model.

much easier to be interpreted by human than classical rules. Indeed, the linguistic values are more understandable than in numerical form.

Furthermore, FRBSs are methodologies to represent human expert knowledge in a set of fuzzy rules in a linguistic fashion. With respect to the structure of the rule, there exist two basic FRBS models: the Mamdani and Takagi Sugeno Kang (TSK) models. The differences and characteristics of both models are discussed in the following.

The Mamdani Model

This model type was introduced by [183, 184]. It is built by linguistic variables in both the antecedent and consequent parts of the rules. So, considering multi-input and single-output (MISO) systems, fuzzy IF-THEN rules are of the following form:

$$\textbf{IF } \boldsymbol{X}_1 \text{ is } \boldsymbol{A}_1 \text{ and } \ldots \text{ and } \boldsymbol{X}_n \text{ is } \boldsymbol{A}_n \textbf{ THEN } \boldsymbol{Y} \text{ is } \boldsymbol{B} \tag{1.2}$$

where $\boldsymbol{X}_i$ and $\boldsymbol{Y}$ are input and output linguistic variables, respectively, and $\boldsymbol{A}_i$ and $\boldsymbol{B}$ are linguistic values.

The standard architecture for the Mamdani model is displayed in Figure 1.10. It consists of four components: *fuzzification*, *knowledge base*, *inference engine*, and *defuzzifier*. The fuzzification interface transforms the crisp inputs into linguistic values. The knowledge base is composed of a database and a rule base. While the database includes the fuzzy set definitions and

parameters of the membership functions, the rule base contains the collections of fuzzy IF-THEN rules. The inference engine performs the reasoning operations on the appropriate fuzzy rules and input data. The defuzzifier produces crisp values from the linguistic values as the final results.

Since the Mamdani model is built out of linguistic variables it is usually called linguistic or descriptive system. A key advantage is that its interpretability and flexibility to formulate knowledge are higher than for other FRBSs. However, the model suffers some drawbacks. For example, its accuracy is lower for some complex problems, which is due to the structure of its linguistic rules [52].

The TSK Model

Instead of working with linguistic variables on the consequent part as in the Mamdani model in Equation 3.1, the TSK model [273, 271] uses rules whose consequent parts are represented by a function of input variables. The most commonly used function is a linear combination of the input variables: $Y = f(X_1, \ldots, X_n)$ where X_i and Y are the input and output variables, respectively. The function $f(X_1, \ldots, X_n)$ is usually a polynomial in the input variables, so that we can express it as $Y = p_1 \cdot X_1 + \cdots + p_n \cdot X_n + p_0$ with a vector of real parameters $p = (p_0, p_1, \ldots, p_n)$. Since we have a function on the consequent part, the final output is a real value, so that there is no defuzzifier for the TSK model.

The TSK model has been successfully applied to a large variety of problems, particularly, when accuracy is a priority. Its success is due to the fact that this model type provides a set of system equations on the consequent parts whose parameters are easy to estimate by classical optimization methods. Their main drawback, however, is that the obtained rules are not so easy to interpret.

Variants of FRBSs

Other variants have been proposed in order to improve the accuracy and to handle specific problems. Their drawback is that they usually have higher complexity and are less interpretable. For example, the disjunctive normal form fuzzy rule type has been used in [92]. It improves the Mamdani model in Equation 3.1 on the antecedent part, in the sense that the objects are

allowed to consider more than one linguistic value at a time. These linguistic values are joined by a disjunctive operator. The approximate Mamdani type proposed by [116] may have a different set of linguistic values for each rule instead of sharing a common definition of linguistic values as it is the case of the original Mamdani formulation. So they are usually depicted by providing the values of the corresponding membership function parameters instead of a linguistic label. The advantages of this type are the augmented degree of freedom of parameters so that for a given number of rules the system can better be adapted to the complexity of the problems. Additionally, the learning processes can identify the structure and estimate the parameters of the model at the same time.

Fuzzy rule-based classification systems (FRBCSs) are specialized FRBSs to handle classification tasks. A main characteristic of classification is that the outputs are categorical data. Therefore, in this model type we preserve the antecedent part of linguistic variables, and change the consequent part to be a class C_j from a prespecified class set $C = \{C_1, \ldots, C_M\}$. Three structures of fuzzy rules for classification tasks can be defined as follows. The simplest form introduced by [49] is constructed with a class in the consequent part. The FRBCS model with a certainty degree (called weight) in the consequent part was discussed in [129]. FRBCS with a certainty degree for all classes in the consequent part are proposed by [185]. It means that instead of considering one class, this model provides prespecified classes with their respective weights for each rule.

Constructing FRBSs

Constructing an FRBS means defining all of its components, especially the database and rule base of the knowledge base. The operator set for the inference engine is selected based on the application or kind of model. For example, minimum or product are common choices for the conjunction operator. But the part that requires the highest effort is the knowledge base. Basically, there are two different strategies to build FRBSs, depending on the information available [291]. The first strategy is to get information from human experts. It means that the knowledge of the FRBS is defined manually by knowledge engineers, who interview human experts to extract and represent their knowledge. However, there are many cases in which this approach is not feasible, e.g., experts are not available, there is not enough knowledge available, etc. The second strategy is to obtain FRBSs

by extracting knowledge from data by using learning methods.

Generally the learning process involves two steps: structure identification and parameter estimation [272, 219]. In the structure identification step, we determine a rule base corresponding to pairs of input and output variables, and optimize the structure and number of the rules. Then, the parameters of the membership function are optimized in the parameter estimation step. The processing steps can be performed sequentially or simultaneously.

Regarding the components of the FRBSs that need to be learned or optimized, the following has to be performed:

- Rule base: Qualified antecedent and consequent parts of the rules need to be obtained, the number of rules needs to be determined and the rules have to be tuned.

- Database: Optimized parameters of the membership functions have to be defined.

- Weight of rules: Especially for fuzzy rule-based classification systems, optimized weights of each rule have to be calculated.

Additionally, tuning can be also performed after obtaining the model.

After the inference engine operators are set and the knowledge base is built, the FRBS is ready. Obviously, as in other modeling or machine learning methods, a final validation step is required. After achieving a successful validation the FRBS is ready for use. Figure 1.11 shows the learning and prediction stages of an FRBS. An FRBS can be used just like other classification or regression models —e.g., classification trees, artificial neural networks, and Bayesian networks.

Researchers have recently implemented methods based on FRBSs to software libraries for both academic and industry purposes, e.g., "Xfuzzy" [21], FisPro [105], GUAJE [12], and KEEL [10]. These software libraries are important to be implemented, for a number of reasons. First, researchers outside the field of computer science can just use the tools for tackling their problems. In other words, researchers can focus on doing experiments to solve the problems. After that, by using open-source tools, we can reproduce and communicate our scientific results with others.

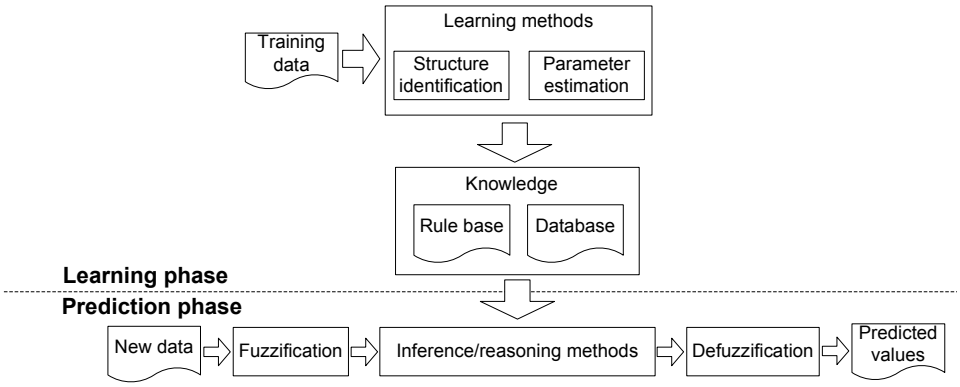


Figure 1.11: Learning and prediction phases of an FRBS.

Learning Methods in FRBSs

As mentioned above there are two ways to construct a model in FRBSs: by human experts and learning from data by using machine-learning methods. In this part, we explain the latter by presenting several well-known algorithms. We can classify them into five groups: FRBSs based on space partition, genetic algorithms, clustering, neural networks, and gradient descent. In the following, we discuss these five groups in detail.

FRBSs based on space partition approaches. Learning methods included in this group use a strategy of dividing the variable space, and then considering this partition to obtain the parameters of the membership functions. The following methods use space partition approaches to build FRBSs:

- Wang and Mendel's technique (WM). It was proposed in [292] using the Mamdani model.
- FRBCS using Chi's method (FRBCS.CHI). This method was proposed in [49], which is an extension of WM method, for tackling classification problems.
- FRBCS using Ishibuchi's method with weight factor (FRBCS.W). This method is adopted from [127]. It implements the second type of FRBCS which has certainty grades (weights) in the consequent parts of the rules.

FRBSs based on neural networks. The systems in this group are commonly also called neuro-fuzzy systems or fuzzy neural networks (FNN) [36] since they combine artificial neural networks (ANN) with FRBSs. An FRBS is laid upon the structure of an ANN and the learning algorithm of the latter is used to adapt the FRBS parameters, usually the membership function parameters. There exist many variants of methods based on FNN, such as the adaptive-network-based fuzzy inference system (ANFIS) and the hybrid neural fuzzy inference system (HYFIS).

- ANFIS: This method was proposed in [132]. It considers a TSK FRBS model which is built out of a five-layered network architecture. The ANFIS learning algorithm consists of two processes, the forward and the backward stage.
- HYFIS: This learning procedure was proposed in [155]. It uses the Mamdani model as its rule structure. There are two phases in this method for learning, namely the knowledge acquisition module and the structure and parameter learning.

FRBSs based on genetic algorithms. Genetic fuzzy systems (GFS) [52] are a combination of genetic algorithms and FRBSs. Generally, the genetic algorithms are used to search and optimize the parameters of the membership functions and of the fuzzy rule construction process. The following are methods that can be included in this group:

- GFS based on Thrift's method (GFS.THRIFT). [278] introduces a technique for learning of Mamdani models based on a genetic algorithm.
- GFS based on the MOGUL methodology (GFS.FR.MOGUL). This method is proposed in [116]. It uses a genetic algorithm to determine the structure of the fuzzy rules and the parameters of the membership functions simultaneously.
- Ishibuchi's method based on genetic cooperative competitive learning (GFS.GCCL). This method is based on [128] using genetic cooperative competitive learning to handle classification problems.
- Ishibuchi's method based on hybridization of GCCL and Pittsburgh (FH.GBML). This method is based on Ishibuchi's method using the hybridization of GCCL and Pittsburgh approach for GFSs [131].

- Structural learning algorithm on vague environment (SLAVE). This method is adopted from [91]. SLAVE is based on the IRL approach which means that we get only one fuzzy rule in each execution of the genetic algorithm.

FRBSs based on clustering approaches. Fuzzy rules can be constructed by clustering approaches through representing cluster centers as rules such as the following methods:

- Subtractive clustering (SBC). This method is proposed by [50]. For generating the rules in the learning phase, the SBC method is used to obtain the cluster centers. It is an extension of Yager and Filev's mountain method [305]. After getting all the cluster centers from SBC, the cluster centers are optimized by fuzzy c-means.

- Dynamic evolving neural fuzzy inference system (DENFIS). This method is proposed by [151]. There are several steps in this method that are to determine the cluster centers using the evolving clustering method, to partition the input space and to find optimal parameters on the consequent part of the TSK model, using a least squares estimator.

FRBSs based on the gradient descent approach. Some methods use a gradient descent approach to optimize the parameters on both antecedent and consequent parts of the rules. The following are methods included in this group:

- Fuzzy inference rules with descent method (FIR.DM). This method is proposed by [208]. FIR.DM uses simplified fuzzy reasoning where the consequent part is a real number (a particular case within the TSK model), while the membership function on the antecedent part is expressed by an isosceles triangle.

- FRBS using heuristics and the gradient descent method (FS.HGD). This method is proposed by [130]. It uses fuzzy rules with non-fuzzy singletons (i.e., real numbers) in the consequent parts.

1.6 Rough Set Theory and Fuzzy Rough Set Theory

1.6.1 Rough Set Theory

Basic Concepts

RST was introduced by Pawlak in 1982 [213] as a methodology for data analysis based on the approximation of concepts in information systems. It revolves around the notion of discernibility: the ability to distinguish between objects, based on their attribute values. Given an indiscernibility relation, we can construct lower and upper approximations of concepts. Objects included in the lower approximation can be classified with certainty as members of the concept. In contrast, the upper approximation contains objects possibly belonging to the concept. Additionally, for more than three decades RST has been attracting researchers and practitioners in many different areas.

In RST, a data set is represented as a table called an information system $\mathscr{A} = (U, A)$, where U is a non-empty set of finite objects known as the universe of discourse (note: it refers to all instances/rows in datasets) and A is a non-empty finite set of attributes, such that $a : U \rightarrow V_a$ for every $a \in A$. The set V_a is the set of values that attribute a may take. Information systems that involve a decision attribute, containing classes for each object, are called decision systems or decision tables. More formally, it is a pair $\mathscr{A} = (U, A \cup \{d\})$, where $d \notin A$ is the decision attribute. The elements of A are called conditional attributes. The information system representing all data in a particular system may contain redundant parts. It could happen because there are the same or indiscernible objects or some superfluous attributes. The indiscernibility relation is a binary relation showing the relationship between two objects. In the following, we describe an equivalence relation. Let $\mathscr{A} = (U, A)$ be an information system, then for any $B \subseteq A$ there is an equivalence relation $R_B(x, y)$:

$$R_B(x, y) = \{(x, y) \in U^2 | \forall a \in B, a(x) = a(y)\} \tag{1.3}$$

If $(x, y) \in R_B(x, y)$, then x and y are indiscernible by attributes from B. The equivalence classes of the B-indiscernibility relation are denoted $[x]_B$. The indiscernibility relation will be further used to define basic concepts of RST which are lower and upper approximations. Let $B \subseteq A$ and $X \subseteq$

U, X can be approximated using the information contained within B by constructing the B-lower and B-upper approximations of X:

$$R_B \downarrow X = \{x \in U | [x]_B \subseteq X\}, \tag{1.4}$$
$$R_B \uparrow X = \{x \in U | [x]_B \cap X \neq \emptyset\}. \tag{1.5}$$

The tuple $\langle R_B \downarrow X, R_B \uparrow X \rangle$ is called a rough set. The objects in $R_B \downarrow X$ mean that they can be with certainty classified as members of X on the basis of knowledge in B, while the objects in $R_B \uparrow X$ can be only classified as possible members of X on the basis of knowledge in B.

In a decision system, for X we use decision concepts (equivalence classes of decision attribute) $[x]_d$. We can define B-lower and B-upper approximations as follows:

$$R_B \downarrow [x]_d = \{x \in U | [x]_B \subseteq [x]_d\}, \tag{1.6}$$
$$R_B \uparrow [x]_d = \{x \in U | [x]_B \cap [x]_d \neq \emptyset\}. \tag{1.7}$$

The positive, negative and boundary of B regions can be defined as:

$$POS_B = \bigcup_{x \in U} R_B \downarrow [x]_d \tag{1.8}$$

The boundary region, BND_B, is the set of objects that can possibly —but not certainly— be classified as follows:

$$BND_B = \bigcup_{x \in U} R_B \uparrow [x]_d - \bigcup_{x \in U} R_B \downarrow [x]_d \tag{1.9}$$

Furthermore, we can calculate the degree of dependency of the decision on a set of attributes. The decision attribute d depends totally on a set of attributes B, denoted $B \Rightarrow d$, if all attribute values from d are uniquely determined by values of attributes from B. It can be defined as follows: for $B \subseteq A$, it is said that d depends on B in a degree of dependency:

$$\gamma_B = \frac{|POS_B|}{|U|} \tag{1.10}$$

The decision table $\mathscr{A}$ is called *consistent* if $\gamma_A = 1$.

Let $(U, \mathscr{A})$ be an information system. The discernibility matrix $M(\mathscr{A})$ is a symmetric $n \times n$ matrix whose elements (c_{ij}) are defined as:

$$c_{ij} = \{a \in A : a(x_i) \neq a(x_j)\} \text{ for } i, j = 1, \ldots, n.$$

In other words, c_{ij} contains those attributes for which objects x_i and x_j differ.

The discernibility matrix can be adapted to work with a decision system, and is then called decision-relative discernibility matrix. It is defined as follows:

$$c_{ij} = \begin{cases} \{a \in A : a(x_i) \neq a(x_j)\}, & \text{if } d(x_i) \neq d(x_j) \\ \varnothing, & \text{otherwise,} \end{cases} \tag{1.11}$$

Application Areas

RST has been used to deal with problems in many areas, sometimes in collaboration with other approaches. For example, Pawlak and Skowron [215] quote many applications that employ RST and boolean reasoning.

This section presents some application areas of both theories: discretization, feature selection, instance selection, rule induction, and nearest neighbor-based classifiers. We focus on them as they are the tasks in which RST is most frequently applied. Additionally, there are three important steps in data modeling and analysis: preprocessing, learning, and prediction.

Discretization. Discretization refers to an approach for converting real-valued attributes into nominal ones in information systems. It should be ensured that this approach maintains the discernibility between objects. Therefore, at the learning stage we may want to produce more general models which avoid overfitting at the prediction step.

Based on the perspective proposed in [67], approaches to the discretization can be classified with regard to three different criteria:

- *global* vs. *local*: it refers to whether approaches evaluate discretizing values over the whole continuous instance/attribute in the information system or localized regions of instances/attributes. For example, the study in [205] proposes both local and global approaches handling of continuous attributes in large data bases.

- *supervised* vs. *unsupervised*: it refers to whether approaches consider values of instances in the discretization process or not. A simple example of an unsupervised approach is an equal width interval method that works by dividing the range of continuous attributes into k equal intervals, where k is given.

- *static* vs. *dynamic*: it refers to whether approaches need a parameter for determining the number of labels/symbolic values or not. In other words, the dynamic approaches are generating the number automatically along the discretization process.

As a basic discretization method, we explain here in detail the maximal discernibility (MD) heuristics presented in [23]. Other methods can be found in, e.g., [23, 205].

Feature Selection. Feature selection is a process to find a subset of attributes which represents the same information as the complete feature set. In other words, the purpose of the feature selection is to identify significant attributes and to eliminate the dispensable ones. An attribute $a \in B \subseteq A$ can be regarded as dispensable in B if $R_B = R_{B\setminus\{a\}}$ otherwise a is called indispensable in B. Furthermore, in both RST and FRST the feature selection typically refers to finding a reduct or a superreduct. A superreduct is a set of attributes $B \subseteq A$, such that $R_B = R_A$, where R_B and R_A are the indiscernibility relations defined by B and A, respectively [214, 216]. If it is also minimal (w.r.t. inclusion), then it is called a reduct. The intersection of all reducts is called the core.

In this section, we focus on calculating a reduct of a decision table which is called decision reduct. A decision reduct of $\mathscr{A} = (U, A \cup \{d\})$ is a minimal (w.r.t inclusion) non-empty set of attributes $B \subseteq A$ such that $\delta_B = \delta_A$, where δ_A is the mapping on U such that for any object x it specifies all rows in the table whose attribute values are the same as for x, and then collects the decision values from each row [216]. Therefore, we can transform the problem of feature selection into looking for criteria that measure the quality of a set of features. Based on the measurement criteria in [260], we can evaluate sets of features to be decision reducts by the following approaches:

1. Conditional independence: Following [259], we define B as a decision reduct iff for any $u \in U$ we have following equality of probabilities P:

$$P_{\mathscr{A}}(d(u)/B(u)) = P_{\mathscr{A}}(d(u)/A(u)),$$

where

$$P_{\mathscr{A}}(d(u)/A(u)) = \frac{|\{u \in U : A(u) = w \wedge d(u) = w'\}|}{|\{u \in U : A(u) = w\}|},$$

for $w \in V_A^U$ and $w' \in V_d^U$.

2. Degrees of consistency: By considering the degree of dependency in Equation (1.10), a set of attributes $B \in A$ is a decision reduct iff

$$\gamma_B = \gamma_A.$$

3. Approximate entropy: By considering entropy as a measure of information [253], we say that $B \in A$ is a decision reduct, iff

$$H_{B(u)}(d(u)) + \log_2(1-\epsilon) \leq H_{A(u)}(d(u)),$$

where

$$H_{B(u)}(d(u)) = -\frac{1}{|U|}\sum_{u \in U} \log_2 P_{B(u)}(d(u)),$$

and for $\epsilon \in [0,1)$.

4. Discernibility relation: We construct the decision-relative discernibility matrix which is based on Equations (1.11). In order to generate decision reducts, we calculate the discernibility function $f_{\mathscr{A}}$ of the matrix. It is a boolean function of m boolean variables $\bar{a}_1,\ldots,\bar{a}_m$ corresponding to the attributes $a_1,\ldots,a_m$ respectively, and defined by

$$f_{\mathscr{A}}(\bar{a_1},\ldots,\bar{a_m}) = \wedge\{\vee \bar{c}_{ij} : 1 \leq j < i \leq n, c_{ij} \neq \varnothing\}, \tag{1.12}$$

where $\bar{c}_{ij} = \{\bar{a} : a \in c_{ij}\}$. The decision reducts of A are then the prime implicants of the function $f_{\mathscr{A}}$. Detailed explanations are given in [258].

Many algorithms have been proposed to find reducts in RST. According to the output produced by these algorithms, we may divide them into three groups: those that produce a superreduct, a set of reducts, or a single reduct.

- Superreduct. An example is the algorithm based on RST proposed by Shen and Chouchoulas [254] called the QuickReduct algorithm. According to Algorithm 1, it can be seen that the computation time depends on the number of attributes instead of the number of objects, which has NP complexity. Other methods based on RST can be found in, e.g., [134, 260, 301].

- Set of reducts. Basically, a set of reducts is obtained by constructing the decision-relative discernibility matrix and then employing the discernibility function. We can construct the matrix by Equations (1.11) and then compute the discernibility function using Equation (1.12).

input : A decision table $\mathscr{A} = (U, A \cup \{d\})$.
output: A superreduct SR.

$SR \leftarrow \{\}$;
repeat
  $T \leftarrow SR$;
  foreach $x \in (A - SR)$ **do**
    if $\gamma_{SR \cup \{x\}} > \gamma_T$ **then** $T \leftarrow SR \cup \{x\}$;
    $SR \leftarrow T$;
  end
until $\gamma_{SR} == \gamma_A$;

Algorithm 1: QuickReduct

- Reduct. There exist two simple ideas to obtain a single reduct which are by considering methods generating a set of reducts and a superreduct. First, through methods constructing the decision-relative discernibility matrix we obtain a set of reducts. After that we can choose a single reduct from the resulting reducts. The other method is by employing algorithms generating a superreduct. Here, we include some procedures to eliminate features. The elimination process is iterated until obtaining a reduct. The algorithm proposed in [133] is an example for obtaining a single reduct based on permutation procedures employing the elimination process.

Rule Induction. Knowledge can be represented in many different ways. Production rules are arguably the most popular knowledge representation. The general structure of such a rule is *IF ... THEN ...*, where the *IF* part refers to the *predecessor* and the *THEN* part to the *successor* [216]. One advantage of building rules is that the model is easy to interpret and manipulate. In this section, we focus on rule induction through RST and FRST techniques.

In RST, a rule for the decision table $\mathscr{A}$ is called a decision rule denoted by *IF* φ *THEN* $d = v$, where $\varphi \in \mathscr{C}(A, V_a)$. $\mathscr{C}(A, V_a)$ is a set of pairs of conditional attributes A and their corresponding values V_a that are connected by the propositional $\wedge$ (conjunction), $\vee$ (disjunction), and $\neg$ (negation). The decision rule is *true* in $\mathscr{A}$ if, and only if, $\| \varphi_{\mathscr{A}} \| \subseteq \| d = v_{\mathscr{A}} \|$ where in this case, $\| . \|$ is the set of objects *matching* the decision rule [216].

For generating rules from data, decision rules of classes in d must meet two properties: *completeness* and *consistency* [192]. Completeness means that for every instance $u \in U$ from class of the attribute d there exists a decision rule representing u while consistency means that there are no two different decision rules that describe the same instance in U. Additionally, in rule induction approaches, there exist three different types: *minimum*, *exhaustive*, and *satisfactory requirements* [267]. In the first case, we generate the smallest number of decision rules that are adequate for describing all given instances. On the other hand, the exhaustive approach refers to algorithms that induce all possible decision rules, whereas the last case contains decision rules that meet the pre-defined requirements.

Many algorithms to induce rules from data based on RST have been proposed. For example, the learning system LERS (Learning from Examples based on RST) introduced the learning from example module, version 2 (LEM2) [95]. It produces a minimal set of rules which is based on computing a single local covering for each concept from the decision table. An improvement of LEM2 is introduced in [96] and the modified learning from examples module, version 2 (MLEM2) algorithm [97]. MLEM2 treats numerical and symbolic attributes differently. For numerical attributes, it computes cut values as in discretization. After that, MLEM2 continues calling the LEM2 processes. Furthermore, an improvement of MLEM2 which is a local version of the method can be found in [100]. The Explore algorithm generating an exhaustive set of rules was presented in [194, 268].

1.6.2 Fuzzy Rough Set Theory

Basic concepts

Just like in RST (see Section 1.6.1), a data set is represented as a table called an information system $\mathscr{A} = (U, A)$, where U is a non-empty set of finite objects as the universe of discourse (note: it refers to all instances/experiments/rows in datasets) and A is a non-empty finite set of attributes, such that $a : U \to V_a$ for every $a \in A$. The set V_a is the set of values that attribute a may take. Information systems that involve a decision attribute, containing classes or decision values of each objects, are called decision systems (or said as decision tables). More formally, it is a pair $\mathscr{A} = (U, A \cup \{d\})$, where $d \notin A$ is the decision attribute. The elements of A are called conditional attributes. However, different from RST, FRST has several ways to express

indiscernibility.

In FRST, it is assumed that $\boldsymbol{R}$ is at least a fuzzy tolerance relation in $\boldsymbol{U}$, that is, $\boldsymbol{R}$ satisfies

- Reflexivity: $\forall x \in U, R(x,x) = 1$.
- Symmetry: $\forall x, y \in U, R(x,y) = R(y,x)$.

Sometimes, additionally the following condition is imposed:

- $\mathscr{T}$-transitivity: $\forall x, y, z \in U, \mathscr{T}(R(x,y), R(y,z)) \leq R(x,z)$.

In this case, $\boldsymbol{R}$ is called a fuzzy $\mathscr{T}$-equivalence relation or $\mathscr{T}$-similarity relation. In a case when $\mathscr{T} = \mathbf{min}$, $\boldsymbol{R}$ is simply called a fuzzy equivalence relation or similarity relation.

For example, Hu *et al.* [119] considered the following kernel-based fuzzy relations:

- Gaussian kernel: $R(x,y) = \exp\left(-\frac{||x-y||^2}{\delta}\right)$
- Exponential kernel: $R(x,y) = \exp\left(-\frac{||x-y||}{\delta}\right)$
- Rational quadratic kernel: $R(x,y) = 1 - \frac{||x-y||^2}{||x-y||^2+\delta}$
- Circular kernel: if $||x-y|| < \delta$, $R(x,y) = \frac{2}{\pi}\arccos\left(\frac{||x-y||}{\delta}\right) - \frac{2}{\pi}\frac{||x-y||}{\delta}\sqrt{1-(\frac{||x-y||}{\delta})^2}$
- Spherical kernel: if $||x-y|| < \delta$, $R(x,y) = 1 - \frac{3}{2}\frac{||x-y||}{\delta} + \frac{1}{2}\left(\frac{||x-y||}{\delta}\right)^3$

where $\delta > 0$. They showed that each of them is a $\mathscr{T}_{cos}$-similarity relation, where the t-norm $\mathscr{T}_{cos}$ is defined as, for a, b in $[0,1]$,

$$\mathscr{T}_{cos}(a,b) = \max\left(ab - \sqrt{1-a^2}\sqrt{1-b^2}, 0\right)$$

Given a fuzzy tolerance relation $\boldsymbol{R}$, it is always possible to transform it into a $\mathscr{T}$-similarity relation [200]. A simple procedure to calculate the min-transitive closure of $\boldsymbol{R}$ is shown in Algorithm 2.

A common way to construct a fuzzy $\boldsymbol{B}$-indiscernibility relation for $\boldsymbol{B} \subseteq \boldsymbol{A}$ proceeds by considering a fuzzy tolerance relation $\boldsymbol{R_a}$ for each quantitative

input : A fuzzy relation R.
output: A **min**-transitive fuzzy relation R^m.

while $R^m \neq R$ **do**
  foreach $x, y \in U$ **do**
    $R^m(x,y) = \max(R(x,y), \max_{z \in U}(\min(R(x,z), R(z,y))))$
  end
  $R \leftarrow R^m$
end

Algorithm 2: Computing the min-transitive closure of a fuzzy relation R.

attribute a, such as the following equations considered by Jensen and Shen in [143]:

$$\begin{aligned} R_a(x,y) &= 1 - \tfrac{|a(x)-a(y)|}{|a_{max}-a_{min}|}, \\ R_a(x,y) &= \exp\left(-\tfrac{(a(x)-a(y))^2}{2\sigma_a^2}\right), \\ R_a(x,y) &= \max\left(\min\left(\tfrac{a(y)-a(x)+\sigma_a}{\sigma_a}, \tfrac{a(x)-a(y)+\sigma_a}{\sigma_a}\right), 0\right), \end{aligned} \tag{1.13}$$

where σ_a^2 is the variance of feature a. For a qualitative (i.e., nominal) attribute a, the classical manner of discerning objects is used, i.e., $R(x,y) = 1$ if $a(x) = a(y)$ and $R(x,y) = 0$, otherwise. Then we can define for any subset B of A, the B-indiscernibility relation by $R_B(x,y) = \mathcal{T}(\underbrace{R_a(x,y)}_{a \in B})$, where $\mathcal{T}$ is a t-norm.

Following Radzikowska and Kerre [238], crisp lower and upper approximations are generalized by means of an implicator $\mathscr{I}$ and a t-norm $\mathcal{T}$. Given a fuzzy indiscernibility relation R_B and a fuzzy set X in U, we define

$$\begin{aligned} (R_B \downarrow X)(y) &= \inf_{x \in U} \mathscr{I}(R_B(x,y), X(x)), \\ (R_B \uparrow X)(y) &= \sup_{x \in U} \mathcal{T}(R_B(x,y), X(x)). \end{aligned} \tag{1.14}$$

Other approaches for defining lower and upper approximations in FRST have been proposed such as vaguely quantified rough sets (VQRS) [53], ordered weighted average rough sets (OWA) [56], fuzzy variable precision rough sets (FVPRS) [309], soft fuzzy rough sets (SFRS) [118], robust fuzzy rough sets (RFRS) [120], and β-precision fuzzy rough sets (β-PFRS) [249].

Regarding regions and degree of Dependency, FRST has the same equations as RST. Furthermore, while in RST the (decision-relative) discernibility matrix is uniquely defined, there exist various alternatives in FRST. For instance, Chen *et al.* [45] defined the decision-relative discernibility matrix in FRST as

$$c_{ij} = \begin{cases} \{a \in A : 1 - R_B(x_i, x_j) \leq \lambda_i\}, & \text{where } \lambda_i = (R_B \downarrow [x_i]_d)(x_i), \text{ if } d(x_i) \neq d(x_j) \\ \oslash, & \text{otherwise.} \end{cases} \tag{1.15}$$

While $M(\mathscr{A})$ based on RST is symmetric, in FRST it is not necessarily symmetric. Other approaches for constructing the decision-relative discernibility matrix based on FRST can be found in, e.g., [46, 145, 282, 309].

Application areas

This section briefly discusses application areas of FRST. Related to the research, we only consider four tasks: feature selection, instance selection, rule-based classifiers (rule induction), and instance-based classifiers using nearest neighbors.

Feature selection. As RST, feature selection in FRST is to find significant attributes by employing the lower and upper approximations. Many algorithms have been proposed to find reducts and superreducts in the FRST setting. For example, Based on FRST, the fuzzy rough QuickReduct algorithm, which is a modified QuickReduct, was proposed in [143]. To obtain the degree of dependency γ, we can calculate the degree by using many variants of lower and upper approximations, e.g., implicator/t-norm approach [54], VQRS [53], OWA [56], FVPRS [309], SFRS [118], RFRS [120], β-PFRS [249]. Intuitively, other modifications can also be made by changing the stopping criterion (as, e.g., in [30]), or by randomizing the features considered for reducing the computation time. The following is the fuzzy rough QuickReduct algorithm [143]:

Instance selection. The aim of instance selection is to remove or replace noisy, superfluous, or inconsistent instances from training datasets but retain consistent ones at the same time. Basically, from the RST perspective, it refers to evaluating each object included in the boundary region. In other

input : A decision table $\mathscr{A} = (U, A \cup \{d\})$.
output: A superreduct SR.

$SR \leftarrow \{\}$; $\gamma_{best} = 0$; $\gamma_{prev} = 0$;
repeat
  $T \leftarrow SR$;
  $\gamma_{prev} \leftarrow \gamma_{best}$;
  foreach $x \in (A - SR)$ **do**
    if $\gamma_{SR \cup \{x\}} > \gamma_T$ **then** $T \leftarrow SR \cup \{x\}$; $\gamma_{best} \leftarrow \gamma_T$;
    $SR \leftarrow T$;
  end
until $\gamma_{best} == \gamma_{prev}$;

Algorithm 3: Fuzzy rough QuickReduct

words, according to the evaluation, we preserve objects in lower approximations, but we change objects included in the boundary region, for example deleting them or changing their class labels to be consistent values.

In FRST, there exist some methods performing instance selection. The fuzzy-rough instance selection (FRIS) method proposes three algorithms that employ a positive region as a measurement tool for selecting instances [138]. If the value of the positive region of objects is less than a threshold value then the objects can be removed. The complete FRIS-1 algorithm is defined as follows:

input : A decision table $\mathscr{A} = (U, A \cup \{d\})$;
A granularity parameter α;
A threshold value τ.
output: A set of selected objects Y.

$Y \leftarrow U$;
foreach $x \in U$ **do**
  if $(POS_A^{\alpha,U}(x) < \tau)$ **then** $Y \leftarrow Y - x$;
end

Algorithm 4: Fuzzy-rough instance selection version 1 (FRIS-1)

The algorithm uses the following indiscernibility relation:

$$R_a^{\alpha}(x,y) = max\left(0, 1 - \alpha\frac{|a(x) - a(y)|}{l(a)}\right),$$

where α is a level of granularity and $l(a)$ is the range of the attribute a. In order to select objects the algorithm offers linear complexity with respect to the number of objects in the dataset, while computation of the positive region needs $O(|A| \cdot |U|^2)$.

The FRIS method is improved in [285] to obtain Fuzzy Rough Prototype Selection (FRPS), a method specifically designed to optimize the accuracy of the k-nearest neighbor (kNN) algorithm. First, instances are ordered according to a measure based on FRST that evaluates the lack of predictive ability of the instances, and a wrapper approach is then used to decide which instances to select.

Rule-based classifiers (rule induction). In case of FRST, an algorithm attempting to combine rule induction and feature selection at the same time is proposed in [140]. Basically it inserts some steps to generate rules in the fuzzy QuickReduct algorithm [141, 143]. In [310], a set of decision rules is induced by constructing the decision-relative discernibility matrix based on the fuzzy variable precision rough sets. There exist other methods proposed to improve previous algorithms such as an extension of LEM2 for FRST called FRLEM2 [175].

Nearest neighbor-based classifiers (instance-based classifiers). The nearest neighbor-based classifiers were introduced first by Fix and Hodges in [79]. Then, in 1967 they were improved and made famous by Cover and Hart [57]. In supervised learning, the k-nearest neighbor-based classifiers are defined as methods that predict new data/patterns based on the most similar or nearest k patterns in training data. This section attempts to illustrate enhancements of k-nearest neighbors based on FRST.

The fuzzy-rough ownership nearest neighbor algorithm was proposed by Sarkar [250] and then improved in [251, 252]. To avoid determining k by trial and error, it uses all training data to construct the fuzzy-rough ownership function. It uses a squared weighted distance between a test pattern and all training data, and constrained fuzzy membership.

The study in [139] considered a summation of lower and upper approximations to predict the class of new instances. It uses lower and upper ap-

proximations based on Equation (1.14) and on VQRS with fuzzy tolerance relations in Equation (1.13). The procedure is outlined in Algorithm 5. The complexity of this algorithm for classification of one object is $O(|U|+K\cdot|d|)$.

input : A decision table $\mathscr{A}=(U,A\cup\{d\})$ as training data; y is new data.
output: $Class$ is a predicted class.

$N \leftarrow NN(y,K)$; `/* NN is the` k`-nearest neighbor algorithm [57] */`
$\tau \leftarrow 0$; $Class \leftarrow \oslash$;
foreach $C \in d$ **do**
  if $((R\downarrow C)(y)+(R\uparrow C)(y))/2 \geq \tau$ **then**
    $Class \leftarrow C$;
    $\tau \leftarrow ((R\downarrow C)(y)+(R\uparrow C)(y))/2$;
  end
end

Algorithm 5: Fuzzy-rough nearest neighbor (FRNN)

The fuzzy rough positive region (POSNN) [286] algorithm considers the positive region to predict the class of new data. Basically, the following steps are used to determine the class of an object y:

1. Determine the set NN of the k-nearest neighbors of y over the training data
2. Select the class for which

$$\frac{\sum_{x\in NN} R(x,y)C(x)POS(x)}{\sum_{x\in NN} R(x,t)}$$

is maximal; where R, C, and POS are a fuzzy relation, a class membership function, and the positive region, respectively. The class membership function [153] is defined as

$$C(x)=\begin{cases}0.51+\frac{n_C}{K}\cdot 0.49, & \text{if } x \text{ is in class } C\\ \frac{n_C}{K}\cdot 0.49, & \text{otherwise,}\end{cases}$$

with n_C the number of instances having class C in NN.

1.7 Decision Trees and Random Forests

1.7.1 Decision Trees

Learning from data by generating tree-based models (DTs) was popularized and unified by Breiman et al. [35], and then were improved by Quinlan [232]. DTs are algorithms that construct a tree as the model by performing a recursive partition of the instance space. There are various frameworks regarding DT methods, such as ID3 [229, 230, 231], C4.5 [232], and CART [35].

DTs have been attempted to deal with many problems. For instance, the study [8] proposes a combination Seasonal Auto Regressive Integrated Moving Average (SARIMA) and DTs for constructing tourism demand modeling. In [202], the DTs were able to determine the individual roles of permeability and solubility in oral absorption process.

Before exploring the methods in more detail, there are several aspects that make them attractive in practical uses, which are as follows [178]:

- DTs are non-parametric. It means that a model mapping inputs and outputs does not need a priori assumption about the probability distributions of the variables being assessed.
- DTs are able to deal with mixed variables: categorical and real values.
- Since DTs calculate impurity or information gain, they implicitly perform feature selection. Therefore, the method is relatively robust to irrelevant and noisy variables and outliers.
- A model generated by DTs is represented in the tree-based structure. So, it is relatively easy to understand and interpret.
- Furthermore, DTs are employed to develop other sophisticated algorithms such as random forests [33] and boosting methods [82, 83].

In the following we briefly explain an introduction to DTs that is divided into the following topics: tree-based model, induction of DTs, stopping criteria, splitting node, application areas, and prediction using tree-based model.

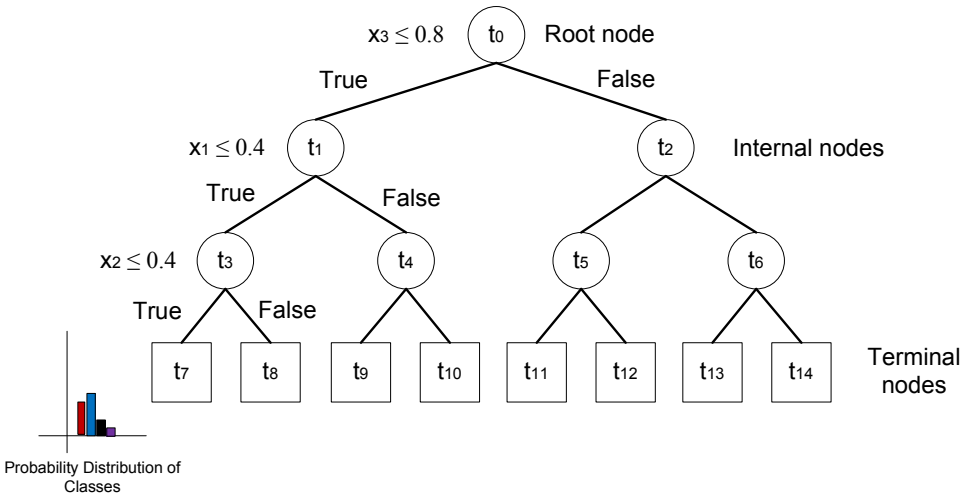


Figure 1.12: A DT model for a binary classification problem.

Tree-Based Model

A model representing a piece of knowledge can be represented in many ways, for example as a rule, instance, or tree format. This part focuses on a tree-based model that is generated by DTs.

A tree is a graph that does not contain a cycle. In other words, nodes and edges build a tree in the hierarchical shape. As shown in Figure 1.12, the root of a tree is a node that does not have an incoming edge, but only outgoing edges. A node that has incoming and outgoing edges is called internal node, whereas a terminal one only has incoming edges. A tree allows to have more than 2 incoming and outgoing edges on each node. However, in this research we only focus on a binary tree, whose root and internal nodes have exactly two outgoing edges.

According to Figure 1.12, basically an internal node divides instances into two spaces by considering the given condition. For example, on the node t_1, the instances X_{t_1} are divided by $x_1 \leq 0.4$ into X_{t_3} on the child t_3 and X_{t_4} on t_4. Furthermore, for classification tasks, on each terminal node we can calculate probability distributions of classes. According the distributions, predicted classes of new data that meet the condition are obtained.

Furthermore, the tree structure can be transformed into rules by tracking from root into terminal nodes with considering the threshold value on each

node. For example, the first rule, which follows the left side of the tree, is as follow:

$$\text{IF } x_3 \leq 0.8 \text{ and } x_1 \leq 0.4 \text{ and } x_2 \leq 0.4 \text{ THEN } C_{t_7}$$

Induction of Decision Trees

As we mentioned above, DTs are algorithms that automatically build a tree from training data. There are various DT methods such as ID3, C4.5, and CART. In this research, we only consider the CART algorithm without pruning for constructing a binary DT, which only has two children on each node. Moreover, the approach is also used as a main part for algorithms of decision forest/random forest.

As other methods in machine learning, there are two steps included in the algorithm, which are learning/training and testing/prediction. The training step refers to a process aimed to grow a tree. Algorithm 6 shows that basically there are four looping steps, which are constructing an initial tree, checking stopping criteria, splitting node, and updating the tree.

Stopping Criteria

The following are criteria used for stopping iterations on Algorithm 6:

1. When the output values of the samples in t are homogeneous. That is, if $y = y_0$ for all $(x, y), (x_0, y_0) \in t$.
2. When the input variables X are each locally constant in t. That is, if $x = x_0$ for all $(x, y), (x_0, y_0) \in t$, for each input variable X.
3. Set t as a terminal node if it contains less than N_{min} samples. (N_{min} is also known as $minSamples_{split}$.)
4. Set t as a terminal node if its depth d_t is greater or equal to a threshold d_{max}. (d_{max} is also known as max_{depth}.)
5. Set t as a terminal node if there is no split such that t_l and t_r both count a least N_{leaf} samples. (N_{leaf} is also known as $minSamples_{leaf}$.)

In all of the above, stopping criteria are defined in terms of user defined hyper-parameters (N_{min}, d_{max}, or N_{leaf}) that have to be tuned in order to

input : Training data.
output: A DT model.

create the root node t_0 and get instances A
push t_0 to an empty stack S
while S *is not empty* **do**
  if *the stopping criterion is met for* t **then**
    t is a leaf node
  end
  else
    find s which is a partition on A that gives maximum information gain
    create the left child t_l according to s
    create the right child t_r according to s
    push t_r to S
    push t_l to S
  end
end

Algorithm 6: Induction of the DT algorithm.

find the right trade-off. Ideally, they need to be such that they are neither too strict nor too loose for the tree to be neither too shallow nor too deep. Too large a tree will have a higher generalization error than the right sized tree. Likewise, too small a tree will not use some of the information in t, again resulting in a higher generalization error than the right sized tree.

Splitting nodes

It refers to a process for dividing a node into two child nodes: right and left, that can be classified into two groups, which are for categorical and real values. For categorical value, we can define a value as a parameter randomly, which represents a percentage of splitting. Using this value, we do not calculate the information gain of all possible parameters according to the percentage. In order to calculate information gain, there are several equations available, such as the Shannon entropy, gini index, etc. In real-valued variable, we can select a single value randomly between minimum and maximum values as the splitting parameter.

Prediction on Tree-Based Model

The second step after learning and constructing a model is the prediction step. It can be done by following a path from the root to a leaf node as in Algorithm 7.

input : A DT model.
output: Predicted values.

$t = t_0$
while t *is not a terminal node* **do**
| t is the child node t_0 of t such that $x \in X_{t_0}$
end

Algorithm 7: Prediction of the output in a DT.

1.7.2 Random Forests

Random forests are a set of methods constructing a model by assembling

DTs that are generated from bootstrap samples and a randomized features. The following is an algorithm of random forest considered in the package:

input : Training data.
output: A random forest model.

for $b = 1$ *to the number of trees* B **do**
  Draw a bootstrap sample from the training data
  Growing DTs with:
  - Select m features at random from all available features,
  - Pick the best variable/split parameter among the m features,
  - Split the node into two child nodes.
end

Algorithm 8: Induction of Random Forest.

It is obvious that in addition to including the DT algorithm, random forest needs the number of trees B and the number of features m as the input data. Therefore, it can be seen that the output of the training step is a model containing a collection of DTs.

Because we now have a collection of trees, in the prediction stage we need to aggregate predicted values of each tree by using aggregation functions considering a particular task, as follows:

- Classification: There are two well-known strategies for aggregating results: majority vote and maximum probability.
- Regression: For this task we just calculate *mean* of predicted values of trees.
- Clustering: We obtain a predicted cluster center by calculating a similarity matrix. Then, we calculate eigen vectors and execute the trivial clustering method, such as k-means.
- Manifold learning: In this case, we produce eigen vectors by considering the number of eigen values as a used-defined parameter.

1.8 Random Ferns

Random ferns are methods employing bootstrap sampling and Naive

Bayesian classification [212]. In order words, the method replace DTs with non-hierarchical ferns based on Bayes's formula. Let c_i, $i = 1,\ldots,H$ be the set of classes and let f_j, $j = 1,\ldots,N$ be the set of binary features that will be training datasets. In prediction, actually we are trying to find the predicted value $\hat{c}_i$, as follows:

$$\hat{c}_i = \underset{c_i}{\operatorname{argmax}} P(C = c_i | f_1, f_2, \ldots, f_N),$$

where C is a random variable representing the class. According to Bayes's formula, we have

$$P(C = c_i | f_1, f_2, \ldots, f_N) = \frac{P(f_1, f_2, \ldots, f_N | C = c_i) P(C = c_i)}{P(f_1, f_2, \ldots, f_N)}.$$

Since other parts are constant, we can also write as

$$\hat{c}_i = \underset{c_i}{\operatorname{argmax}} P(f_1, f_2, \ldots, f_N | C = c_i).$$

By assuming independence between features, we can define the previous equation as

$$\hat{c}_i = \underset{c_i}{\operatorname{argmax}} \prod_{j=1}^{N} P(f_j | C = c_i).$$

Then, we set a group based on the bootstrap algorithm, so now we have the following equation:

$$\hat{c}_i = \underset{c_i}{\operatorname{argmax}} \prod_{k=1}^{M} P(F_k | C = c_i),$$

where the number of group is equal to $\frac{N}{M}$.

The above method was extended by [164] by introducing random ferns for handling non binary features, which include categorical and real-value ones.

1.9 Big Data Processing and Platforms

1.9.1 The Big Data Phenomenon

Recently, The term Big Data has been widely used as a keyword in many documents. According to database of `http://scopus.com` on March 11, 2015, the academic articles including the keyword Big Data have been

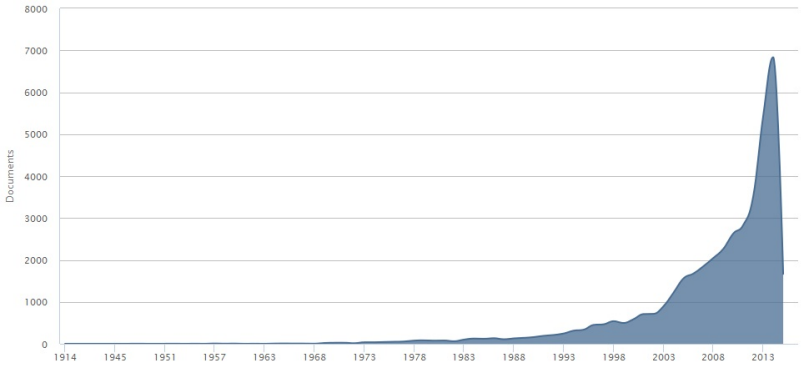


Figure 1.13: The number of published articles containing the keyword Big Data.

significantly increasing since 2003, as shown in Figure 1.13. For example, in 2014 there were 6833 documents whereas we found 3518 and 5440 in 2012 and 2013. These documents are written in several publication media as in Figure 1.14, such as in academic journals/articles (54.4%), conference papers (34.5%), and book chapters (0.8%). Furthermore, Figure 1.15 shows that 30% of the articles is included in the Computer Science area. Meanwhile, engineering, medicine, earth and planetary sciences, and social sciences take 24.5%, 13.6%, 10.1%, and 8.3%, respectively. On the other side, the term Big Data is also used as a headline in public media. For instance, The New York Times publishes an article with the title: "In Big Data, Shepherding Comes First" [176]. BBC News provides a report whether Big Data can help to analyze the Ebola spread in [288]. We can also find other headlines related to Big Data in [179, 61]. In addition, many governments have been announcing their programs involving the technologies of Big Data, e.g., [73, 65]. It shows that Big Data has been attracting researchers and practitioners working in wide and different areas. It also means that the term is an emerging challenges that need to be solved, and it can be a big opportunity offering interesting benefits.

The attention of researchers and practitioners shown in the previous paragraph is forced by real phenomena happened regarding the amount of data that have been exponentially growing recently. Data grow in the incredible size because of many available sources generating them, such as digitized government administrations, Internet of Things (IoT, e.g., GPS), mobile computing (e.g., mobile phone, PDA), social media, sensor data,

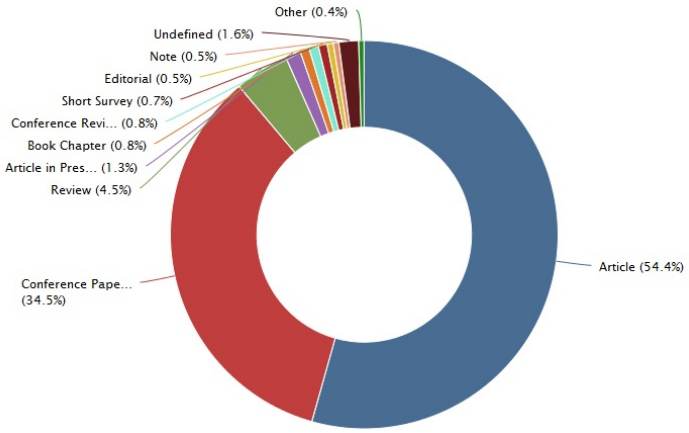


Figure 1.14: The media publishing articles that contain the keyword Big Data.

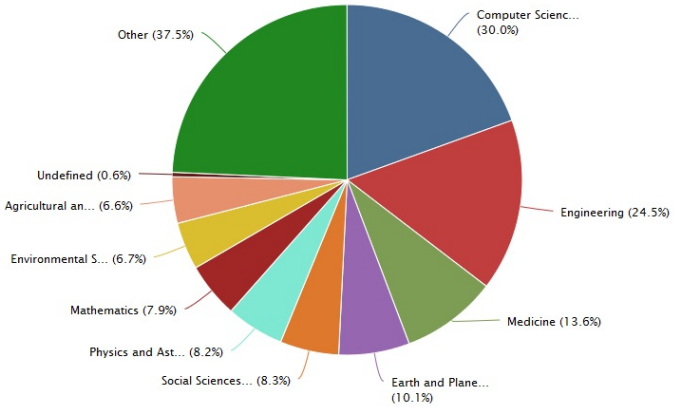


Figure 1.15: The subject areas of published articles including the keyword Big Data.

business applications, public webs, etc. For example, based on a report from International Data Corporation, in 2011, the overall digital data volume in the world was 1.8 zettabytes, which is expected to grow by nearly nine times within five years [87]. A survey in [281] mentions that Facebook handles more than 250 million photo uploads and the interactions of 800 million active users with more than 900 million objects (pages, groups, etc.) for each day. Wal-Mart processes more than a million customer transactions

each hour and imports those into databases estimated to contain more than 2.5 petabytes of data. In addition, we can classify data generator based on the following sources: sensors from electronic devices, social interactions, business transactions, and electronic files. Therefore, not only big companies generate and handle Big Data but also every body in the world. For example, still according to the survey, more than 5 billion people are calling, texting, tweeting and browsing on mobile phones worldwide.

Before stepping further, knowing definitions of Big Data is an essential knowledge. Unfortunately, Big Data is a fuzzy concept, especially on the first word. It may be reasonable that the word big refers to a unit of volume. However, it can be a case: we would say ours are called Big Data when they are just small for others. Of course, the concept of Big Data is not only related to their volume. Due to this reason, several definitions for Big Data have been proposed:

- In 2012, Gartner [29] says: "Big Data is high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making". According to this definition, there are 3Vs, which are volume, velocity, and variety, that should be taken into account. At least, two issues regarding the data volume need to be handled: storage (i.e., memory capacity and performance for reading and writing) and analysis (i.e., computation cost for learning, noise handling, etc). Velocity refers to the speed of data coming in/out the system, such as batch, real time, and stream. Moreover, data may have various types: structured (e.g., table and SQL), unstructured (e.g., email, text, and videos), and semi-structured (e.g., XML). Recently, 3Vs has been extended by adding other Vs: value, veracity, variability, validity, and volatility.

- The study in [112] defines: "Big Data is a set of techniques and technologies that require new forms of integration to uncover large hidden values from large data sets that are diverse, complex, and of a massive scale."

- Microsoft provides a definition as follows: "Big Data is the term increasingly used to describe the process of applying serious computing power — the latest in machine learning and artificial intelligence — to seriously massive and often highly complex sets of information" [193].

- According to National Institute of Standards and Technology (NIST),

Big Data refers to the inability of traditional data architectures to efficiently handle new data sets, which have the following characteristics: high volume, high velocity, and high variety [207].

- The McKinsey Global Institute's Big Data report defines Big Data as datasets whose sizes are beyond the ability of typical software tools to capture, store, manage, and analyze them [186].

Based on the above definitions, there are four important aspects we should consider when dealing with Big Data:

1. Size and complexity: In these aspects, volume, variety, and velocity are critical issues.
2. Technologies: It is clear that traditional technologies are no longer sufficient to face such size and complexity data. Other features that should be taken into account besides the size of storage are scalability, fault tolerance, and accessibility.
3. Algorithms: Sophisticated approaches need to be invented by considering emerging technologies, such as data streaming, parallel and distributed computing, etc. Furthermore, these approaches should be robust, reliable, and smart to tackle uncertainty, noise, imbalanced, and redundant information.
4. Workflow: New procedures should be proposed to analyze Big Data. Interactions among data, technologies, and algorithms can be different from the old processes with small data. For example, instead of bringing data into applications, now we bring and execute code programs to the place where data are. Additionally, visualization by plotting the data is no longer feasible to do.

Nowadays, many problems that can be considered involving Big Data have been carried out. In 2009, the study [90] proposed a method for early detection of disease activity, which is influenza epidemics, by using Google search engine. Netfix, which is a company providing movies and TV shows, faces Big Data to be involved in its recommender systems, streaming, other processing [14]. In Astronomy, [122] mentions that the Large Synoptic Survey Telescope (LSST) in Northern Chile will generate around 150 Petabyte imaging dataset of the sourthern hemispehre sky in 2022. Therefore, the Astroinformatics and Astrostatistics fields have been introduced to deal with Big Data involving machine learning, statistics, and computational intelligence.

1.9.2 The Issues on Big Data

There are two main issues related to Big Data that should be taken into consideration, which are database/storage frameworks and computational models. These aspects are related to each other since even though we have a sophisticated method to deal with Big Data, it is useless if we do not have technology for storing and loading the data. In addition, Big Data saved and managed by data management systems do not offer values and benefits if we can not process and analyze the data.

The first aspect, which is storage frameworks, focuses on technologies and mechanisms to write, read, and manage Big Data efficiently. Furthermore, handling fault tolerance, availability, consistency, scalability, and heterogeneity of Big Data should be considered as well. Traditional data management and analytics systems based on the relational database management system (RDBMS) are no longer able to deal with these criteria. The popular software solution to the problem of huge data management has been Apache Hadoop/YARN [255, 199]. Hadoop's ecosystem provides Hadoop Distributed File System (HDFS) which is a system that manages very large amounts of data through a distributed model. Detailed descriptions about Hadoop will be explained in the next section. Other technologies for storing Big Data are the Google File System (GFS) [89], Colossus[187], Haystack [26], BigTable [43], Hbase [275], MongoDB, Cassandra [166], Amazon's Dynamo system [63], Voldemort [287]. Their comparisons can be found in the study [237].

After discussing the technologies used for managing Big Data, now we should think how the data are processed and analyzed. The MapReduce programming model [62] has turned a new era in the parallelism field. It is a parallel programming model consisting of a map and a reduce function that is used on large-scale clusters of machine. This model is then included in Apache Hadoop as a main core of the parallel computation. In 2010, Spark was introduced, which is a computing model that allows iterative queries and stream processing for Big Data [308]. It is based on an abstraction called Resilient Distributed Datasets (RDD), and contains other functionalities, such as task scheduling, memory management, fault recovery, interacting with storage systems. Since these two models are strongly related to this research, we will explain them in more detail in the next section. Furthermore, in the literature, we can also find other models for parallel computing. For example, Dryad and DryadLINQ [126] developed by Microsoft Research is used as a system and a set of language extensions that enable to perform

large scale distributed computing by employing the programming languages C++, C#, and VisualBasic. A computational model processing problems including large graphs is implemented in Pregel [182]. It was inspired by Valiant's Bulk Synchronous Parallel model [284], and has been considered for efficient, scalable, and fault-tolerant implementation on cluster. Other models handling Big Data by employing parallel and distributed computing are e.g., All-pairs [198] and Message Passing Interface OpenMPI [86]. Furthermore, processing Big Data can also be done through query system on database management systems, such as Dremel [188], X10 [70], Hive and HiveQL [276], BigQuery [93], BigTable [43], and HBase [275]. The detailed survey regarding frameworks for Big Data can be found in [77].

1.9.3 Apache Hadoop

Apache Hadoop offers a vast ecosystem of tools and applications for Big Data. In fact, there are two versions of Hadoop available: Hadoop version 1 and Hadoop version 2 with YARN. In this section we only discuss the latter version. Three main components included in Hadoop YARN, as follows:

- MapReduce: It is a computation framework that allows to break the problems into two steps: mapping and reducing. Between them, it automatically performs shuffling and sorting data. Moreover, we need to take into account that the input data have to be constructed as key-value pairs. Detailed information regarding MapReduce and its applications can be found in [172].

- YARN: It is used as Hadoop's cluster resource management system, which is an interface for requesting and working with cluster resources so that Hadoop becomes a host for other applications as shown in Figure 1.16. It can be seen that other applications, such as Spark, Hive, Tez, etc, can use YARN for utilizing storage management systems based on HDFS. Furthermore, users can also create specific applications for handling their own problems on top of the available computing models.

- HDFS: It is a distributed file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware [255]. According to [294], HDFS produces the following benefits:

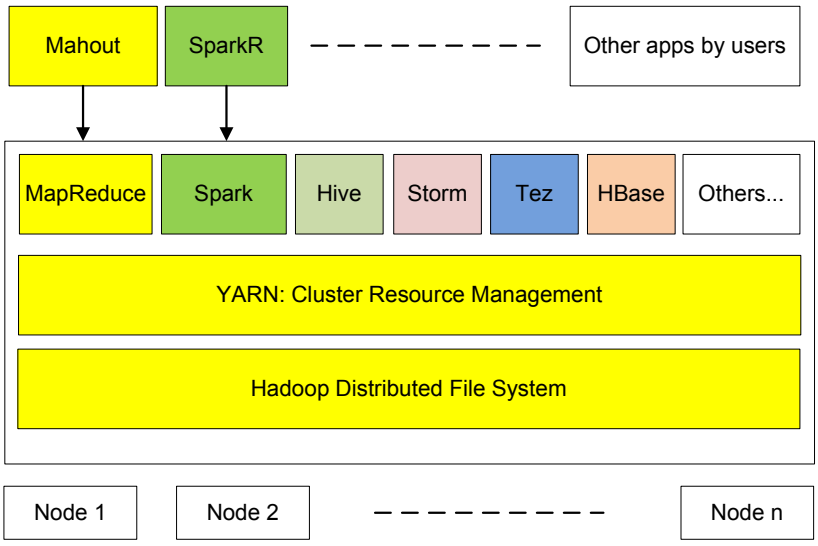


Figure 1.16: The architecture of Apache Hadoop YARN.

- Very large files: It means that HDFS allows to store petabytes of data.
- Streaming data access: It refers to HDFS used for a write-once and read-many-times storage. So, basically HDFS is not so useful for applications requiring low-latency access to data.
- Commodity hardware: A scalable storage system can be built out of commodity hardware.

Furthermore, data saved in HDFS will be distributed along nodes available in the cluster. To manage the system, there are two types of nodes: namenode (master) and datanode (worker). The namenode manages the file system namespace. It maintains the file system tree and the metadata for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files: the namespace image and the edit log. The namenode also knows the datanodes on which all the blocks for a given file are located; however, it does not store block locations persistently, because this information is reconstructed from datanodes when the system starts. Datanodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are

storing.

In summary, Apache Hadoop offers the following benefits: fault tolerance, high availability, scalability, cheapness, distributed file system, multitenancy, and data integrity. Furthermore, it has been used by big companies, such as Amazon, Adobe, Baidu, Facebook, Google, IBM, Linkedln, Twitter, etc.

1.9.4 Apache Spark

Apache Spark aims to extend and generalize MapReduce, which provides computations in memory with several APIs in Python, Java, Scala, R, and SQL [308, 150]. The core component of Spark is the abstraction Resilient Distributed Datasets (RDD). It is simply an immutable distributed collection of objects. Each RDD is split into multiple partitions, which may be processed on different nodes of the cluster at the same time. Furthermore, it contains components for task scheduling, memory management, fault recovery, interacting with storage systems, and more. Spark is designed to be highly accessible, offering simple APIs in Python, Java, Scala, R, and SQL.

Features included in Spark can be seen in Figure 1.17. First of all, Spark can be installed on Hadoop YARN, and it can access any Hadoop data source, including Cassandra. There are other features in Spark [150]:

- SparkR: It integrates R with Apache Spark and enables native R commands in a distributed fashion. By using the package, a code script containing R Spark Context will be translated by the "rJava" to Java Spark Context.

- SparkSQL: It is Spark's package for working with structured data. It allows querying data via SQL as well as the Apache Hive variant of SQL-called the Hive Query Language (HQL)-and it supports many sources of data, including Hive tables, Parquet, and JSON.

- Spark Streaming: It is a Spark component that enables processing of live streams of data. Examples of data streams include logfiles generated by production web servers, or queues of messages containing status updates posted by users of a web service.

- GraphX: It is a library for manipulating graphs (e.g., a social network's friend graph) and performing graph-parallel computations. Like Spark

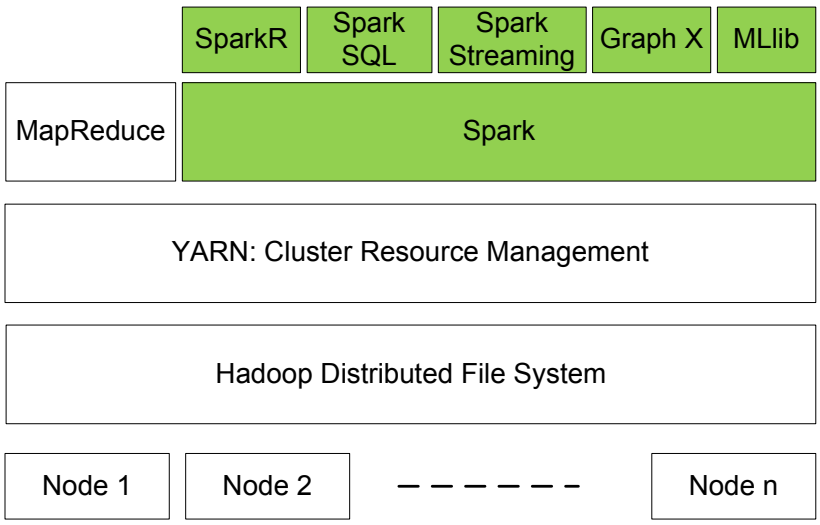


Figure 1.17: The architecture of Apache Spark.

Streaming and SparkSQL, GraphX extends the Spark RDD API, allowing us to create a directed graph with arbitrary properties attached to each vertex and edge.

- MLlib: Spark comes with a library containing common machine learning (ML) functionality, called MLlib. MLlib provides multiple types of machine learning algorithms, including classification, regression, clustering, and collaborative filtering, as well as supporting functionality such as model evaluation and data import.

As we mentioned above, RDD is a core component in Spark. In general, the following are the steps involving RDDs:

- Constructing RDD: Spark provides two ways to construct RDDs:
  - Loading an external dataset: It means that datasets are saved in other places (e.g., HDFS, EC2, etc), and then we load them to Spark. It can be done by typing the following command: *textFile()* and *objectFile()*.
  - Parallelizing an object: In the case objects are created in the Spark environment, we can process them in the parallel way by executing the *parallelize()* command.

- Manipulating/processing RDD: RDD objects can be manipulated by two types of operations: transformations and actions, as follows:
  - Transformation: consisting of operations on RDDs that produce a new RDD, such as *map()* and *filter()*.
  - Action: consisting of operations that return a result to the driver program or write it to storage, and kick off a computation, such as *count()* and *first()*.
- Displaying results: Basically, this step is the same as the action operation. In a particular of saving RDD into files, Spark provides two commands to do that, which are *saveAsTextFile()* and *saveAsObjectFile()*.

1.9.5 R Tools for Big Data

As mentioned in section, R offers many advantages for data analysis. For example, over 6000 packages are ready to use for handling various complex problems, e.g., finance, time series, machine learning, visualizations, and statistics. However, as a common tool R also has drawbacks that need to be resolved. A main weakness related to the research is that data used in computations have to be loaded into Random Access Memory (RAM). It means that the data must be smaller than the available RAM. Therefore, basically R is only able to compute small datasets. Of course, this constraint must be overcome if we are working with Big Data.

In this part, we focus on discussing strategies and implementations in software libraries for dealing with Big Data on R environment. In order to make a simplicity, we classify these approaches as follows:

1. Sampling: It refers to taking a sample of the specified size from the dataset x using either with or without replacement. R provides the primitive function *sample()* and packages aiming to draw and calibrate samples, such as "sampling" [280].
2. Data streaming: It means that data are continuously processed as an ordered sequence of data points. It can tackle the issue of the RAM limitation because R only compute a point of data for single time. We can find several R packages allowing data stream, e.g., "stream" [107], "streamR" [20], and "RMOA" [297]. The package "stream" is a

framework for data stream modeling and associated data mining tasks such as clustering and classification, whereas functions in "streamR" allow R users to access Twitter's filter, sample, and user streams, and to parse the output into data frames. "RMOA" attempts to connect R with Massive Online Analysis [161] to build classification and regression models on streaming data.

3. Memory management: There are R packages dealing with Big Data by managing the memory used for computation. The package "bigmemory" creates, stores, accesses, and manipulates massive matrices by allocating shared memory and may use memory-mapped files [148, 147]. The data structures may be allocated to shared memory, allowing separate processes on the same computer to share access to a single copy of the data set. The data structures may also be file-backed, allowing users to easily manage and analyze data sets larger than the available RAM and share them across nodes of a cluster. Then, a generalized linear model for large datasets is implemented in the "biglm" package [180]. The sophisticated R package "ff" provide data structure that are stored on disk but behave (almost) as if they were in RAM by transparently mapping only a section (pagesize) in main memory [5]. Moreover, "ff" supports R's standard atomic data types 'double,' 'logical,' 'raw' and 'integer' and non-standard atomic types 'boolean' (1 bit), 'quad' (2 bit unsigned), 'nibble' (4 bit unsigned), 'byte' (1 byte signed with NAs), 'ubyte' (1 byte unsigned), 'short' (2 byte signed with NAs), 'ushort' (2 byte unsigned), 'single' (4 byte float with NAs). The"ff" objects store raw data in binary flat files in native encoding, and complement this with metadata stored in R as physical and virtual attributes. They have well-defined hybrid copying semantics, which gives rise to certain performance improvements through virtualization. Moreover, they can be stored and reopened across R sessions. The "ff" files can be shared by multiple "ff" R objects (using different data en/de-coding schemes) in the same process or from multiple R processes to exploit parallelism.

4. Parallel and distributed computing: In this strategy, there are a lot of R packages that can be mentioned. By considering the main focus of the research, we divide the R packages as shown in Figure 1.18. Three groups are considered as follows:

    - MapReduce: It has been explained in Section 1.9.3. Moreover, we will explain R packages supporting the MapReduce paradigm

in more detail in the next section.

- Apache Spark: It has been discussed in Section 1.9.4. Detailed descriptions related to implementations of R tools with Apache Spark will be explained in the next section.

- Other frameworks: It should be noted that we do not mean that packages included in this group are less important and useful than previous ones. In this group, we can consider various frameworks e.g., based on Message Passing Interface (MPI) and Graphics Processing Unit (GPU)-based computing. For example, "PivotalR" [228] utilizes the full power of parallel computation and distributive storage, and thus gives the normal R user access to Big Data. PivotalR also provides the R wrapper for MADlib. MADlib is an open-source library for scalable in-database analytics [114]. It provides data-parallel implementations of mathematical, statistical andmachine-learning algorithms for structured and unstructured data. The R package "pbdMPI" provides an efficient interface to Message Passing Interface (MPI) by utilizing *S4* classes and methods with a focus on Single Program/Multiple Data (SPMD) parallel programming style, which is intended for batch parallel execution [47, 48]. Many functions are included for parallel computing; For example *pbdApply*, *pbdLapply*, and *pbdSapply* have quite similar functionalities as the **apply* family in R, but they are performed in a parallel way. A mechanism of the loop in parallel is accommodated by the package "foreach" [241]. Based on GPU, the "gputools" package offers several common data-mining algorithms which are implemented by an integration between nVidia's CUDA language and cublas library [37]. Matrix algebra on GPU and multicore is implemented in the "magma" package [264]. The package "cudaBayesreg" promises to provide a CUDA implementation of a Bayesian multilevel model for the analysis of brain fMRI data [78]. Other R packages considered for parallel computing are e.g., "snow"[279], "parallelMap" [31], "plyr" [295], and "DistributedR" [239]. The exhaustive list regarding R packages for high performance and parallel computing can be seen at `http://cran.r-project.org/web/views/HighPerformanceComputing.html`.

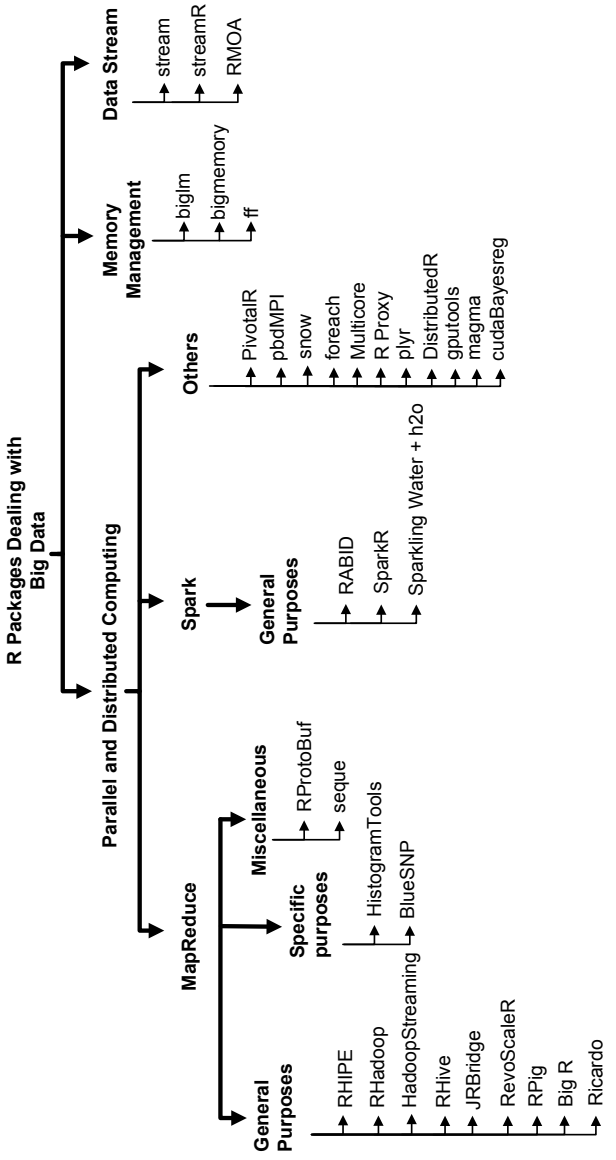


Figure 1.18: A survey on R packages for Big Data.

R with MapReduce

As shown in Figure 1.18, according to the functionalities the following three groups are considered in the implementations of R with MapReduce:

general purposes, specific purposes, and miscellaneous. The following are R packages included in these classifications:

- General purposes: These packages are used as a framework, so that we can create a specific application on top of Apache Hadoop.
  - "RHadoop" [6, 240]: It is a well-know package integrating R and Hadoop that was created by Revolution analytics. In current version, it contains five R packages:
    1. "rmr2": It is an interface to perform MapReduce inside the R environment. Therefore, as MapReduce in Hadoop, to use the package we need to write three parts of code, which are map, reduce, and driver program.
    2. "rhdfs": It contains functions that provide the HDFS commands from the R environment for managing files, such as put, copy local files to HDFS, make a directory, etc.
    3. "rhbase": It is an R interface for manipulating Hbase from R, such as creating and modifying Hbase tables.
    4. "plyrmr": It is a package that allows to process on a Hadoop cluster of large data sets based on "rmr2". For example, it provides functions related to the data.frame operations such as transmute, select, melt, etc.
    5. "ravro": It is used to read and write files in the Apache Avro (`https://avro.apache.org/`) serialization format.
  - "RHIPE" [104]: It is an R package that integrate R and Hadoop based on the divide and recombine paradigm. Furthermore, it consists of several function to interact with HDFS, e.g. create, delete, and copy. The output of the computation can be produced in the PDF files, R datasets, CVS files, etc.
  - "HadoopStreaming" [248]: It provides a framework for writing MapReduce for use in Hadoop Streaming. It Also facilitates operating on data in a streaming manner, without Hadoop. In the package, there are 3 functions for reading data:
    1. *hsTableReader()*: It is used for reading data in table format.
    2. *hsKeyValReader()*: It is used for reading key/value pairs, where each is a string.
    3. *hsLineReader()* It is used for reading entire lines as string, without any data parsing.

Generally, in order to perform MapReduce using the package, we consider the following three steps: making a connection, writing mapper and reducer, and running on command line the functions with separated by a vertical line.

- "Ricardo" [60]: It is a part of the eXtreme Analytics Platform (XAP) project at the IBM Almaden Reseach Center. For data processing, the package decompose data and save them into HDFS. Then they are computed by R separately. Furthermore, the decomposition minimizes the through put cost to improve the computation performance.
- "JRBridge" [304]: It offers an integration of R with Hadoop by employing JVM-based computational infrastructures, which uses Java APIs code wrapper around the native R code automatically and tackling type conversion. Moreover, it supports to store data into HDFS and implements MapReduce.
- "RHive" [203]: It is an R package for distributed computing via HIVE query. Moreover, it presents functions connecting HIVE SQL and R objects (HQL).
- "RevoScaleR"[265]: It offers a mechanism for scaling the R language to deal with Big Data. It contains three major components, as follows:
  1. A new file format especially designed for large files, which is the XDF file format.
  2. External memory implementations of the statistical algorithms most commonly used with large data sets.
  3. An extensible programming framework to write user-defined external memory algorithms.
- "RPig" [110]: The basic idea of the package is to provide an interface between R and Pig. Pig is a high level programming language used in Hadoop. Furthermore, it provides a scalable advanced data analysis solution for machine learning and statistical analysis.
- "BigR" [123]: It is an R package aimed to integrate R with IBM InfoSphere BigInsights for dealing with Big Data with the Hadoop/MapReduce framework. It also provides *bigr.frame*, *bigr.vector*, and *bigr.list* which are similar to native R data structures, but they are for Big Data.

- Specific purposes:
  - "HistogramTools" [269]: In Big Data, creating and manipulating histograms are not longer easy tasks. This package provides a number of utility functions used for manipulating large histograms, such as trim, subset, merge buckets, and merge histograms.
  - "BlueSNP" [121]: It implements genome-wide association studies (GWAS) statistical tests in the R environment and Hadoop. Basically, the "BlueSNP" package depends on "Rhipe" to communicate with Hadoop.
- Miscellaneous:
  - "RProtobuf" [71, 81]: It is an interface for encoding R data structures to the Google Protocol Buffers, which are a language for data interchange format that independent of programming languages or operating systems. Moreover, it is embedded in "HistogramTools".
  - "seque" [177]: It allows to do parallel processing on Amazon Web Services (AWS) Elastic Map Reduce, so that we can easily perform *lapply*-style operations.

R with Apache Spark

In this part, we can find the following packages to deploy R and Apache Spark jointly:

- General purposes:
  - "RAPID" [171]: It is a distributed framework for data analysis by integrating R and Apache Spark. There are three mechanisms for dealing with Big Data: using the Apache Spark framework, user distributed data structures (e.g., *lapply* and *aggregate*), and optimizations on computations.
  - "SparkR" [15]: It integrates R with Apache Spark and enables native R commands in a distributed fashion. By using the package, a code script containing R Spark Context will be translated by the "rJava" package to Java Spark Context. Then, it will be distributed to whole workers/nodes on the cluster. On each

worker, the program will be transformed and executed in the R environment. This package offers the following advantages:

  * All features included in Apache Spark are inherited by the "SparkR" package, such as fault tolerance, scalable storage, distributed file systems, and the RDD schema.
  * Since the package works on R, we are allowed to include any packages built in R by calling *includePackage()*. This feature is the most important aspect since R contains a huge number of packages for visualizations, statistics, machine-learning methods, etc.
  * It works on the interactive mode, which is easy to use for data analysis.
  * As Apache Spark naturally written in the Scala functional programming, "SparkR" can be quite similar since R supports the functional programming as well.

– "Sparkling Water" (`http://0xdata.com/`): It is a framework that integrate Apache Spark with the H2O platform. H2O allows to apply math and predictive analytics to solve business problem. It offers many benefits, such as open source, familiar interface, interchangeable data structure (e.g., Microsoft Excel, Tableau, HDFS, SQL, and NoSQL), massively scalable Big Data analysis, and real time data scoring. Regarding data science, H2O provides various methods: cox proportional hazards Model, deep learning, generalized linear model, gradient boosted regression and classification, $\boldsymbol{k}$-means, naive Bayes, principal component analysis, random forest, etc. Furthermore, we find the "h2o" package allowing to work with H2O from the R environment [84].

Chapter 2

The "frbs" Package

This chapter aims to answer Objective 1(a), namely to produce a high quality and easy to use implementation of the most widely used FRBS models as well as methods to learn them. We are mainly concerned with FRBS applied on classification and regression problems. This software will incarnate in the shape of an R package. Furthermore, we will explain main features and architecture of the package. After that, some examples of usage and a comparison with other packages are presented. Lastly, we explain a short summary regarding the chapter.

2.1 Introduction

As mentioned above, FRBSs are well known methods within computational intelligence, based on fuzzy concepts to address complex real-world problems. They have become a powerful method to tackle various problems such as uncertainty, imprecision, and non-linearity. They are commonly used for identification, classification, and regression tasks. On CRAN, there are already some packages present that make use of fuzzy concepts. The "sets" package [191] includes the fundamental structure and operators of fuzzy sets: class construction, union, intersection, negation, etc. Additionally, it provides simple fuzzy inference mechanisms based on fuzzy variables and fuzzy rules, including fuzzification, inference, and defuzzification. The package "fuzzyFDR" [168] determines fuzzy decision rules for multiple testing of hypotheses with discrete data, and genetic algorithms for learning

FRBSs are implemented in the package "fugeR" [38]. The "e1071" package [190] provides many useful functions for latent class analysis, support vector machines, etc. With respect to fuzzy concepts, this package offers implementations of algorithms for fuzzy clustering, and fuzzy k-means, which is an enhancement of the k-means clustering algorithm using fuzzy techniques. According to this short survey, it can be seen that a comprehensive and effective tool, which unify the following features: constructing FRBS models by learning from data using well-known learning methods and by human experts manually, is still missing. Once identified the need, we set the goal of the research through a complete package in R. The result is the "frbs" package.

The "frbs" package is written in pure R and provides implementations of the most relevant models of FRBSs and more than fifteen different learning methods to construct FRBSs from data for regression and classification tasks. Furthermore, constructing FRBS models can also be done by human experts manually. The package is available from CRAN at `http://CRAN.R-project.org/package=frbs`. As of this writing, the package version is 3.1-0. It is developed under the term GPL v.3. Moreover, the web page providing some examples in detail can be found in `http://sci2s.ugr.es/dicits/software/FRBS`.

Figure 2.1 shows global features available in the package. Two main fashions to construct an FRBS model: learning from data by using learning methods and defined by human experts manually. It implements FRBSs based on the Mamdani, TSK, and FRBCS models. For the case of learning from data, five different models can be produced: Mamdani, TSK, FRBCS, approximate, and clustering.

Furthermore, in the package, some facilities are provided such as plotting membership functions, a summary of FRBS models, demos, a structured manual, and datasets. As other R packages submitted in CRAN, to keep a standard quality of CRAN package, "RoughSets" has been checked by the CRAN teams. Moreover, "frbs" has been included in the CRAN view: Machine Learning & Statistical Learning.

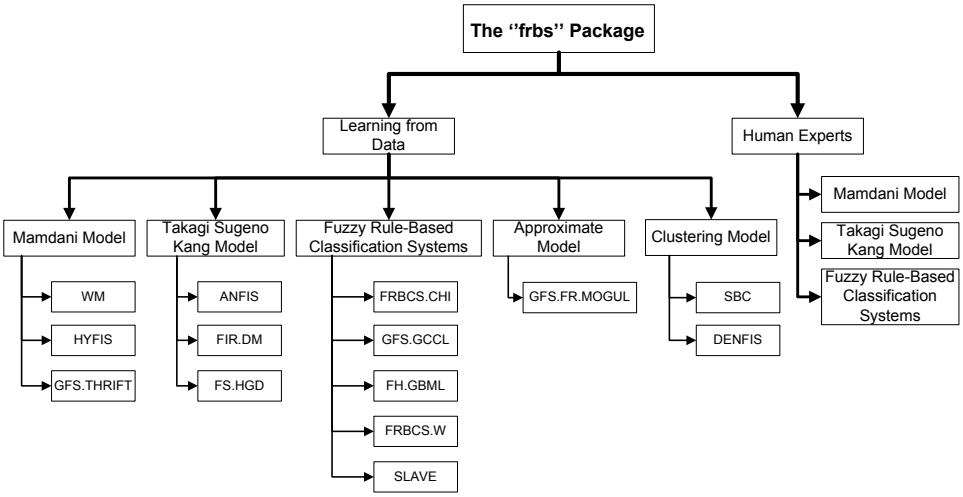


Figure 2.1: Main features in the "frbs" package.

2.2 The Package Architecture and Implementation Details

Regarding the learning approaches to construct FRBS models considered in "frbs". They can be classified into five groups:

1. FRBS based on space partition: It refers to any approach using a strategy of splitting the variable space, and then considering these partitions to obtain parameters of membership functions. We have implemented WM [292], FRBCS.CHI [49], and FRBCS.W [127].

2. FRBS based on gradient descent: It refers to approaches using the gradient descent approach to optimize parameters on both the antecedent and consequent parts of rules, for example FIR.DM [208] and FS.HGD [130].

3. FRBS based on genetic algorithms: It refers to GFS which is a combination of FRBSs with genetic algorithms where the genetic algorithms are used to search and optimize parameters of membership functions and of the fuzzy rule construction process [52, 115]. We have implemented GFS.Thrift [278], GFS.FR.MOGUL [116], GFS.GCCL [128], FH.GBML [131], GFS.LT.RS [9], and SLAVE [91]. In the case of the

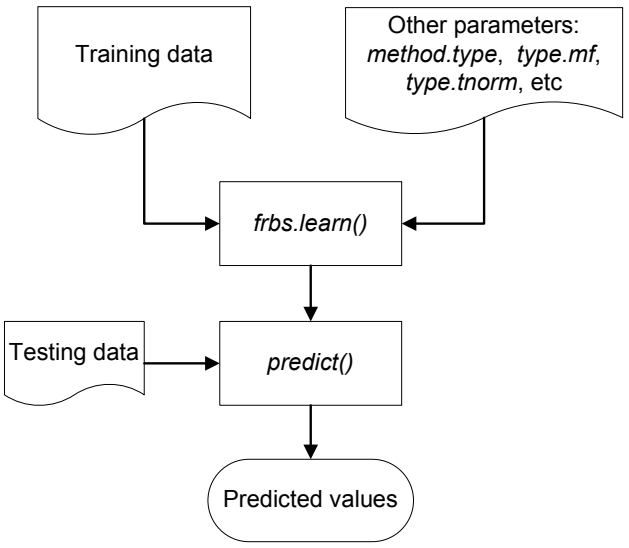


Figure 2.2: Constructing an FRBS model from data and the reasoning process (prediction)

GFS.GCCL and FH.GBML algorithms, they introduce a new term which is *dont_care* for simplification rule bases.

4. FRBS based on neural networks: The systems using neural networks, called FNN [36], combine artificial neural networks with FRBSs. An FRBS is laid upon the structure of an artificial neural networks and the learning algorithm of the latter is used to adapt the FRBS parameters, usually the membership function parameters. In this group, we have considered ANFIS [132] and HYFIS [155].

5. FRBS based on clustering: It refers to FRBSs constructed by clustering approaches through representing cluster centers as fuzzy rules. We have included SBC [50] and DENFIS [151].

In order to construct an FRBS model by learning from data, we just follow the process illustrated in Figure 2.2. There are only two reasonable steps which are executing *frbs.learn()* and *predict()* used for learning data and prediction, respectively.

"frbs" allows to build an FRBS model defined by human experts manually. To follow this approach, we need to supply rulebase, database and

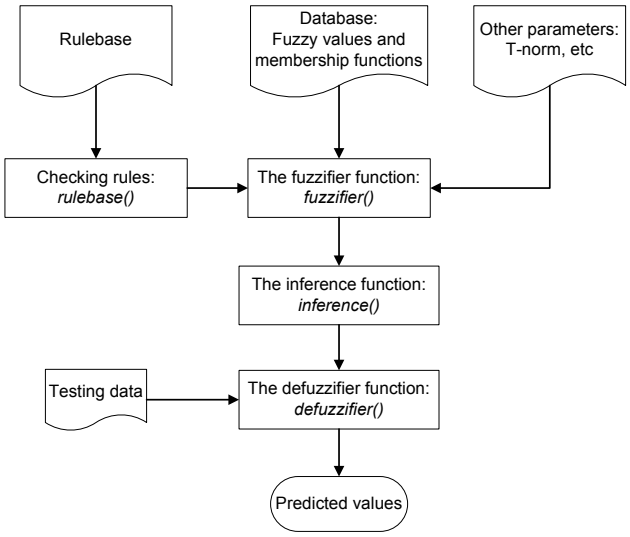


Figure 2.3: Constructing an FRBS model by human experts and the reasoning process

other parameters for reasoning processes as depicted by Figure 2.3. Rulebase contains a set of fuzzy rules that defines a matrix whereas the database consists of definitions of fuzzy values and a matrix representing parameters on membership functions. For checking consistency of rules, we need to perform *rulebase()*. After defining all required parameters, we invoke the following functions: *fuzzifier()* and *inference()*. Predicted values are obtained by supplying testing data along with the FRBS model into *defuzzifier()*.

Besides providing many learning methods, the "frbs" package offers many other functionalities to construct an FRBS model. First, the package implements various choices of the following parameters:

- Triangular norm (t-norm): It is used for computing the two-valued logical conjunction. We provide the following options:
  - "MIN": referring to the standard t-norm: $\mathbf{min}(x_1, x_2)$,
  - "HAMACHER": referring to the Hamacher product: $(x_1 * x_2)/(x_1 + x_2 - x_1 * x_2)$,
  - "YAGER": referring to the Yager class: $1 - \mathbf{min}(1, ((1 - x_1) + (1 - x_2)))$,
  - "PRODUCT": referring to the product operator: $(x_1 * x_2)$,

  - "BOUNDED": referring to the bounded product: $\max(0, x_1 + x_2 - 1)$.

- Triangular conorm (s-norm): It is used for computing the two-valued logical disjunction. We provide the following options:
  - "MAX": referring to the standard s-norm: $\max(x_1, x_2)$,
  - "HAMACHER": referring to the Hamacher sum: $(x_1 + x_2 - 2 * x_1 * x_2)/(1 - x_1 * x_2)$,
  - "YAGER": referring to the Yager class: $\min(1, (x_1 + x_2))$,
  - "SUM": referring to the sum operator: $(x_1 + x_2 - x_1 * x_2)$,
  - "BOUNDED": referring to the bounded sum: $\min(1, x_1 + x_2)$.

- Implicator functions: Methods used to calculate the implication on a rule. We provide the following options, where a rule is expressed by $a \rightarrow b$:
  - "DIENES_RESHER" means $(b > 1 - a ? b : 1 - a)$,
  - "LUKASIEWICZ" means $(b < a ? 1 - a + b : 1)$,
  - "ZADEH" means $(a < 0.5 \| 1 - a > b ? 1 - a : (a < b ? a : b))$,
  - "GOGUEN" means $(a < b ? 1 : b/a)$,
  - "GODEL" means $(a \leq b ? 1 : b)$,
  - "SHARP" means $(a \leq b ? 1 : 0)$,
  - "MIZUMOTO" means $(1 - a + a * b)$,
  - "DUBOIS_PRADE means $b == 0 ? 1 - a : (a == 1 ? b : 1)$,
  - "MIN" means $(a < b ? a : b)$

- Defuzzification methods: The following are methods available:
  - "WAM": it refers to the weighted average method,
  - "FIRST.MAX": it refers to the first maxima,
  - "LAST.MAX": it refers to the last maxima,
  - "MEAN.MAX": it refers to the mean maxima,
  - "COG": it refers to the modified center of gravity.

- Membership functions: To perform fuzzification, which is a process for determining a degree of membership, we provide the following functions:

- "TRIANGLE": representing the triangular function, which has three parameters (a, b, c) where b is the center point, and a and c are the left and right points, respectively.
- "TRAPEZOID": representing the trapezoidal function, which has four parameters representing the corner points: (a, b, c, d).
- "GAUSSIAN": representing the Gaussian function, which has two parameters: mean c and deviation σ.
- "SIGMOID": representing the sigmoid function, which has two parameters expressing steepness and distance from the origin: (γ, c).
- "BELL": representing the generalized bell function, which has three parameters (a, b, c).

Moreover, even though we focus on constructing an FRBS model by learning from data using various learning methods, we facilitate users to build an FRBS model manually from knowledge of human experts. Also, to obtain a representative model, experts can define linguistic hedges. The kinds of hedges that can be used are

- "extremely" reduces the truth value, e.g., membership function "extremely a_1": $\mu(a_1)' = \mu(a_1)^3$,
- "very" reduces the truth value, e.g., membership function "very a_1": $\mu(a_1)' = \mu(a_1)^2$,
- "somewhat" increases the truth value, e.g., membership function "somewhat a_1": $\mu(a_1)' = \mu(a_1)^{0.5}$,
- "slightly" increases the truth value, e.g., membership function "slightly a_1": $\mu(a_1)' = \mu(a_1)^{0.33}$,

The *dont_care* is a lingustic value representing a value that always has the degree of 1 so that we can minimize the complexity of the rules.

In summary, Table 2.1 shows the main functions in the package, where the last three are functions designated for managing the frbsPMML format that will be explained in Section 3.1. First, there are two functions that are used for constructing models: *frbs.learn()* and *frbs.gen()*. Then, the two functions *frbsPMML()* and *write.frbsPMML()* are used for converting FRBS models to frbsPMML. Finally, to obtain prediction for new data, there is the

function *predict()*. Two additional functions: *summary()* and *plot.MF()*, are used to display an FRBS model in the R environment and plot membership functions, respectively.

Table 2.1: The main functions of the "frbs" package.

Functions	Description
frbs.learn()	It is a main function used to construct an FRBS model automatically from data.
predict()	It performs fuzzy reasoning to obtain predicted values for new data, using a given FRBS model.
frbs.gen()	It is used to construct an FRBS model manually from expert knowledge.
summary()	It is used to show a summary of an FRBS model.
plotMF()	It is used to plot the membership functions.
frbsPMML()	It is a main function used to convert a model to the frbsPMML format.
read.frbsPMML()	It is used to read and convert a model in frbsPMML format to an R object.
write.frbsPMML()	It is used to write and save a model in frbsPMML format to a file.

2.3 Examples of Usage

In order to use "frbs", the following steps should be considered.

1. Install and load "frbs". We need to install "frbs" before it is available. It should be noted that this has to be done only once. We can install it from CRAN directly or from a local file in, e.g., .zip and .tar.gz formats.

2. Prepare data. Usually, data are splitted into two parts: training and testing. Data are not allowed to contain missing values and should be in a *matrix* or *data.frame* type.

3. Construct an FRBS model. We construct an FRBS model by executing *frbs.learn()*.

4. Predict/inference new data. We predict new data by calling *predict()*.

5. Report or summarize the model. The package provides functions to make a summary of the model and plot the membership functions.

In this section, we provide three examples of the use of "frbs": regression, classification, and human expert construction. The first two refers to an extraction of knowledge from data by using learning methods, whereas human experts construct an FRBS model manually on the latter.

2.3.1 Installation and Loading the "frbs" Package

Before using "frbs", first we need to install it from CRAN by the following simple command in the R environment.

```
R> install.packages("frbs")
```

After installing the package, in any session using "frbs" we need to load it with the command:

```
R> library(frbs)
```

which makes any functions of "frbs" available in the R environment. We can see a list of functions included in "frbs" by typing the following code.

```
R> library(help=frbs)
```

All R functions available in "frbs" are documented in the R hypertext and pdf format. The manual of "frbs" in pdf format can be found in [244]. Furthermore, to get information of a particular function, we can apply the *help* command as follows:

```
R> help(frbs.learn)
```

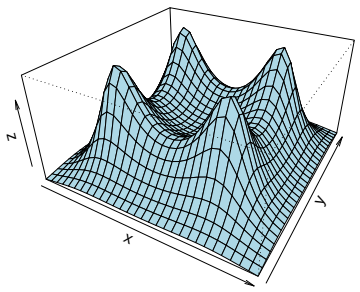


Figure 2.4: The *four hill* function.

2.3.2 Regression Problem

In the regression task, we describe how to use "frbs" to predict real-valued output based on the input variables expressed by a continuous function. The following is a function called the *four hill* function, which is plotted in Figure 2.4. It involves two input variables $x \in [-2,2]$ and $y \in [-2,2]$.

$$f(x,y) = \frac{1}{x^4 + y^4 - 2x^2 - 2y^2 + 3}$$

To provide training and testing data, we need to generate data according to the function in a matrix format as follows. Here, we are using step size 0.14 and assigning the output to z. We will obtain a matrix containing 841 rows and 3 columns representing the X, Y, and Z variables.

```
R> fun <- function(input.xy){
+     z <- 1/(input.xy[1]^4 + input.xy[2]^4
+     - 2 * input.xy[1]^2 - 2 * input.xy[2]^2 + 3)
+  }
R> input.xy <- expand.grid(seq(-2, 2, by = 0.14),
+    seq(-2, 2, by = 0.14))
R> z <- apply(input.xy, 1, fun)
```

```
R> data <- cbind(input.xy, z)
R> colnames(data)<- c("X", "Y", "Z")
```

After that, we split the data into two parts: training data and testing data. For example, we use 80% of the data for training, and the rest for testing, as follows [1]:

```
R> cut.indx <- round(0.8 * nrow(data))
R> data.tra <- data[1:cut.indx,]
R> data.tst <- data[(cut.indx + 1):nrow(data), 1:2]
R> real.val <- data[(cut.indx + 1):nrow(data), 3,
+    drop=FALSE]
```

Then, we need to calculate the interval of each variable by

```
R> range.data <- apply(data, 2, range)
```

So, now our data is ready to use.

In order to construct an FRBS model, we need to assign values to available parameters. We note that all parameters have default values, if we ignore them. For instance, we use the Wang and Mendel's algorithm ("WM") as the learning method and assign it to the parameter *method.type*. Then, we define other parameters in the *control* parameter, for instance the number of linguistic values, 5. And, we use the center of gravity ("COG"), "MIN", and "LUKASIEWICZ" to be our defuzzification method, types of t-norm, and implicator operators, respectively. Finally, let us call our simulation "fourhill" by assigning the parameter *name*, as follows:

```
R> method.type <- "WM"
R> control <- list(num.labels = 5,
+    type.mf = "GAUSSIAN", type.defuz = "COG",
+    type.tnorm = "MIN", type.implication.func = "LUKASIEWICZ",
+    name="fourhill")
```

It is a simple way to execute the learning method as follows.

[1] Indicate that the sampling should be done randomly, but it is not done here, because the example is just for illustration purposes

```
R> mod.reg <- frbs.learn(data.tra,range.data,
+    method.type,control)
```

We can summarize our model by the following command.

```
R> summary(mod.reg)

The name of model:  fourhill
Model was trained using:  WM
The names of attributes:  X Y Z
The interval of training data:
        X     Y          Z
min -2.00 -2.00 0.05263158
max  1.92  1.92 0.99674630
Type of FRBS model:
[1] "MAMDANI"
Type of membership functions:
[1] "GAUSSIAN"
Type of t-norm method:
[1] "Standard t-norm (min)"
Type of s-norm method:
[1] "Standard s-norm"
Type of defuzzification technique:
[1] "modified COG"
Type of implication function:
[1] "LUKASIEWICZ"
The names of linguistic terms on the input variables:
 [1] "very.small" "small"      "medium"     "large"      "very.large"
 [6] "very.small" "small"      "medium"     "large"      "very.large"
The parameter values of membership function on the input variable
(normalized):
     very.small  small medium  large very.large very.small  small
[1,]     5.0000 5.0000 5.0000 5.0000     5.0000     5.0000 5.0000
[2,]     0.0000 0.2500 0.5000 0.7500     1.0000     0.0000 0.2500
```

```
[3,]     0.0875 0.0875 0.0875 0.0875     0.0875     0.0875 0.0875
[4,]         NA     NA     NA     NA         NA         NA     NA
[5,]         NA     NA     NA     NA         NA         NA     NA
     medium  large very.large
[1,] 5.0000 5.0000     5.0000
[2,] 0.5000 0.7500     1.0000
[3,] 0.0875 0.0875     0.0875
[4,]     NA     NA         NA
[5,]     NA     NA         NA
The names of linguistic terms on the output variable:
[1] "very.small" "small"      "medium"     "large"      "very.large"
The parameter values of membership function on the output variable
(normalized):
     very.small  small medium  large very.large
[1,]     5.0000 5.0000 5.0000 5.0000     5.0000
[2,]     0.0000 0.2500 0.5000 0.7500     1.0000
[3,]     0.0875 0.0875 0.0875 0.0875     0.0875
[4,]         NA     NA     NA     NA         NA
[5,]         NA     NA     NA     NA         NA
The number of linguistic terms on each variables
     X Y Z
[1,] 5 5 5
The fuzzy IF-THEN rules:
   V1 V2 V3         V4  V5 V6 V7         V8   V9 V10 V11        V12
1  IF  X is very.small and  Y is very.small THEN   Z  is very.small
2  IF  X is      small and  Y is very.small THEN   Z  is very.small
3  IF  X is     medium and  Y is very.small THEN   Z  is very.small
4  IF  X is      large and  Y is very.small THEN   Z  is very.small
5  IF  X is very.large and  Y is very.small THEN   Z  is very.small
6  IF  X is      small and  Y is very.small THEN   Z  is      small
7  IF  X is     medium and  Y is very.small THEN   Z  is      small
8  IF  X is      large and  Y is very.small THEN   Z  is      small
9  IF  X is very.large and  Y is very.small THEN   Z  is      small
```

```
10 IF  X is very.small and  Y is      small THEN   Z  is very.small
11 IF  X is very.small and  Y is      small THEN   Z  is      small
12 IF  X is      small and  Y is      small THEN   Z  is      small
13 IF  X is      small and  Y is      small THEN   Z  is     medium
14 IF  X is     medium and  Y is      small THEN   Z  is      small
15 IF  X is      large and  Y is      small THEN   Z  is      small
16 IF  X is      large and  Y is      small THEN   Z  is     medium
17 IF  X is very.large and  Y is      small THEN   Z  is      small
18 IF  X is very.large and  Y is      small THEN   Z  is very.small
19 IF  X is     medium and  Y is      small THEN   Z  is     medium
20 IF  X is      small and  Y is      small THEN   Z  is      large
21 IF  X is      large and  Y is      small THEN   Z  is      large
22 IF  X is      small and  Y is      small THEN   Z  is very.large
23 IF  X is      large and  Y is      small THEN   Z  is very.large
24 IF  X is very.small and  Y is     medium THEN   Z  is very.small
25 IF  X is very.small and  Y is     medium THEN   Z  is      small
26 IF  X is      small and  Y is     medium THEN   Z  is      small
27 IF  X is      small and  Y is     medium THEN   Z  is     medium
28 IF  X is     medium and  Y is     medium THEN   Z  is     medium
29 IF  X is     medium and  Y is     medium THEN   Z  is      small
30 IF  X is      large and  Y is     medium THEN   Z  is     medium
31 IF  X is very.large and  Y is     medium THEN   Z  is      small
32 IF  X is very.large and  Y is     medium THEN   Z  is very.small
33 IF  X is      large and  Y is     medium THEN   Z  is      small
34 IF  X is very.small and  Y is      large THEN   Z  is very.small
35 IF  X is very.small and  Y is      large THEN   Z  is      small
36 IF  X is      small and  Y is      large THEN   Z  is      small
37 IF  X is      small and  Y is      large THEN   Z  is     medium
38 IF  X is     medium and  Y is      large THEN   Z  is     medium
39 IF  X is      large and  Y is      large THEN   Z  is     medium
40 IF  X is very.large and  Y is      large THEN   Z  is      small
41 IF  X is very.large and  Y is      large THEN   Z  is very.small
42 IF  X is      small and  Y is      large THEN   Z  is      large
```

```
43 IF  X is      large and  Y is      large THEN   Z  is      large
44 IF  X is      small and  Y is      large THEN   Z  is very.large
45 IF  X is      large and  Y is      large THEN   Z  is very.large
46 IF  X is very.small and  Y is very.small THEN   Z  is      small
47 IF  X is     medium and  Y is      large THEN   Z  is      small
```

The reader can also refer to the project web site `http://dicits.ugr.es/software/FRBS/` in order to see the model. And, we plot the membership functions as seen in Figure 2.5 by

```
R> plotMF(mod.reg)
```

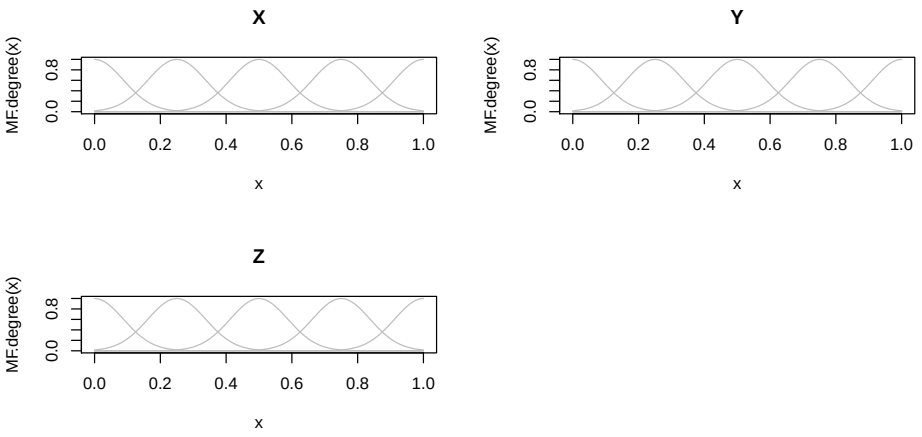


Figure 2.5: The plot of membership functions in the regression example.

The final step is to predict testing data using the *predict()* function. It needs two arguments which are *mod.reg* and new data. It can be done as follows.

```
R> res.test <- predict(mod.reg, data.tst)
```

The predicted values are generated in matrix format. They can be compared with the real values using the mean square error (MSE) by

```
R> err.MSE <- mean((real.val-res.test)^2)
R> print(err.MSE)
```

```
[1] 0.07852261
```

2.3.3 Classification Problem

In this example, we are using the *iris* data set which is already included in the R environment. The *iris* data set is a well-known data set in the pattern recognition literature. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are not linearly separable from each other. To use it, we just load the data by the command:

```
R> data(iris)
```

To get a relatively good proportion, usually we randomize the data by

```
R> set.seed(2)
R> irisShuffled <- iris[sample(nrow(iris)), ]
```

Because the decision attribute, which is in the last column, and is represented with a string, we need to convert it into numerical values. Then, the data are split into two parts which are *tra.iris* for training data and *tst.iris* for testing ones.

```
R> irisShuffled[,5] <- unclass(irisShuffled[, 5])
R> tra.iris <- irisShuffled[1:105,]
R> tst.iris <- irisShuffled[106:nrow(irisShuffled)
+    , 1:4]
R> real.iris <- matrix(irisShuffled
+    [106:nrow(irisShuffled), 5], ncol = 1)
```

Then, even though "frbs" by default calculates the range of the input data, we strongly recommend to define it manually.

```
R> range.data.input <- apply(iris[, -ncol(iris)],
+    2, range)
```

It should be noted that for classification tasks we only need to define the range of input data.

As in the regression example, after our data is ready to use, we need to define some parameters concerning the used method and its *control* parameter. For example, we are going to use the FRBCS.CHI method and we define three linguistic values, the trapezoid membership function, and minimum and Zadeh for the types of t-norm, and implicator operators, respectively.

```
R> method.type <- "FRBCS.CHI"
R> control <- list(num.labels = 3,
+    type.mf = "TRAPEZOID", type.tnorm = "MIN",
+    type.implication.func = "ZADEH")
```

We generate an FRBS model through the following command.

```
R> mod.class <- frbs.learn(tra.iris,
+    range.data.input, method.type, control)
```

As in the regression example, we do prediction as follows:

```
R> res.test <- predict(mod.class, tst.iris)
```

Then, we can check the result by calculating the percentage error:

```
R> err = 100*sum(real.iris!=res.test)/
+    nrow(real.iris)
R> print(err)

[1] 4.444444
```

The plot of membership functions can be seen in Figure 2.6. Further information and examples can be found in our project web site at `http://dicits.ugr.es/software/FRBS/`.

2.3.4 Human Expert Constructions

There are two manners to construct an FRBS model as follows:

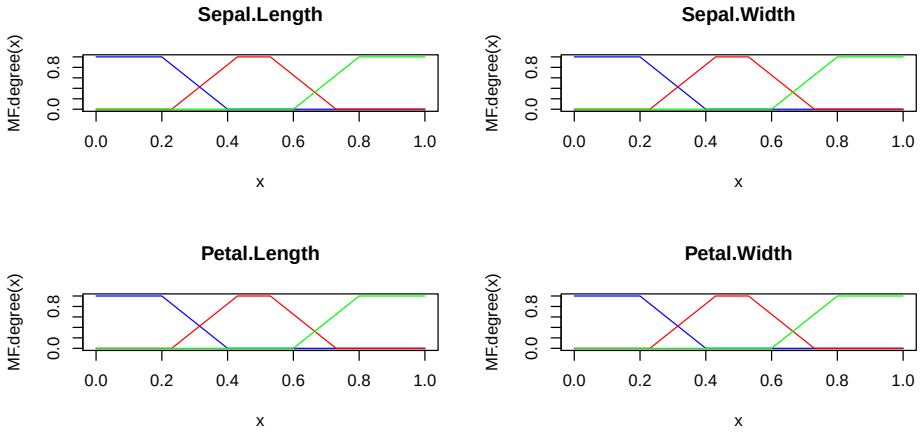


Figure 2.6: The plot of membership functions in the classification example.

- by calling *frbs.gen()*. It is a function for building the *frbs* object that contains database, rulebase, and other parameters. After generating the object, we execute *predict()* for prediction new data.
- by executing all involved functions: *rulebase()*, *fuzzifier()*, *inference()*, and *defuzzifier()*, separately.

User is allowed to construct the Mamdani, TSK, and FRBCS models.

In this example, we only show the second way, so users interested in another way should take a look at [244].

First, we need to construct a set of rules in *matrix*, e.g. as follow:

```
R> rule <- matrix(
+   c("a1", "and", "b1", "and", "c1", "and", "d1", "->", "e1",
+   "a2", "and", "b2", "and", "c2", "and", "d2", "->", "e2",
+   "a3", "and", "b2", "and", "c2", "and", "d1", "->", "e3"),
+   nrow = 3, byrow = TRUE)
```

Then, we check and validate the rules with

```
R> rule <- rulebase(type.model = "MAMDANI", rule, func.tsk = NULL)
```

It can be seen that we construct a Mamdani model.

After that, we define database that contains fuzzy definitions and membership functions. For example, we consider the trapezoid function as the membership function as follows:

```
R> varinp.mf <- matrix(c(2, 0, 20, 40, NA,
+     4, 20, 40, 60, 80, 3, 60, 80, 100, NA,
+     2, 0, 35, 75, NA, 3, 35, 75, 100, NA,
+     2, 0, 20, 40, NA, 1, 20, 50, 80, NA, 3, 60, 80, 100, NA,
+     2, 0, 20, 40, NA, 4, 20, 40, 60, 80, 3, 60, 80, 100, NA),
+     nrow = 5, byrow = FALSE)
```

with the following fuzzy definitions:

```
R> num.fvalinput <- matrix(c(3, 2, 3, 3), nrow=1)
```

It means that we have four input variables: e.g., "A", "B", "C", and "D", where these variables have 3, 2, 3, and 3 fuzzy terms, repectively, as follows:

```
R> A <- c("a1", "a2", "a3")
R> B <- c("b1", "b2")
R> C <- c("c1", "c2", "c3")
R> D <- c("d1", "d2", "d3")
R> names.varinput <- c(A, B, C, D)
```

Moreover, we set the following range:

```
R> range.data <- matrix(c(0,100, 0, 100, 0, 100, 0, 100, 0, 100),
   nrow = 2)
```

Furthermore, since we consider the Mamdani model, we need to define membership functions for the output variable as well, as follows:

```
R> varout.mf <- matrix(c(2, 0, 20, 40, NA,
+    4, 20, 40, 60, 80, 3, 60, 80, 100, NA),
+    nrow = 5, byrow = FALSE)
R> names.varoutput <- c("e1", "e2", "e3")
R> num.fvaloutput <- matrix(c(3), nrow = 1)
```

After that, we execute the following function for converting crisp values of data into fuzzy ones:

```
R> newdata <- matrix(c(15, 80, 85, 85, 45, 75, 78, 70), nrow = 2,
     byrow = TRUE)
R> num.varinput <- ncol(num.fvalinput)
R> MF <- fuzzifier(newdata, num.varinput, num.fvalinput, varinp.mf)
```

It should be noted that *newdata* is a dataset for the testing step. For constructing an FRBS model by human experts, we do not need any training data.

After converting into fuzzy values, we calculate the confidence factor on each antecendet rule by calling the following function:

```
R> miu.rule <- inference(MF, rule, names.varinput,
+     type.tnorm = "MIN", type.snorm = "MAX")
```

It can be seen here that we assign *"MIN"* and *"MAX"* to *type.tnorm* and *type.snorm*.

Finally, we convert back fuzzy values to crisp values by running the defuzzification, as follows:

```
R> range.output <- range.data[, ncol(range.data), drop = FALSE]
R> result <- defuzzifier(newdata, rule, range.output, names.varoutput,
+    varout.mf, miu.rule, type.defuz = "WAM",
+    type.model = "MAMDANI", func.tsk = NULL)
```

We see the result by

```
R> print(result)
```

```
     [,1]
[1,]   50
[2,]   40
```

2.4 Experimental Studies

This section presents an experimental comparison of the "frbs" package with other packages available in CRAN. The goal of this comparison is basically to illustrate that the performance of the implementation of FRBSs done on the "frbs" package is competitive to other approaches. The comparison includes both regression and classification problems. In regression, the response or output variable is numerical/continuous, whereas in classification the output is a category. We perform experiments using several datasets to evaluate the methods.

2.4.1 Regression Tasks

In the following, we describe the experiment design for regression, which includes the datasets, the methods considered for comparison, their parameters, and finally the experimental results.

Datasets

In this task, we consider two time series, which are the gas furnace dataset [32] and the Mackey-Glass series [181]. Both datasets have been included in the package. Originally, the first dataset has two attributes which are methane gas and the percentage of carbon dioxide inside the gas furnace. However, in this experiment we arrange the dataset as follows. As input variables, we use 292 consecutive values of methane at time $(t-4)$ and the CO_2 at time $(t-1)$, with the produced CO_2 at time (t) as an output variable. In other words, each training data point consists of $[u(t-4), y(t-1), y(t)]$, where u is methane and y is CO_2. Then, we divide the data into two groups: 70% of the data are used for training and the rest of the data is used for testing.

Secondly, The Mackey-Glass chaotic time series is defined by the following delayed differential equation:

$$\frac{dx(t)}{dt} = \frac{(\alpha \times x(t-\tau))}{(1+x(t-\tau)^{10})} - \beta \times x(t)$$

Using the above equation, we generate 1000 samples, with input parameters as follows: $\alpha = 0.2$, $\beta = 0.1$, $\tau = 17$, $x_0 = 1.2$, $dt = 1$. The dataset is

embedded in the following way: input variables: $x(t-18)$, $x(t-12)$, $x(t-6)$, $x(t)$ and output variable: $x(t+6)$. After that, we split the data into two equally sized datasets (i.e., training and testing data).

Methods considered for comparison and their parameters

The following R packages are used to compare them to the "frbs" package. The selection of packages is certainly not exhaustive, however, it is a representative set of well-known standard methods, which furthermore have different characteristics and implement different approaches to deal with the regression problems.

- "randomForest" [34]: This package implements the random forests method, which combines various decision trees to predict or classify the values. The method is suitable for both classification and regression tasks.
- "RSNNS" [27]: This package implements many standard procedures of neural networks. Here, we use the multi-layer perceptron (*mlp*) for regression.
- "fRegression" [303]: This package implements various methods for regression tasks. We use three methods from the package: linear regression model (*lm*), generalized linear modeling (*glm*), and projection pursuit regression (*ppr*).
- "nnet" [242]: This package is the standard/recommended package for neural networks in R, so that it is available directly from within R. It implements a multi-layer perceptron with one hidden layer (*nnet*), and uses a general quasi-Newton optimization procedure (the BFGS algorithm) for learning.
- "CORElearn" [246]: This package contains several learning techniques for classification and regression. We use the regression tree method (*regTree*) of the package here.
- "e1071" [190]: From this package, we use the available support vector machine, *svm*, to perform regression tasks.

The parameters of the methods considered in the experiments are shown in Table 2.2. We use the same parameter specifications for the two datasets.

Table 2.2: Parameters of the methods selected for comparison for regression.

Methods	Parameters
randomForest	importance = TRUE, proximity = TRUE
mlp	size = 5, learnFuncParams = **[0,1]**, maxit = 350
lm	none
glm	none
ppr	none
nnet	size = 30, linout = TRUE, maxit = 1000
regTree	none
svm	cost = 10, gamma = 0.01
ANFIS	num.labels = 5, max.iter = 300, step.size = 0.01, type.mf = 3
HYFIS	num.labels = 5, max.iter = 200, step.size = 0.01
SBC	r.a = 0.3, eps.high = 0.5, eps.low = 0.15
DENFIS	Dthr = 0.15, max.iter = 5000, step.size = 0.01, d = 2
FIR.DM	num.labels = 5, max.iter = 1000, step.size = 0.01
FS.HGD	num.labels = 5, max.iter = 100, step.size = 0.01, alpha.heuristic = 1
GFS.THRIFT	popu.size = 30, num.labels = 5, persen_cross = 0.9, persen_mutant = 0.3, max.gen = 100
GFS.FR.MOGUL	persen_cross = 0.9, max.iter = 300, max.gen = 200, max.tune = 500, persen_mutant = 0.3, epsilon = 0.95
"WM"	num.labels = 15, type.mf = 3, type.defuz = 1, type.tnorm = 1, type.snorm = 1

Table 2.3: Results obtained in the regression tasks.

Methods	G. Furnace (RMSE)	M.-Glass (RMSE)	Methods	G. Furnace (RMSE)	M.-Glass (RMSE)
randomForest	0.91	0.016	*SBC*	0.72	0.022
mlp	0.86	0.011	*DENFIS*	0.89	0.101
lm	0.72	0.094	*FIR.DM*	1.23	0.234
glm	0.72	0.094	*FS.HGD*	0.83	0.052
ppr	0.64	0.050	*GFS.THRIFT*	1.64	0.225
nnet	**0.58**	**0.002**	*GFS.FR.MOGUL*	1.08	0.084
regTree	1.41	0.062	*WM*	0.78	0.019
svm	0.72	0.033	*HYFIS*	0.87	0.087
			ANFIS	0.64	0.032

Experimental results

This section presents the results of the methods considered for comparison. To evaluate the results, we calculate the RMSE. The complete results are shown in Table 2.3. It can be seen that the best result for the gas furnace dataset are an RMSE of 0.58, obtained by the *nnet* method. For the Mackey-Glass series, the best method is *nnet*, with an RMSE of 0.002. Based on this benchmarking experiment, we can say that the methods that provide the best three results for the gas furnace dataset are *nnet*, *ANFIS*, and *ppr*. One of these methods is from the "frbs" package. In the case of the Mackey-Glass series, methods from other packages like *nnet*, *mlp*, and *randomForest* outperform the methods included in package "frbs." Generally, the methods included in package "frbs" obtain reasonable, competitive results.

2.4.2 Classification Tasks

In this section an illustrative empirical study of FRBS methods in classification is provided. We describe again the experiment design, which includes the datasets, the methods considered for comparison, their parameters, and finally the experimental results.

Table 2.4: Datasets considered for classification tasks.

Name	Attributes	Patterns	Classes
iris	4	150	3
pima	8	768	2
wine	13	178	3

Datasets

In these experiments, we consider three datasets, namely, the iris, pima, and wine datasets. Some properties of the datasets can be seen in Table 2.4. To validate the experiments, we consider a 5-fold cross-validation, i.e., we randomly split the datasets into five folds, each containing 20% of the patterns of the dataset. Then, we use four partitions for training and one partition for testing. All of the data are available from the KEEL-dataset repository [10].

Methods considered for comparison and their parameters

Again, we compare the classification methods in the "frbs" package with several different packages available on CRAN. The methods are chosen since they are well-known methods and represent different characteristics in the way they solve the particular tasks. The packages used for comparison are the following ones:

- "CORElearn" [246]: As mentioned before, this package contains several methods. In this case, we use the k-nearest-neighbors classifier method (*knn*).
- "C50" [163]: The package implements the *C5.0* algorithm presented by [232].
- "randomForest" [34]: The *randomForest* can be used both for regression and for classification, so that we use it also here.
- "nnet" [242]: We use *nnet* as in the regression task.
- "RSNNS" [27]: As in the regression task, we use *mlp* for classification.

- "tree" [243]: The package implements the *tree* method.
- "kernlab" [149]: The package implements SVM methods. In this experiment, we use the *ksvm* function to perform classification tasks.
- "fugeR" [38]: The package implements the *fugeR* method which is a genetic algorithm to construct an FRBS model. We consider this package for comparison because it is a package already available from CRAN that applies FRBSs.

The parameters of the methods used in the experiments are shown in Table 2.5. The same parameter specifications were used for all the datasets in classification.

Experimental results

Table 2.6 shows the results obtained from the three experiments using 5-fold cross-validation. By considering all datasets, in these experiments the best results are obtained by *FRBCS.CHI*, *tree*, and *ksvm* for iris, pima, and wine, respectively. So, we see that the methods available in the "frbs" package can be considered competitive for classification tasks.

2.5 A Comparison with Other Software Libraries

In this section, we develop two comparisons: a comparison "frbs" with other packages included in CRAN and a comparison "frbs" with other software libraries.

2.5.1 Other FRBS Packages Available in CRAN

Here, we review in more detail the packages available in CRAN which implement FRBSs. We compare them to "frbs," considering functionality and capability. The following packages provide functions which are able to construct FRBSs (i.e., "sets" and "fugeR"):

"sets" As already stated briefly, "sets" [191] provides standard procedures for the construction of sets, fuzzy sets, and multisets. Especially w.r.t. fuzzy

Table 2.5: Parameters of the methods selected for comparison for classification.

Methods	Parameters
knn	none
C5.0	trial = 100
randomForest	importance = TRUE, proximity = TRUE
nnet	size = 5, rang = 0.8, decay = 5e-4, maxit = 1000
mlp	maxit = 350, learnFuncParams = **[0, 1]**, size = 5
tree	none
ksvm	type = "C-bsvc", kernel = "rbfdot", kpar = list(sigma = 0.1), C = 10, prob.model = TRUE
fugeR	generation = 100, population = 100, elitism = 20, verbose = TRUE, threshold = NA, sensiW = 0.0, speciW = 0.0, accuW = 0.0, rmseW = 1.0, maxRules = 10, maxVarPerRule = 2, labelsmf = 3
FRBCS.CHI	num.labels = 9, type.mf = 3
FRBCS.W	num.labels = 9, type.mf = 3
GFS.GCCL	popu.size = 70, num.labels = 3, persen_cross = 0.9, max.gen = 100, persen_mutant = 0.3
FH.GBML	popu.size = 50, max.num.rule = 100, persen_cross = 0.9, max.gen = 100, persen_mutant = 0.3, p.dcare = 0.8, p.michigan = 1
SLAVE	num.labels = 5, persen_cross = 0.9, max.iter = 100, max.gen = 100, persen_mutant = 0.3, k.low = 0, k.upper = 1, epsilon = 0.7

Table 2.6: Results obtained in the classification experiments.

Methods	Classification rate (%)		
	iris	pima	wine
knn	94.67	74.09	96.62
C5.0	94.00	74.34	94.35
randomForest	95.33	76.56	96.61
nnet	95.33	65.50	93.19
mlp	94.00	73.43	97.18
tree	94.67	**76.57**	92.67
ksvm	96.00	76.56	**98.29**
fugeR	95.33	76.09	89.31
FRBCS.CHI	**97.34**	67.44	92.67
FRBCS.W	96.00	69.92	92.67
GFS.GCCL	94.00	66.54	84.91
FH.GBML	95.34	68.62	81.93
SLAVE	97.33	72.91	88.17

sets, an advantage of "sets" is that it does not only rely on the R built-in *match()* function to perform set operations, but it also provides comprehensive operations such as negation, conjunction, disjunction, implication, etc. For example, the conjunction operator, *.T.()*, provides many options such as: "Zadeh," "drastic," "product," "Lukasiewicz," "Fodor," "Hamacher," "Yager," etc. Furthermore, there are several functions to set the shape of the membership function which are *fuzzy_normal()* for the Gaussian function, *fuzzy_trapezoid()* for trapezoid, *fuzzy_triangular()* for a triangle shape, etc. Regarding the construction of FRBSs, "sets" has the capability to perform fuzzy reasoning by using *fuzzy_inference()*, and to convert fuzzy into crisp values by using *gset_defuzzify()*. However, the package does not include learning algorithms, which is the main focus of our package. So, at first sight "sets" may seem an ideal base for the implementation of the functionality available in our package. But there is only the Mamdani model available, and we found it difficult to extend the "sets" package to our needs, as the underlying data types and syntactics do not facilitate automatization of the construction process of FRBSs[2]. So, finally we opted for simple numerical matrices as the basic data type in the "frbs" package. In "frbs," we provide many different learning procedures to learn from numerical data, as well as a mechanism for fuzzy reasoning without learning, by using our function *frbs.gen()*. Furthermore, "frbs" does not only implement the Mamdani model but it also has the TSK and FRBCS models implemented.

"fugeR" The package "fugeR" [38] implements genetic algorithms to construct an FRBS from numerical data for classification. It is based on fuzzy cooperative coevolution [217] where two coevolving species are defined: the databases and the rule base. In this package, there are two main functions which are *fugeR.run()* for construction of the FRBS model and *fugeR.predict()* for prediction. So, "fugeR" implements one particular classification method based on genetic algorithms. Our package implements the same workflow, but with more than ten different models, both for classification and regression, among them various different ones which use genetic algorithms.

In summary, while "sets" focuses on constructing FRBS models by human experts and providing the set representation and "fugeR" is used for learning from data based on genetic algorithms, "frbs" offers the implementations of

[2]Actually we tried pretty hard but did not find a way to get the parameters to *fuzzy_inference()* evaluated, as they are passed to *substitute* internally by that function.

several well-known machine-learning methods for learning from data and the construction of FRBS models by human experts manually.

2.5.2 Other Fuzzy Tools

In this section, we review well-known software libraries implementing FRBS concepts: Xfuzzy [21], Fuzzy Logic Toolbox for *MATLAB* [277], Fuzzy Inference System Professional (FisPro) [105], Generating Understandable and Accurate Fuzzy Models in a Java Environment (GUAJE) [12], and Knowledge Extraction based on Evolutionary Learning (KEEL) [11, 10]. All of them support learning from data using various learning procedures.

Xfuzzy is an open-source framework released under the term of the GPL License implementing fuzzy inference-based systems [21]. The software has an architecture containing several parts which share the proposed language *XFL3*. Using the graphical user interface, these parts have different functionalities, such as *xfedit* which can be used to describe the logical structure needed for the inference process. It is quite similar to the *frbs.gen()* function in "frbs" to perform inference based on knowledge constructed manually by human experts. *xfsl* is a tool used to extract knowledge from data. Therefore, it is obvious that *xfsl* has functionalities similar to *frbs.learn()* in "frbs." Some learning methods have been considered in this part, such as gradient descent, second-order, Gauss-Newton, and statistical algorithms. It can be seen that "frbs" offers more algorithms for generating FRBS models. For the inference process, Xfuzzy provides the *xfmt* tool which is the same as *predict()* in "frbs." Other capabilities of Xfuzzy are, e.g., plotting membership functions and converting codes using *xfc*, *xfcpp*, and *xfj* into *XFL3*. Furthermore, many options for t-norm, s-norm, and implicator operators and defuzzification methods are available as well.

The Fuzzy Logic Toolbox for *MATLAB* is a toolkit for analysis, design, and simulation systems based on fuzzy logic. It provides Simulink, a graphical user interface (GUI), and a command line mode to build FRBS models [277]. It supports standard Mamdani and Sugeno-type fuzzy inference systems. In order to design an FRBS model, it allows to use ANFIS, substractive clustering, and fuzzy C-means. Two of the three methods are provided in "frbs," and we provide several others. Additionally, in this toolbox, the *FISEditor* is used to display general information about a fuzzy inference system while the *Membership Function Editor* and *Rule Editor* are used to provide functions to display and edit membership function and rules. Since

"frbs" works with a scripting interface, editing is a straightforward process by changing the matrix of the model.

FisPro, built in C++ and Java, implements fuzzy inference systems considering the rule base interpretability and modularity in an open source software. As "frbs," it provides two modes of FRBS models which are expert rule design and automatic induction for regression and classification cases. It implements k-means, hierarchical fuzzy partitioning (HFP), Wang and Mendel (WM), fast prototyping algorithm (FPA), and fuzzy decision trees (FDT) in order to generate fuzzy partitions and rule bases. Additionally, FisPro provides the aggregation operators "MAX" and "SUM" for conjunction, which are also available in "frbs." FisPro also includes mechanisms for merging and improving fuzzy rules in a separated part which is an optimization modul. In contrast, "frbs" simultaneously performs optimization with along learning processes by using learning methods such as genetic fuzzy systems.

GUAJE is an open-source software that implements fuzzy rule-based systems in Java. It is an extension of the Knowledge Base Configuration Tool (KBCT) [13] and aimed to provide interpretable fuzzy systems. Additionally, it integrates several other software programs, such as FisPro, ORE, Espresso, Graphviz, JMetal, and WEKA. From the perspective of the algorithms used to generate fuzzy rules, GUAJE adopts the approaches implemented in FisPro. Also, the generated FRBS models can be exported to FisPro, Xfuzzy, and the Fuzzy Logic Toolbox for *MATLAB*. Although GUAJE provides data pre-processing algorithms (e.g., feature selection) and implements other approaches, "frbs" has the advantage of being built in the R environment which offers additional benefits since R provides more comprehensive and complete algorithms for data pre-processing and other prediction methods.

KEEL is a big and comprehensive software library containing classical knowledge extraction algorithms, preprocessing techniques (e.g., instance selection, feature selection, discretization, etc.), learning algorithms for clustering, regression, and classification problems, and a statistical test module for comparison [11, 10]. It provides three important blocks: data management, design of experiments, and educational experiments. Therefore, it can be seen that KEEL is intended as a research and educational tool. From the perspective of FRBSs, it is focused on implementation of learning methods based on GFS, such as GFS based on Thrift's algorithm [278], SLAVE [91], etc. Some of the algorithms considered in KEEL are implemented in "frbs"

as well. Even though KEEL has implemented more than thirty learning algorithms based on FRBSs, it does not provide construction of FRBS models from human experts as in "frbs."

According to the comparison, it can be seen that "frbs" has some advantages that are not available in other tools separately. For example, "frbs" provides the capabilities of learning from data and constructing by human experts, the complete FRBS models, and the available parameters for building the models.

2.6 Summary

This chapter presents the implementation of the "frbs" package that fulfills Objective 1(a). The package offers the following functionalities:

1. It includes implementations of three well-known FRBS models: Mamdani, TSK, and FRBCS, that are used for dealing with classification and regression.

2. There are over 10 machine-learning methods employing genetic algorithms, artificial neural networks, gradient descent, clustering, and space partitions.

3. It allows constructing FRBS models by human experts by using the following functions: *frbs.gen()*, *fuzzifier()*, *rulebase()*, *inference()*, and *defuzzifier()*.

4. Plotting membership functions and summarizing fuzzy rules.

5. Wide diversity in choices for different components of triangular norm (t-norm), s-norm, implicator functions, defuzzification methods, membership functions, and linguistic hedges.

6. Free and open source software under the GPL License.

7. Many demos and datasets embedded in the package with the structured and complete documentation/manual.

Moreover, some comparisons with other tools show that "frbs" should be considered as a software system unifying various advantages of the separately

available tools. In this chapter, we also provide some example showing how to use the package.

The "frbs" package is available in CRAN: `http://cran.r-project.org/package=frbs` and in the project website: `http://sci2s.ugr.es/dicits/software/FRBS`. Additionally, the journal paper describing the package is published:

> 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, Vol. 65(6), p. 1–30, 2015, `http://www.jstatsoft.org/v65/i06/`.

Chapter 3

frbsPMML: A Universal Representation Framework for FRBSs Based on PMML

This chapter associated Objective 2, namely designing a universal representation framework of FRBS models based on PMML. The representation called frbsPMML will be introduced in Section 3.1. Moreover, the frbsPMML specifications and their implementations are presented before explaining detailed features and benefits. Then some examples of usage are provided. Lastly, we provide a summary of the chapter.

3.1 Introduction

Nowadays, many software systems implementing FRBSs are available for both academic and industry purposes. For example, "Xfuzzy" is an open-source framework based on fuzzy inference-based systems [21]. To represent an FRBS model, it uses a formal language called the Xfuzzy 3.0 specification language (XFL3). XFL3 contains declarations about membership functions, a set of rules, and other parameters. In the MATLAB environment, there is the Fuzzy Logic Toolbox [277]. It is developed by utilizing Simulink, which is a graphical user interface (GUI) used for data flow, and a command-line

mode to build an FRBS model saved in the so-called *.fis* file format. Additionally, apart from these most relevant software systems, there are others, e.g., "FisPro" [105], "GUAJE" [12], and "KEEL" [11, 10]. Though available software libraries provide many useful features for tackling real-world problems, we note that there is not a standard interface that connects between them, so that it is difficult to exchange models between the different software systems. As interoperability is an important issue not only in industry cases but also for academic purposes, this is a shortcoming that we address with our work.

In this work, we overcome this shortcoming by designing and implementing a proposal for a universal representation framework of FRBSs for PMML, called frbsPMML. By developing an extension of PMML, an FRBS model can be easily read, checked, verified, deployed in most computing platforms, and even modified by human experts because the model is stored in an XML text file. Additionally, in this research we present an extension of the "frbs" package to produce and consume an FRBS model in PMML format. Another implementation, written in Java, is presented as well. It is called "frbsJpmml," and can be used to deploy PMML models and perform predictions on new, unknown data.

As mentioned before, a universal representation framework naturally offers the advantages of interoperability and reproducible research. Moreover, two essential aspects considered for measuring the performance in FRBSs are accuracy and interpretability. Using FRBSs in the PMML format, we gain a benefit which is high levels of interpretability. In other words, because of the XML-based language, an FRBS model becomes readable by human and machine. Therefore, human experts can easily check, verify, and modify the model. Additionally, from the FRBS point of view, interpretability mainly refers to the capability of the fuzzy model to express the behaviour of system in a understandable way, which depends on several aspects: the model structure, the number of input variables, the number of fuzzy rules, the number of linguistic terms, and the shape of the fuzzy sets [42]. FRBSs in the PMML format allow to represent a model in accordance with these criteria. It happens since a database and a rulebase are specified by the XML-based language in flexible way.

Three models, which can be used for handling regression and classification tasks, are specified by the proposed representations: Mamdani, TSK, and FRBCS. A key advantage of FRBS model specification in frbsPMML is that high degrees of transparency and interpretability can be achieved so

that human experts can easily understand, verify, modify, and communicate the models. So, an easier deployment and integration of FRBSs with other tools for modelling and data analysis becomes possible, as well as easier reproducibility of research. The new representation can be considered an open standard for representing FRBS models.

3.2 Specifications of frbsPMML

In this section, we describe its basic elements, the XML schemata for specifying an FRBS model in PMML format. Based on Listing 1.1, the extension is made in the *MODEL-ELEMENT* part, while other components are still based on the existing PMML schema. Therefore, we only discuss on the new components of the extension.

The general schema specifying an FRBS model is described in Listing 3.1.

```
<xs:element name="FrbsModel">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0"
          maxOccurs="unbounded"/>
      <xs:element ref="MiningSchema"/>
      <xs:element ref="Output" minOccurs="0"/>
      <xs:element ref="InferenceSchema"/>
      <xs:element ref="Database"/>
      <xs:element ref="Rulebase"/>
      <xs:element ref="ModelVerification" minOccurs="0"/>
      <xs:element ref="Extension" minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="modelName" type="xs:string"
        use="required"/>
    <xs:attribute name="functionName" type="MINING-FUNCTION"
        use="optional"/>
    <xs:attribute name="algorithmName" type="xs:string"
        use="optional"/>
    <xs:attribute name="targetFieldName" type="xs:string"
        default="optional"/>
```

```
  </xs:complexType>
</xs:element>
```

Listing 3.1: XML Schema for FRBS models.

It can be seen that the *FrbsModel* tag is required for representing an FRBS model. In the *FrbsModel*, there are two types of components: attribute and element. We define four attributes: *modelName*, *functionName*, *algorithmName*, and *targetFieldName*, where only *modelName* is required to be set. The *modelName* attribute refers to the type of FRBS model, i.e., MAMDANI, TSK, and FRBCS, for representing the Mamdani, TSK, and FRBCS model, respectively. In the elements, three components are important and emphasized, as follows: *InferenceSchema*, *Database*, and *Rulebase*.

InferenceSchema is a schema representing essential parameters in an FRBS model for inference/reasoning: conjunction, disjunction, implication, and aggregation operators. For example, the conjunction operators can be any of the following functions: *MIN*, *PRODUCT*, *HAMACHER*, *YAGER*, and *BOUNDED*. It should be noted that the parameters are defined as an optional components depending on the models. For instance, we need to set the *AggregationOperator* value if we use the Mamdani model. The schema of *InferenceSchema* can be seen in Listing 3.2.

```
<xs:element name="InferenceSchema">
  <xs:complexType>
    <xs:element ref="ConjunctionOperator" use="optional"/>
    <xs:element ref="DisjunctionOperator" use="optional"/>
    <xs:element ref="ImplicationOperator" use="optional"/>
    <xs:element ref="AggregationOperator" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="ConjunctionOperator">
  <xs:attribute name="value">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="MIN"/>
        <xs:enumeration value="PRODUCT"/>
        <xs:enumeration value="HAMACHER"/>
        <xs:enumeration value="YAGER"/>
        <xs:enumeration value="BOUNDED"/>
```

```
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>

<xs:element name="DisjunctionOperator">
  <xs:attribute name="value">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="MAX"/>
        <xs:enumeration value="SUM"/>
        <xs:enumeration value="HAMACHER"/>
        <xs:enumeration value="YAGER"/>
        <xs:enumeration value="BOUNDED"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>

<xs:element name="ImplicationOperator">
  <xs:attribute name="value">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="DIENES_RESHER"/>
        <xs:enumeration value="LUKASIEWICZ"/>
        <xs:enumeration value="ZADEH"/>
        <xs:enumeration value="GOGUEN"/>
        <xs:enumeration value="SHARP"/>
        <xs:enumeration value="MIZUMOTO"/>
        <xs:enumeration value="DUBOIS_PRADE"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>

<xs:element name="AggregationOperator">
  <xs:attribute name="value">
```

```
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="WAM"/>
        <xs:enumeration value="FIRST.MAX"/>
        <xs:enumeration value="LAST.MAX"/>
        <xs:enumeration value="MEAN.MAX"/>
        <xs:enumeration value="COG"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>
```

Listing 3.2: XML Schema for the *InferenceSchema* component.

The database is represented by the *Database* element, and it contains the following information:

- names of variables including the number of their linguistic values,
- types of membership functions, such as Gaussian, trapezoid, and triangular memberships,
- parameters of membership functions. For example, in the Gaussian membership function, the parameters are mean and variance.

The XML Schema of the *Database* is described in Listing 3.3 and Listing 3.4. Basically, the *Database* contains *MembershipFunction* involving the element *FuzzyTerm* and two attributes: *name* and *numberOfLabels*. While *FuzzyTerm* represents databases containing linguistic values and their parameters; *name* and *numberOfLabels* express the variable name and the number of linguistic terms corresponding to each variable.

```
<xs:element name="Database">
  <xs:complexType>
    <xs:element ref="MembershipFunction"/>
  </xs:complexType>
</xs:element>

<xs:element name="MembershipFunction">
  <xs:complexType>
    <xs:sequence>
```

```
      <xs:element ref="FuzzyTerm"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="numberOfLabels" type="INT-NUMBER"
        use="required"/>
  </xs:complexType>
</xs:element>
```

Listing 3.3: Part 1: XML Schema for the *Database* component.

```
<xs:element name="FuzzyTerm">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="type" use="required">
      <xs:complexType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="GAUSSIAN">
            <xs:element name="Parameters">
              <xs:element name="Mean" type="REAL-NUMBER"/>
              <xs:element name="Variance" type="REAL-NUMBER"/>
            </xs:element>
          </xs:enumeration>
          <xs:enumeration value="TRAPEZOID">
            <xs:element name="Parameters">
              <xs:element name="Left" type="REAL-NUMBER"/>
              <xs:element name="LeftMiddle" type="REAL-NUMBER"/>
              <xs:element name="RightMiddle" type="REAL-NUMBER"/>
              <xs:element name="Right" type="REAL-NUMBER"/>
            </xs:element>
          </xs:enumeration>
          <xs:enumeration value="TRIANGLE">
            <xs:element name="Parameters">
              <xs:element name="Left" type="REAL-NUMBER"/>
              <xs:element name="Middle" type="REAL-NUMBER"/>
              <xs:element name="Right" type="REAL-NUMBER"/>
            </xs:element>
          </xs:enumeration>
          <xs:enumeration value="SIGMOID">
```

```
            <xs:element name="Parameters">
              <xs:element name="Gamma" type="REAL-NUMBER"/>
              <xs:element name="Distance" type="REAL-NUMBER"/>
            </xs:element>
          </xs:enumeration>
          <xs:enumeration value="BELL">
            <xs:element name="Parameters">
              <xs:element name="Width" type="REAL-NUMBER"/>
              <xs:element name="Power" type="REAL-NUMBER"/>
              <xs:element name="Center" type="REAL-NUMBER"/>
            </xs:element>
          </xs:enumeration>
        </xs:restriction>
      </xs:complexType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

Listing 3.4: Part 2: XML Schema for the *Database* component.

According to the *FuzzyTerm* schema, we provide five types of membership functions:

1. *GAUSSIAN*: In this case, we need to define two elements: *Mean* and *Variance* representing mean and variance of the Gaussian function.
2. *TRAPEZOID*: We supply four components *Left*, *LeftMiddle*, *RightMiddle*, and *Right* for representing the corner points.
3. *TRIANGLE*: It has three parameters: *Left*, *Middle*, and *Right* that represent the corner points.
4. *SIGMOID*: There are two parameters: *Gamma* and *Distance*, representing steepness of the function, and distance from the origin, respectively.
5. *BELL*: Three parameters need to be defined in *BELL*: *Width*, *Power*, and *Center*, which determine the width of the curve, a positive number for the power, and the center of the curve.

Furthermore, it is possible to define different membership functions and numbers of linguistic values for the variables. We can also assign different

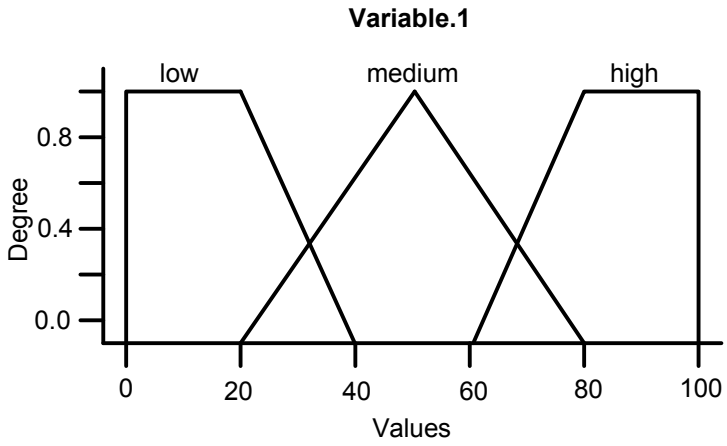


Figure 3.1: The membership functions of "Variable.1."

numbers of linguistic values for other variables. For instance, "Variable.1" has 3 linguistic values which are "low", "medium", and "high". To determine the degree we define that "medium" has *TRIANGLE* and the rest have *TRAPEZOID* memberships as in Figure 3.1. This example can be specified in frbsPMML format as in Listing 3.5.

```
<Database>
  <MembershipFunction name="Variable.1" numberOfLabels="3">
        <FuzzyTerm name="low" type="TRAPEZOID">
          <Parameters>
            <Left>0</Left>
            <LeftMiddle>0</LeftMiddle>
            <RightMiddle>20</RightMiddle>
            <Right>40</Right>
          </Parameters>
        </FuzzyTerm>
    <FuzzyTerm name="medium" type="TRIANGLE">
      <Parameters>
        <Left>20</Left>
        <Middle>50</Middle>
        <Right>80</Right>
      </Parameters>
    </FuzzyTerm>
    <FuzzyTerm name="high" type="TRAPEZOID">
```

```
      <Parameters>
        <Left>60</Left>
        <LeftMiddle>80</LeftMiddle>
        <RightMiddle>100</RightMiddle>
        <Right>100</Right>
      </Parameters>
    </FuzzyTerm>
  </MembershipFunction>
</Database>
```

Listing 3.5: The *Database* schema of "Variable.1."

Finally, Listing 3.6 describes the XML Schema of the *Rulebase* consisting of the element *Rule* and the attribute *numberOfRules*. *Rule* specifies a set of rules whereas *numberOfRules* shows the number of rules used for validation.

```
<xs:element name="Rulebase">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Rule" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="numberOfRules" type="INT-NUMBER" use="required"/>
  </xs:complexType>
</xs:element>
```

Listing 3.6: XML Schema for the *Rulebase* component.

As mentioned before, FRBS models can be classified into two popular models: Mamdani and TSK, used for dealing with regression problems. Additionally, FRBCS is suitable for classification tasks. Since the difference of models is determined by the representations of rules, we explain the components of the *Rulebase* in the following.

3.2.1 The Mamdani Model

This model was introduced by Mamdani in [183, 184]. It is built by linguistic variables in both the antecedent and consequent parts of the rules. So, considering multi-input and single-output (MISO) systems, fuzzy IF-

THEN rules are of the following form:

$$\textbf{IF } \boldsymbol{X}_1 \text{ is } \boldsymbol{A}_1 \text{ and } \ldots \text{ and } \boldsymbol{X}_n \text{ is } \boldsymbol{A}_n \quad \textbf{THEN } \boldsymbol{Y} \text{ is } \boldsymbol{B} \tag{3.1}$$

Here, $\boldsymbol{X}_i$ and $\boldsymbol{Y}$ are input and output linguistic variables, respectively, while $\boldsymbol{A}_i$ and $\boldsymbol{B}$ are linguistic values, e.g., "hot", "medium", and "cold".

Generally, a rule represented by Equation 3.1 can be specified by the XML Schema as in Listing 3.7. It contains two elements: *If* and *Then*, used for expressing the antecedent and consequence parts.

```
<xs:element name="Rule">
  <xs:complexType>
    <xs:element name="If">
      <xs:complexType>
        <xs:element ref="CompoundPredicate"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Then">
      <xs:complexType>
        <xs:element ref="SimplePredicate"/>
      </xs:complexType>
    </xs:element>
    <xs:attribute name="id" type="INT-NUMBER"
        use="optional"/>
  </xs:complexType>
</xs:element>
```

Listing 3.7: XML Schema for the *Rule* component based on the Mamdani model.

The *If* part includes the *CompoundPredicate* component, whose XML Schema is shown in Listing 3.8. Basically, *CompoundPredicate* consists of *SimplePredicate* together with the attribute *booleanOperator* to construct the antecedent part recursively. The *SimplePredicate* element is built from two components: *field* and *value*. The *field* attribute expresses a variable name whereas *value* is a linguistic value. The attribute *booleanOperator* expresses the logic operators (i.e., *and* and *or*). Furthermore, since we assume that the model is MISO, the *Then* part contains a single *SimplePredicate*.

```
<xs:element name="CompoundPredicate">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="SimplePredicate"/>
      <xs:element ref="CompoundPredicate"/>
    </xs:sequence>
    <xs:attribute name="booleanOperator" use="required">
      <xs:simpleType>
        <xs:enumeration value="and"/>
        <xs:enumeration value="or"/>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

<xs:element name="SimplePredicate">
  <xs:simpleType>
    <xs:attribute> name="field" type="xs:string" use="required"/>
    <xs:attribute> name="value" type="xs:string" use="required"/>
  </xs:simpleType>
</xs:element>
```

Listing 3.8: XML Schema for the *CompoundPredicate* and *SimplePredicate* components.

For example, the rule 3.2 is documented in frbsPMML as in Listing 3.9.

$$\textbf{IF } X1 \text{ is } normal \text{ and } X2 \text{ is } tall \text{ and } X3 \text{ is } small \quad \textbf{THEN } Y \text{ is } good \tag{3.2}$$

```
<Rulebase numberOfRules="1">
  <Rule id="1">
    <If>
      <CompoundPredicate booleanOperator="and">
        <SimplePredicate field="X1" value="normal"/>
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="X2" value="tall"/>
          <SimplePredicate field="X3" value="small"/>
        </CompoundPredicate>
```

```
      </CompoundPredicate>
    </If>
    <Then>
      <SimplePredicate field="Y" value="good"/>
    </Then>
  </Rule>
</Rulebase>
```

Listing 3.9: The *Rulebase* schema of the example based on the Mamdani model.

A main benefit of the schema is that it is rather flexible. For example, we can define different values of *booleanOperator* for each predicate. Additionally, this schema allows us to put the negation operator (i.e., *not*) and linguistic hedges (e.g., *very*, *somewhat*, etc.) in the *value* attribute. Furthermore, it is not necessary to involve all input variables for the construction of each rule. In other words, the length of *SimplePredicate* in each rule can be different. We can also set the *dont_care* value which represents a variable whose degree of membership is 1 for all conditions.

3.2.2 The TSK Model

The difference of the TSK model from the Mamdani model is on the consequent part. TSK uses rules whose consequent parts are represented by a function of input variables instead of using linguistic variables [273, 271]. The most commonly used function is a linear combination of the input variables: $Y = f(X_1, \ldots, X_n)$ where X_i and Y are the input and output variables, respectively. Therefore, we can express it as $Y = p_1 \cdot X_1 + \cdots + p_n \cdot X_n + p_0$ with a vector of real parameters $p = (p_0, p_1, \ldots, p_n)$.

We define the XML Schema of *Rule* based on TSK as in Listing 3.10.

```
<xs:element name="Rule">
  <xs:complexType>
    <xs:element name="If">
      <xs:complexType>
        <xs:element ref="CompoundPredicate"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Then">
```

```
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="Coefficient" minOccurs="0"/>
        </xs:sequence>
        <xs:element ref="Constant" minOccurs="1"/>
        <xs:attribute name="type" default="LinearFunction">
          <xs:simpleType>
            <xs:enumeration value="LinearFunction"/>
            <xs:enumeration value="NonLinearFunction"/>
          </xs:simpleType>
        </xs:attribute>
      </xs:complexType>
    </xs:element>
    <xs:attribute name="id" type="INT-NUMBER" use="optional"/>
  </xs:complexType>
</xs:element>
```

Listing 3.10: XML Schema for the *Rule* component based on the TSK model.

It can be seen that on the antecedent part (i.e., the *If* element) we have the same schema as in the Mamdani model, but in the *Then* block, we define two components: *Coefficient* and *Constant*. The XML Schema of both components is described in Listing 3.11. In order to construct a linear function, the attribute *Coefficient* represents coefficient values of each variable while *Constant* is the constant value of the equation. Using the specification, we allow to define first- and zero-order TSK.

```
<xs:element name="Coefficient">
  <xs:simpleType>
    <xs:attribute name="field" type="xs:string" use="required"/>
    <xs:attribute name="value" type="REAL-NUMBER" use="required"/>
  </xs.simpleType>
</xs:element>

<xs:element name="Constant">
  <xs:simpleType>
    <xs:attribute name="value" type="REAL-NUMBER"
        use="required">
  </xs:simpleType>
```

```
</xs:element>
```

Listing 3.11: XML Schema for the *Coefficient* and *Constant* components.

For example, a rule as in 3.3 can be specified in Listing 3.12.

$$\begin{aligned} &\textbf{IF } X1 \text{ is } normal \text{ and } X2 \text{ is } tall \text{ and } X3 \text{ is } small \\ &\textbf{THEN } Y = 0.2 \cdot X_1 + 0.1 \cdot X_2 - 0.2 \cdot X_3 + 0.9 \end{aligned} \tag{3.3}$$

```
<Rulebase numberOfRules="1">
  <Rule id="1">
    <If>
      <CompoundPredicate booleanOperator="and">
        <SimplePredicate field="X1" value="normal"/>
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="X2" value="tall"/>
          <SimplePredicate field="X3" value="small"/>
        </CompoundPredicate>
      </CompoundPredicate>
    </If>
    <Then type="LinearFunction">
      <Coefficient field="X1" value="0.2"/>
      <Coefficient field="X2" value="0.1"/>
      <Coefficient field="X3" value="-0.2"/>
      <Constant value="0.9"/>
    </Then>
  </Rule>
</Rulebase>
```

Listing 3.12: The *Rulebase* schema of the example based on the TSK model.

3.2.3 The FRBCS Model

A main characteristic of classification is that the outputs are categorical data. Therefore, in this model type we preserve the antecedent part of linguistic variables, and change the consequent part to be a class C_j from a prespecified class set $C = \{C_1, \ldots, C_M\}$. Generally, there are three structures for representing FRBCS. First, the simplest form introduced by [49] is

constructed with a class in the consequent part. Then, the FRBCS model with a certainty degree (called weight) in the consequent part is discussed in [129]. In [185], every fuzzy rule has with a certainty degree for all classes in the consequent part. In other words, instead of considering one class, this model provides prespecified classes with their respective weights for each rule. In this paper, we consider the second type.

Listing 3.13 shows the schema of *Rule* for FRBCS. We note that it is quite similar to the Mamdani model in Listing 3.7, but in the *Then* part we have categorical values instead of linguistic ones. Additionally, there is a component *Grade* representing a degree of the certainty of each rule that has a value between 0 and 1.

```
<xs:element name="Rule">
  <xs:complexType>
    <xs:element name="If">
      <xs:complexType>
        <xs:element ref="CompoundPredicate"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Then">
      <xs:complexType>
        <xs:element ref="SimplePredicate"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Grade" value="REAL-NUMBER"/>
    <xs:attribute name="id" type="INT-NUMBER" use="optional"/>
  </xs:complexType>
</xs:element>
```

Listing 3.13: XML Schema for the *Rule* component based on the FRBCS model.

For example, we define a rule as in 3.4, where w is its grade. In the frbsPMML format, it can be specified as in Listing 3.14.

$$\textbf{IF } \boldsymbol{X1} \text{ is } \boldsymbol{normal} \text{ and } \boldsymbol{X2} \text{ is } \boldsymbol{tall} \text{ and } \boldsymbol{X3} \text{ is } \boldsymbol{small} \quad \textbf{THEN } \boldsymbol{class} \text{ is } 1 \text{ with } w = 0.1. \tag{3.4}$$

```
<Rulebase numberOfRules="1">
```

```
  <Rule id="1">
    <If>
      <CompoundPredicate booleanOperator="and">
        <SimplePredicate field="X1" value="normal"/>
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="X2" value="tall"/>
          <SimplePredicate field="X3" value="small"/>
        </CompoundPredicate>
      </CompoundPredicate>
    </If>
    <Then>
      <SimplePredicate field="Class" value="1"/>
    </Then>
    <Grade>0.1</Grade>
  </Rule>
</Rulebase>
```

Listing 3.14: The *Rulebase* schema of the example based on the FRBCS model.

3.3 Implementations of frbsPMML

The frbsPMML format described above is a complete specification for representing the most commonly used model types. As most XML representations, it is designed to be complete and exhaustive, and usually considered for manual editing. Thus, we present two libraries of software for managing the frbsPMML representation. They are published under an open-source license, hence available freely, for use, adaption, and extension.

The two libraries represented in the following are called "frbs" and "frbsJpmml." They can be used to export an FRBS model to the frbsPMML format and vice versa. The general workflow of the applications to generate an FRBS model and perform prediction for new data can be seen in Figure 3.2.

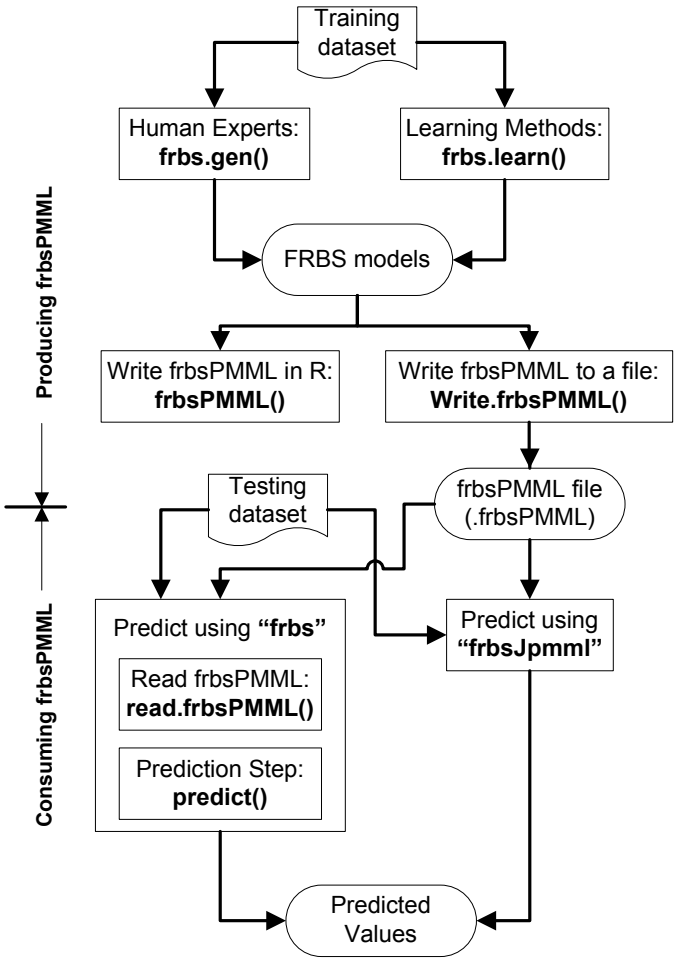


Figure 3.2: Workflow and interactions between "frbs" and "frbsJpmml".

3.3.1 The Extension on the "frbs" Package

As shown in Table 2.1, we have included functions related to frbsPMML in the current version of "frbs." There are three functions designated for managing the frbsPMML format. First, there are two functions that are used for converting FRBS models into the frbsPMML format: *frbsPMML()* and *write.frbsPMML()*. Then, in order to consume/import the frbsPMML format to an FRBS model, we execute *read.frbsPMML()*.

The following are the signatures of the functions related to frbsPMML:

- *frbsPMML()*: Though the function has several arguments, usually only the *model* parameter needs to be supplied which refers to the FRBS model.

```
frbsPMML(model, model.name = "frbs_model", app.name = "frbs",
  description = NULL, copyright = NULL,
  algorithm.name = model$method.type, ...)
```

- *read.frbsPMML()*: The function has as its only parameter the name of the file to read.

```
read.frbsPMML(fileName)
```

- *write.frbsPMML()*: There are two required parameters: *object* and *fileName*. *object* represents the FRBS model in R format whereas *fileName* is the name of the file where the model will be written to.

```
write.frbsPMML(object, fileName = NULL)
```

So, as illustrated in Figure 3.2, we can use "frbs" both as an frbsPMML producer and a consumer with the following steps:

1. Construct an FRBS model: this can be done by executing *frbs.learn()* or *frbs.gen()*.
2. Export the model to frbsPMML format: we call *write.frbsPMML()* to save the model to a file or *frbsPMML()* to store the model in frbsPMML format in an R object. Obviously, after obtaining the model in frbsPMML format, we can also modify directly the file.
3. Import the FRBS model in frbsPMML format to an R object: we execute *read.frbsPMML()*.
4. Perform prediction for new data with *predict()*.

3.3.2 The Predictor Engine "frbsJpmml" in Java

This PMML-consumer application is implemented in Java and can be used to make predictions from the FRBS models that are available in the PMML

format. It is designated in compliance with "frbs," and provides the standard functionalities for constructing an FRBS model.

Basically, "frbsJpmml" consists of four parts, as follows:

- *DataReader*: It is a package containing classes for reading new data and saving results into files.
- *FRBSEngine*: It consists of classes representing the FRBS models. There are three child classes of the *frbsModel* class representing the models: *MamdaniModel*, *TSKModel*, and *FRBCSModel*. Additionally, in the parent class *frbsModel* we include the *fuzzifier()* and *Inference()* which are methods used for fuzzifying data and reasoning, respectively. *predict()*, an abstract method for prediction, is included in this part as well.
- *PMMLreader*: It is a package used for reading/importing FRBS models in the PMML format to Java objects. A verification procedure of the obtained model is also included in this part.
- *MainIOfrbs*: It is a main package included the class *frbsJpmml* which has the *main()* method. Therefore, it is an interface to users to work with the package.

A brief description explaining classes and their methods involved to predict new data can be seen in Figure 3.3.

Detailed descriptions and the installer can be found in the project website at `http://dicits.ugr.es/software/frbsJpmml/`.

3.4 Features and Benefits of frbsPMML

This section aims to recapitulate all features of the presented framework and their benefits of implementations for researchers and practitioners. Moreover, a short comparison with representations from other applications is presented. We discuss features included in the new representation from the following perspectives: completeness of FRBS models and expressiveness of the language.

Regarding FRBS models included in the representation we consider three models: Mamdani, TSK, and FRBCS, with the following detailed specifications:

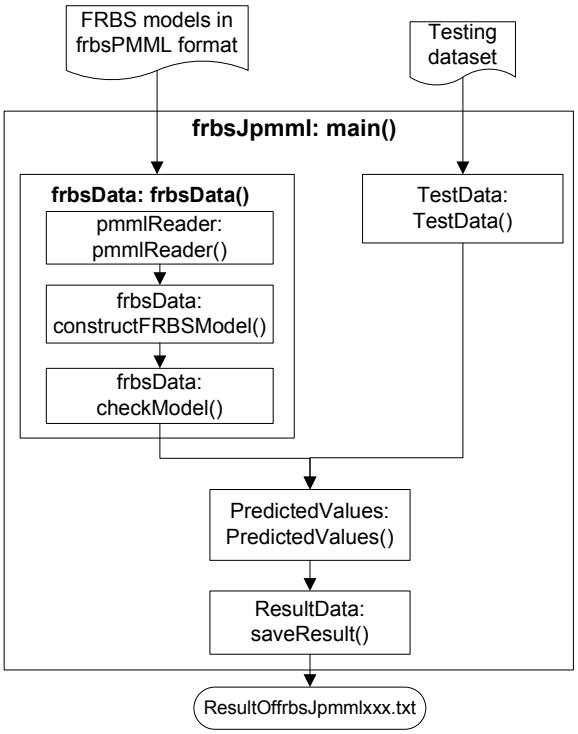


Figure 3.3: Classes and their methods involved to predict new data in *frbsJpmml*.

- In the *InferenceSchema* part, we provide complete parameters, such as conjunction, disjunction, aggregation, and implication operators. Furthermore, each operator has several options representing different approaches.
- The *Database* component supports five membership functions: Gaussian, triangle, trapezoid, sigmoid, and generalized bell. Besides parameters of the membership functions being defined in an easy way, the main benefit is that we allow to set different numbers of labels and membership functions in a particular variable.
- The *Rulebase* used to represent rule-based knowledge has several useful features. Firstly, we can mix boolean operators, i.e., *and* and *or*, together in one rule. It is not necessary to involve all variables in each rule. In other words, the interpretability of rules is favored by the representation. Linguistic hedges can be also included together with

fuzzy terms in the *value* element. Furthermore, for the TSK model, the representation allows to use first- and zero-order TSK. For dealing with classification tasks, we provide the FRBCS representation that involves a degree component for each rule.

- The XML Schema frbsPMML is compatible with PMML that is an established industry standard. Furthermore, as included in PMML, several facilitations for data analysis are available as well, such as data transformation, missing values completion, etc. Additionally, descriptions of data are included in the *DataDictionary* element containing information about names, ranges, and types of variables.

From a language point of view, the new representation offers several advantages as follows:

- In the *Rulebase*, each rule is constructed in a recursive way, and it contains two components: *SimplePredicate* and *CompoundPredicate*. This means it has a sophisticated structure which makes rule reduction and extension a straightforward task. In fact, the expression represents a mathematical formulation of rules.
- The XML schema used to specify the representation provides transparency and readability of documents. Therefore, it is easy for users to read, understand, and modify the documents. Furthermore, since we develop an open standard, other researchers can contribute.
- An FRBS model represented has a text-based representation. So, human experts can read it without problems. It can also be easily modified with a text editor, archived and transferred to other platforms. Moreover, further deployment is possible, e.g., for cloud computing applications.
- The representation provides some validity and verification components, such as *numberOfRules* and *numberOfLabels* are used to validate the number of rules and the number of linguistic values.

It can be seen that the representation helps the interpretability of FRBS models as proposed in [42], which depends on the model structure, the number of input variables, the number of fuzzy rules, the number of linguistic terms, and the shape of the fuzzy sets. For example, the representation allows to define a subset of input variables included in fuzzy rules. Furthermore, we provide an XML Schema of the representation in order to allow

other researchers to extend the deployment and integration of FRBS with other tools and models for data analysis.

3.5 A Comparison with Other Representation Proposals

Some authors have produced some other proposals to represent FRBSs. In this section we compare our proposal to the most relevant alternatives. Table 3.1 shows a comparison of the proposed format with others. We consider four formats: XFL3 [21], *.fis* (MATLAB) [277], XFSML [196], and FisPro [105].

XFL3 is a formal language representing fuzzy systems that is implemented by Xfuzzy. It consists of two parts: the logical definition of the system structure and the mathematical definition of the fuzzy functions. Basically, an FRBS model is specified in a function-based format. Therefore, for common users a complex model in this format can be difficult to read and understand. Additionally, there exist several membership functions, operators, hedges, and defuzzification methods. Next, XFSML is an XML-based language for modelling fuzzy systems. It contains four components: domains, partitions, relations, and modules. One main drawback of this representation is that rulesets are expressed in a relatively complicated manner. Furthermore, though it attempts to be a standard modeling language in the fuzzy community, to the best of our knowledge it is not implemented by any applications, and it is not documented in a formal schema of XML.

In the MATLAB environment, the Fuzzy Logic Toolbox has the proprietary *.fis* format. Since the format is not open, it does not facilitate interoperability. The same holds for FisPro.

Another work that is similar to and improved by our research is the Fuzzy Markup Language (FML). It is an emerging XML-based markup language used for designing and implementing fuzzy controllers (FLC) [3, 4]. Two models are supported in this representation: Mamdani and TSK. An interface connecting to the Matlab Fuzzy Logic Toolbox is provided, as well as Extensible Stylesheet Language Transformations (XSLTs) that are used to convert the FML fuzzy controller to a representation in a general purpose computer language. Even though FML is quite similar to the proposed

representation in this paper, we improve and extend some aspects. For example, FML is not designed to accommodate a rule containing mixed operators (i.e., "and" and "or"). This issue is resolved by frbsPMML since it constructs a fuzzy rule in a recursive way. Secondly, since FML is used for representing FCL, it does not provide other typical components included in data mining, such as data pre-processing, missing value handling, etc. Since frbsPMML adopts the schema of PMML, we have the same capabilities as PMML for dealing with data mining processes. Another drawback of FML that attempts to be refined by frbsPMML is that no formal definition of an XML Schema is provided. So, it is relatively difficult to extend the format.

In addition to a standard representation, the study in [18] proposes a representation based on the unified modeling language (UML), called the evolutionary computing modeling language (ECML). It focuses on representing the concepts of the meta evolutionary computation domain. There is a significant drawback of the representation, which is that the graphical schema can be difficult to be understood and processed when ECML expresses a big model. In [302] rule-based representations based on the Resource Description Framework and Ontology Representation Languages (RDFS and OWL) have been proposed. The format is designed so that it easily allows to supply it to database management systems, such as the Oracle RDBMS. Since these studies are not related to fuzzy sets, we do not include them in Table 3.1.

Table 3.1: Comparison with other representations

Components	frbsPMML	XFL3	.fis (MATLAB)	XFSML	FisPro	FML
General						
Open standard	Yes	No	No	Yes	No	No
Implementations	"frbs", "frbsJpmml"	"Xfuzzy"	"Fuzzy Logic Toolbox"	-	"FisPro", "GUAJE"	"Fuzzy Logic Toolbox"
Other features	Data mining methods (e.g., neural networks, association rules, etc.), Data preprocessing (e.g., transformations, missing value completion)	Support for Java, C, C++, VHDL, and SysGen	-	-	-	-
Completeness of FRBS models						
Models	Mamdani, TSK, FRBCS	Mamdani, TSK	Mamdani, TSK	Mamdani, Fuzzy decision trees,	Mamdani	Mamdani, TSK
Inference parameters	Many options	Many options	Many options	Not specified	Many Options	Not specified
Membership functions	Many options	Many options	Many options	Not specified	Many Options	Not specified
Hedges	Supported	Supported	Supported	-	Supported	Supported
Interpretable rules	Supported	Supported	-	-	Supported	Supported
Operators: AND, OR, NOT	Supported	Supported	Supported	-	Supported	Supported (not mixed)
Expressiveness of languages						
Format base	Text (XML)	Function, GUI	Object, GUI	Text (XML)	GUI	Text (XML)
Interoperability	High	Medium	Low	High	Low	High
Validity and verification components	Provided	-	-	-	-	-
Readability	High	Medium	High	Medium	High	High
Ease of extension	High	High	Low	High	Low	Medium

Therefore, according to the features and their benefits, it can be seen that the new representation should be considered as an open standard for representing FRBS models by researchers and practitioners. Since it is an open standard based on XML, other developers and researchers in the fuzzy community can adopt it in any applications and can propose enhancements, e.g., in form of definitions of XML schemata to accommodate complicated models.

3.6 Example of Usage

In this section, two examples showing how to export and import an FRBS model to/from the frbsPMML format are presented. Basically, we just continue examples that have been presented in Section 2.3.

3.6.1 Regression

After obtaining the model (*mod.reg*) in Section 2.3, we can construct and save the model in frbsPMML format to a file with extension *frbsPMML*, which is *modRegress.frbsPMML*, by using "frbs" as follows:

```
R> write.frbsPMML(frbsPMML(mod.reg), "modRegress")

[1] "modRegress.frbsPMML"
```

We can also export to frbsPMML format but with storing it within R in the memory or directly displaying it as follows:

```
R> frbsPMML(mod.reg)

<frbsPMML version="1.0" xmlns="http://sci2s.ugr.es/dicits/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://sci2s.ugr.es/dicits/">
 <Header copyright="Copyright (c) 2015 Lala">
  <Extension name="user" value="Lala" extender="frbs"/>
  <Application name="frbs" version="1.4"/>
```

```
 <Timestamp>2015-03-28 22:43:31</Timestamp>
</Header>
<DataDictionary numberOfFields="3">
 <DataField name="X" optype="continuous" dataType="double">
  <Interval closure="closedClosed" leftMargin="-2"
     rightMargin="1.92"/>
 </DataField>
 <DataField name="Y" optype="continuous" dataType="double">
  <Interval closure="closedClosed" leftMargin="-2"
     rightMargin="1.92"/>
 </DataField>
 <DataField name="Z" optype="continuous" dataType="double">
  <Interval closure="closedClosed" leftMargin="0.0526315789473684"
     rightMargin="0.996746301114346"/>
 </DataField>
</DataDictionary>
<FrbsModel modelName="MAMDANI" functionName="regression"
    algorithmName="WM" targetFieldName="Z">
 <MiningSchema>
  <MiningField name="X" usageType="active"/>
  <MiningField name="Y" usageType="active"/>
  <MiningField name="Z" usageType="predicted"/>
 </MiningSchema>
 <Output>
  <OutputField name="Predicted_Z" optype="continuous"
     dataType="double" feature="predictedValue"/>
 </Output>
 <InferenceSchema>
  <ConjunctionOperator value="MIN"/>
  <DisjunctionOperator value="MAX"/>
  <ImplicationOperator value="LUKASIEWICZ"/>
  <AggregationOperator value="COG"/>
 </InferenceSchema>
```

```
<Database>
 <MembershipFunction name="X" numberOfLabels="5">
  <FuzzyTerm name="very.small" type="GAUSSIAN">
   <Parameters>
    <Mean>0</Mean>
    <Variance>0.0875</Variance>
   </Parameters>
  </FuzzyTerm>
  <FuzzyTerm name="small" type="GAUSSIAN">
   <Parameters>
    <Mean>0.25</Mean>
    <Variance>0.0875</Variance>
   </Parameters>
  </FuzzyTerm>
  <FuzzyTerm name="medium" type="GAUSSIAN">
   <Parameters>
    <Mean>0.5</Mean>
    <Variance>0.0875</Variance>
   </Parameters>
  </FuzzyTerm>
  <FuzzyTerm name="large" type="GAUSSIAN">
   <Parameters>
    <Mean>0.75</Mean>
    <Variance>0.0875</Variance>
   </Parameters>
  </FuzzyTerm>
  <FuzzyTerm name="very.large" type="GAUSSIAN">
   <Parameters>
    <Mean>1</Mean>
    <Variance>0.0875</Variance>
   </Parameters>
  </FuzzyTerm>
 </MembershipFunction>
```

```
...
<MembershipFunction name="Z" numberOfLabels="5">
 <FuzzyTerm name="very.small" type="GAUSSIAN">
  <Parameters>
   <Mean>0</Mean>
   <Variance>0.0875</Variance>
  </Parameters>
 </FuzzyTerm>
 <FuzzyTerm name="small" type="GAUSSIAN">
  <Parameters>
   <Mean>0.25</Mean>
   <Variance>0.0875</Variance>
  </Parameters>
 </FuzzyTerm>
 <FuzzyTerm name="medium" type="GAUSSIAN">
  <Parameters>
   <Mean>0.5</Mean>
   <Variance>0.0875</Variance>
  </Parameters>
 </FuzzyTerm>
 <FuzzyTerm name="large" type="GAUSSIAN">
  <Parameters>
   <Mean>0.75</Mean>
   <Variance>0.0875</Variance>
  </Parameters>
 </FuzzyTerm>
 <FuzzyTerm name="very.large" type="GAUSSIAN">
  <Parameters>
   <Mean>1</Mean>
   <Variance>0.0875</Variance>
  </Parameters>
 </FuzzyTerm>
</MembershipFunction>
```

```
  </Database>
  <Rulebase numberOfRules="47">
   <Rule id="1">
    <If>
     <CompoundPredicate booleanOperator="and">
      <SimplePredicate field="X" value="very.small"/>
      <SimplePredicate field="Y" value="very.small"/>
     </CompoundPredicate>
    </If>
    <Then>
     <SimplePredicate field="Z" value="very.small"/>
    </Then>
   </Rule>
   ...
   <Rule id="47">
    <If>
     <CompoundPredicate booleanOperator="and">
      <SimplePredicate field="X" value="medium"/>
      <SimplePredicate field="Y" value="large"/>
     </CompoundPredicate>
    </If>
    <Then>
     <SimplePredicate field="Z" value="small"/>
    </Then>
   </Rule>
  </Rulebase>
 </FrbsModel>
</frbsPMML>
```

Then, we import and apply the model from the file *modRegress.frbsPMML* by the following command:

```
R> objReg <- read.frbsPMML("modRegress.frbsPMML")
R> res.test <- predict(objReg, data.tst)
```

3.6.2 Classification

As on the regression task, we can export the FRBS model (i.e., *mod.class* in Section 2.3) to the frbsPMML format as follows:

```
R> frbsPMML(mod.class)
```

```
<frbsPMML version="1.0" xmlns="http://sci2s.ugr.es/dicits/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://sci2s.ugr.es/dicits/">
 <Header copyright="Copyright (c) 2015 Lala">
  <Extension name="user" value="Lala" extender="frbs"/>
  <Application name="frbs" version="1.4"/>
  <Timestamp>2015-03-28 22:43:32</Timestamp>
 </Header>
 <DataDictionary numberOfFields="5">
  <DataField name="Sepal.Length" optype="continuous"
     dataType="double">
   <Interval closure="closedClosed" leftMargin="4.3"
      rightMargin="7.9"/>
  </DataField>
  <DataField name="Sepal.Width" optype="continuous"
     dataType="double">
   <Interval closure="closedClosed" leftMargin="2"
      rightMargin="4.4"/>
  </DataField>
  <DataField name="Petal.Length" optype="continuous"
     dataType="double">
   <Interval closure="closedClosed" leftMargin="1"
      rightMargin="6.9"/>
  </DataField>
  <DataField name="Petal.Width" optype="continuous"
     dataType="double">
   <Interval closure="closedClosed" leftMargin="0.1"
```

```
    rightMargin="2.5"/>
 </DataField>
 <DataField name="Species" optype="categorical"
    dataType="string">
  <Value value="1"/>
  <Value value="2"/>
  <Value value="3"/>
 </DataField>
</DataDictionary>
<FrbsModel modelName="FRBCS" functionName="classification"
   algorithmName="FRBCS.CHI" targetFieldName="Species">
 <MiningSchema>
  <MiningField name="Sepal.Length" usageType="active"/>
  <MiningField name="Sepal.Width" usageType="active"/>
  <MiningField name="Petal.Length" usageType="active"/>
  <MiningField name="Petal.Width" usageType="active"/>
  <MiningField name="Species" usageType="predicted"/>
 </MiningSchema>
 <Output>
  <OutputField name="Predicted_Species" feature="predictedValue"/>
  <OutputField name="Probability_1" optype="continuous" dataType="double"
     feature="probability" value="1"/>
  <OutputField name="Probability_2" optype="continuous" dataType="double"
     feature="probability" value="2"/>
  <OutputField name="Probability_3" optype="continuous" dataType="double"
     feature="probability" value="3"/>
 </Output>
 <InferenceSchema>
  <ConjunctionOperator value="MIN"/>
  <DisjunctionOperator value="MAX"/>
  <ImplicationOperator value="ZADEH"/>
 </InferenceSchema>
 <Database>
```

```
<MembershipFunction name="Sepal.Length" numberOfLabels="3">
 <FuzzyTerm name="small" type="TRAPEZOID">
  <Parameters>
   <Left>0</Left>
   <LeftMiddle>0</LeftMiddle>
   <RightMiddle>0.2</RightMiddle>
   <Right>0.4</Right>
  </Parameters>
 </FuzzyTerm>
 <FuzzyTerm name="medium" type="TRAPEZOID">
  <Parameters>
   <Left>0.23</Left>
   <LeftMiddle>0.43</LeftMiddle>
   <RightMiddle>0.53</RightMiddle>
   <Right>0.73</Right>
  </Parameters>
 </FuzzyTerm>
 <FuzzyTerm name="large" type="TRAPEZOID">
  <Parameters>
   <Left>0.6</Left>
   <LeftMiddle>0.8</LeftMiddle>
   <RightMiddle>1</RightMiddle>
   <Right>1</Right>
  </Parameters>
 </FuzzyTerm>
</MembershipFunction>
...
<MembershipFunction name="Petal.Width" numberOfLabels="3">
 <FuzzyTerm name="small" type="TRAPEZOID">
  <Parameters>
   <Left>0</Left>
   <LeftMiddle>0</LeftMiddle>
   <RightMiddle>0.2</RightMiddle>
```

```
    <Right>0.4</Right>
   </Parameters>
  </FuzzyTerm>
  <FuzzyTerm name="medium" type="TRAPEZOID">
   <Parameters>
    <Left>0.23</Left>
    <LeftMiddle>0.43</LeftMiddle>
    <RightMiddle>0.53</RightMiddle>
    <Right>0.73</Right>
   </Parameters>
  </FuzzyTerm>
  <FuzzyTerm name="large" type="TRAPEZOID">
   <Parameters>
    <Left>0.6</Left>
    <LeftMiddle>0.8</LeftMiddle>
    <RightMiddle>1</RightMiddle>
    <Right>1</Right>
   </Parameters>
  </FuzzyTerm>
 </MembershipFunction>
</Database>
<Rulebase numberOfRules="19">
 <Rule id="1">
  <If>
   <CompoundPredicate booleanOperator="and">
    <SimplePredicate field="Sepal.Length" value="small"/>
    <CompoundPredicate booleanOperator="and">
     <SimplePredicate field="Sepal.Width" value="medium"/>
     <CompoundPredicate booleanOperator="and">
      <SimplePredicate field="Petal.Length" value="small"/>
      <SimplePredicate field="Petal.Width" value="small"/>
     </CompoundPredicate>
    </CompoundPredicate>
```

```
     </CompoundPredicate>
    </If>
    <Then>
     <SimplePredicate field="Species" value="1"/>
    </Then>
    <Grade>1</Grade>
   </Rule>
   ...
   <Rule id="19">
    <If>
     <CompoundPredicate booleanOperator="and">
      <SimplePredicate field="Sepal.Length" value="small"/>
      <CompoundPredicate booleanOperator="and">
       <SimplePredicate field="Sepal.Width" value="small"/>
       <CompoundPredicate booleanOperator="and">
        <SimplePredicate field="Petal.Length" value="medium"/>
        <SimplePredicate field="Petal.Width" value="large"/>
       </CompoundPredicate>
      </CompoundPredicate>
     </CompoundPredicate>
    </If>
    <Then>
     <SimplePredicate field="Species" value="3"/>
    </Then>
    <Grade>0.333333333333333</Grade>
   </Rule>
  </Rulebase>
 </FrbsModel>
</frbsPMML>
```

For importing frbsPMML, users can refer to the way on the regression task.

3.7 Summary

This chapter presents specifications and implementations of the universal representation framework frbsPMML, which is associated to the second objective. It can be summarized as follows:

1. frbsPMML, which is a universal representation framework for FRBSs based on the PMML standard, has been presented. It implements all models supported by FRBSs, namely Mamdani, TSK, and FRBCS models. Three essential components of FRBS models, the database, rulebase, and inference parameters, are provided in a flexible way. The representation offers benefits for: interoperability, reproducibility, transparency, interpretability, and flexibility.

2. An extension of "frbs", which is a standard package for constructing FRBS models in the R environment, to represents models in the frbsPMML format.

3. The software "frbsJpmml", written in Java, can be used to import FRBS models in the frbsPMML format and for prediction on new data.

4. Usage examples of both software libraries have been illustrated in the paper.

5. A comparison with other formats is represented to emphasize important benefits of frbsPMML.

Furthermore, a paper describing the tool has been submitted:

> L.S. Riza, C. Bergmeir, F. Herrera, and J.M. Benítez. A Universal Representation Framework for Fuzzy Rule-Based Systems Based on PMML. Information Sciences, 2015 (submitted).

Chapter 4

The "RoughSets" Package

In this chapter we present the research carried out towards Objective 2, which is to implement algorithms based on RST and FRST used for data preprocessing (i.e., discretization, feature selection, and instance selection), classification, and regression. First, we introduce the "RoughSets" package as the result of the research. Then, the package architecture and its implementation details are explain to provide a complete explanation related to "RoughSets." We present some usage examples and a comparison with other package implementing the same concepts. Finally, a short summary concludes the chapter.

4.1 Introduction

There are several tools implementing RST and FRST for dealing with various problems. Rough Set Data Explorer (ROSE) is an RST-based software system created by the Laboratory of Intelligent Decision Support Systems of the Institute of Computing Science in Poznań [225, 226]. In [24, 25], a free software system for data exploration, classification support, and knowledge discovery called the Rough Set Exploration System (RSES) was presented. The rough set toolkit for analysis of data (ROSETTA), which is an advanced system for RST data analysis [209, 210], includes the RSES library as the computation kernel. Also, a few algorithms from FRST have been implemented in the Waikato Environment for Knowledge Analysis (WEKA) [136]. WEKA is a collection of machine learning algorithms for data mining

tasks implemented in Java [109]. Rough Set Based Intelligent Data Analysis System (RIDAS) is another data mining toolkit utilizing notions from RST [290]. It was developed at Chongqing University of Posts and Telecommunications and consists of a C++ kernel and a GUI for Windows systems. An important data mining system for inducing decision rules from various types of data, called Learning from Examples based on Rough Sets (LERS), was created at University of Kansas [98]. There have also been developed a few rule based expert systems for a medical diagnostics purposes. One of the most prominent is PRIMEROSE [283] which allows generation of decision and inhibitory rules to construct reliable differential diagnosis. Furthermore, after reviewing R packages in CRAN, we cannot find an R package that implements RST and FRST. Therefore, the research attempting to develop an R package unifying RST and FRST is required. The research has produced the "RoughSets" package.

"RoughSets" is an R package integrating implementations of various algorithms based on RST and FRST in a single software library. It is a result of a collaboration between two research groups: Soft Computing and Intelligent Information Systems, Universidad de Granada and Institute of Mathematics, University of Warsaw, Poland. Currently, it is available from CRAN in version 1.2-0, at `http://cran.r-project.org/package=RoughSets`. Moreover, detailed explanation and some examples can be found on our groups' web page of the package, available at `http://dicits.ugr.es/software/RoughSets`. It is licensed under the terms of the GPL so that it can be redistributed or modified under that term.

There are more than 40 algorithms included in the package for dealing with missing values, discretization, feature selection, instance selection, classification using rule-based classifiers and instance-based classifiers, and regression. These features can be seen in Figure 4.1, which depicts the seven tasks that have been considered. For example, in the basic concepts, we have implemented eight functions, e.g., *BC.IND.relation.RST* used to calculate the indiscernibility relations based on RST. We include discretization methods based on RST that calculate cut values using static & dynamic, local & global, and supervised & unsupervised methods. The other tasks included in this package are instance selection, feature selection, rule induction, and nearest neighbor-based classifiers. While algorithms used for feature selection and rule induction are based on RST and FRST, instance selection and nearest neighbor-based classifiers are based on FRST.

Furthermore, another main feature of the package is that it can be used

not only for practitioners who perform data analysis, but also for researchers who develop a new approach based on RST and FRST. In other words, we facilitate researchers to define their own functions expressing a new basic concept, such as lower and upper approximations. It can be done since "RoughSets" employs a functional programming so that instead of modifying code directly, users just need to supply user-defined functions.

Additionally, the package has embedded some datasets used to do experimental studies. For example, the hiring dataset is a dataset used originally in [159]. We also provide the housing [111], wine [80], and pima [11] datasets. A complete manual explaining the package in detail can be downloaded from CRAN. In the manual, we also ship many demos showing to users how to use the package. As other R packages submitted in CRAN, to keep a standard quality of CRAN package, "RoughSets" has been checked by the CRAN teams. Moreover, "RoughSets" has been included in the CRAN view: Machine Learning & Statistical Learning.

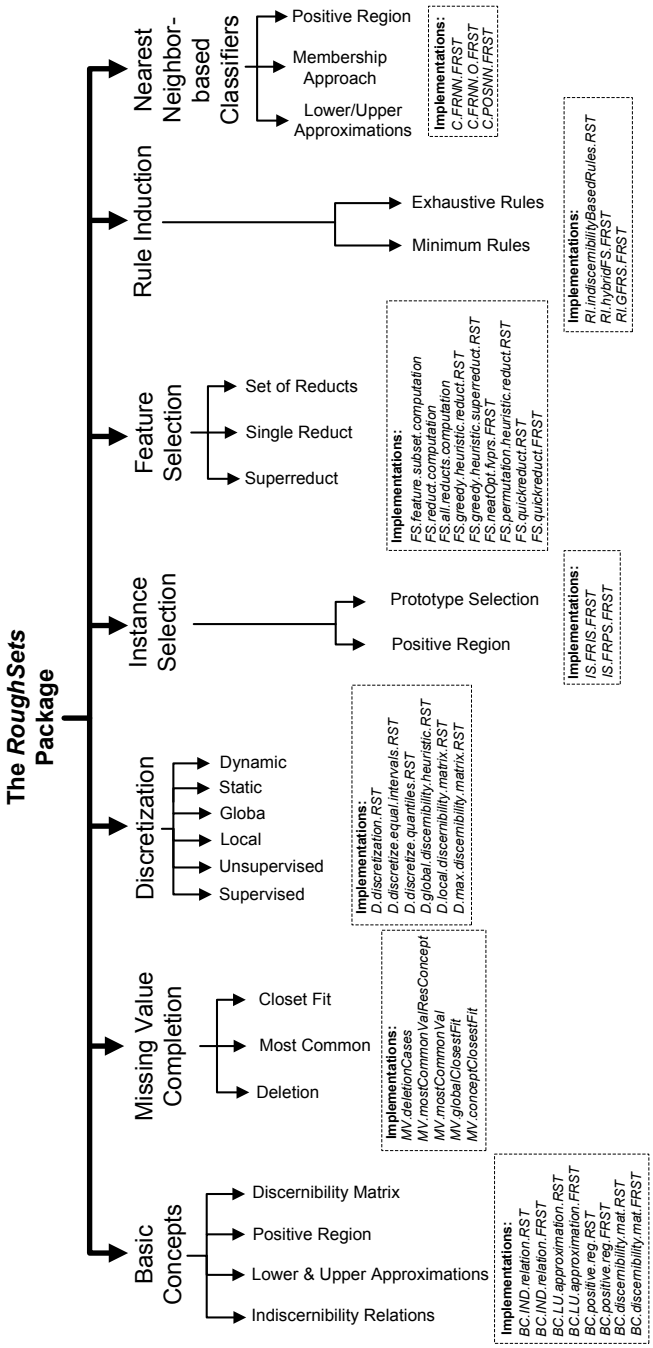


Figure 4.1: Models and their implementations in "RoughSets."

4.2 The Package Architecture and Implementation Details

As shown in Figure 4.1, since there are more than 40 functions included in the package for implementing various algorithms, we compose their names with three parts separated by dots: *prefix*, *suffix*, and *middle* as follows:

- *prefix*: This part shows that corresponding functions perform a particular task as follows:
  - *BC*: basic concepts of a certain theory,
  - *D*: discretization,
  - *FS*: feature selection,
  - *IS*: instance selection,
  - *RI*: rule-based classifiers (rule induction),
  - *C*: nearest neighbor-based classifiers,
  - *SF*: support functions,
  - *MV*: missing value handling,
  - *X*: auxiliary functions.
- *suffix*: Two names in this part are *RST* and *FRST*. The suffix *RST* refers to rough set theory while *FRST* shows that the function is applied to fuzzy rough set theory. Additionally, some functions that do not have these suffixes are used for both the theories.
- *middle*: The actual function name. It can consist of more than one word separated by points.

For example, *BC.positive.reg.RST* is a function based on RST used to calculate the positive region, which is a part of the basic concepts. Other functions that have names not based on the above rules are **S3** functions which are built-in functions in the R framework. For instance, *summary* and *predict* are used to summarize objects and predict new data, respectively.

These tasks will be explained in detail, as follows:

1. Basic concepts: In the basic concepts, we consider algorithms for calculating the indiscernibility relation, the lower and upper approximations, the regions, and the discernibility matrix. These tasks are

Table 4.1: Functions included in the basic concepts of "RoughSets" and their references.

Names of functions	Descriptions	References
BC.IND.relation.RST	Indicernibility relation based on RST	[213]
BC.IND.relation.FRST	Indiscernibility relation based on FRST	[200, 143, 119]
BC.LU.approximation.RST	Approximations based on RST	[213]
BC.LU.approximation.FRST	Approximations based on FRST	[238, 53, 56, 309, 120, 249]
BC.positive.reg.RST	Positive region based on RST	[213]
BC.positive.reg.FRST	Positive region based on FRST	[143]
BC.discernibility.mat.RST	Decision-relative discernibility matrix based on RST	[258]
BC.discernibility.mat.FRST	Decision-relative discernibility matrix based on FRST	[282, 309, 46, 45]

implemented in eight functions, e.g., *BC.IND.relation.RST* used to calculate the indiscernibility relations based on RST. The complete list of the functions is presented in Table 4.1. These functions are important because they are used as the core functions for other functions in the package.

2. Missing value completion: It refers to a process to deal with missing values. This task is useful to be included in the package since all functions based on RST and FRST in the package can not handle missing values automatically. There are five functions implemented to handle missing values, as shown in Table 4.2. Furthermore, in order to provide a simple execution, we can also invoke the wrapper function *MV.missingValueCompletion()* where the following is its signature:

```
MV.missingValueCompletion(decision.table, type.method)
```

where *decision.table* is the training data in the R object *DecisionTable*, and a chosen method is assigned to *type.method*.

Table 4.2: Functions included in the missing value completion in "RoughSets" and their references.

Names of functions	Descriptions	References
MV.deletionCases	Deleting instances	[94]
MV.mostCommonValResConcept	Assigning the most common value of an attribute restricted by concepts	[94]
MV.mostCommonVal	Replacing the attribute mean or common values	[94]
MV.globalClosestFit	The global closest fit	[94]
MV.conceptClosestFit	The concept closest fit	[94]

3. Discretization: It refers to approaches for converting real-valued attributes into nominal ones in information systems. In the case of RST, we have to make sure that our data contains nominal values, otherwise we have to perform discretization first. In the package, we consider several types of discretization, which are dynamic, static, global, local, unsupervised, and supervised. Detailed description of functions in the package is presented in Table 4.3. Furthermore, in order to provide a simple execution, we can also invoke the wrapper function *D.discretization.RST()* where the following is its signature:

```
D.discretization.RST(decision.table, type.method, ...)
```

   where *decision.table* is the training data in the R object *DecisionTable*, and a chosen method is assigned to *type.method*.

4. Instance selection: It is to remove or replace noisy or inconsistent instances from training datasets. In RST, it refers to evaluating each object included in the boundary region or the positive region. In the package, we provide two algorithms as shown in Table 4.4.

5. Feature selection: It is a process to find a subset of attributes which gives the same quality as the complete feature set. In other words, a purpose of the feature selection is to identify significant attributes

Table 4.3: Functions included in the discretization approaches in "RoughSets" and their references.

Names of functions	Descriptions	
D.max.discernibility.matrix.RST	Discretization based on maximal discernibility	[23]
D.local.discernibility.matrix.RST	Discretization based on local strategy	[23]
D.global.discernibility.heuristic.RST	Discretization based on global maximum discernibility heuristic	[205]
D.discretize.quantiles.RST	Discretization based on quantiles	[67]
D.discretize.equal.intervals.RST	Discretization based on equal interval size	[67]

Table 4.4: Functions included in the instance selection in "RoughSets" and their references.

Names of functions	Descriptions	
IS.FRIS.FRST	Fuzzy rough instance selection	[138]
IS.FRPS.FRST	Fuzzy rough prototype selection	[285]

and to eliminate the dispensable ones. According to the output produced by feature-selection algorithms, we may divide them into three groups: those that produce a superreduct, a set of reducts, or a single reduct. An attribute $a \in B \subseteq A$ can be regarded as dispensable in B if $R_B = R_{B\setminus\{a\}}$ otherwise a is called indispensable in B. Furthermore, in both RST and FRST the feature selection typically refers to finding a *reduct* or a *superreduct*. A superreduct is a set of attributes $B \subseteq A$, such that $R_B = R_A$, where R_B and R_A are the indiscernibility relations defined by B and A, respectively [214, 216]. If it is also minimal (w.r.t. inclusion), then it is called a reduct. The intersection of all reducts is called the *core*. All functions included in the package can be seen in Table 4.5. Furthermore, in order to provide a simple execution, we provide three wrapper functions, where the following are their signatures:

```
FS.reduct.computation(decision.table,method,...)
FS.feature.subset.computation(decision.table,method,...)
FS.all.reducts.computation(discernibilityMatrix)
```

where *decision.table* is the training data in the R object *DecisionTable*, and a chosen method is assigned to *type.method*.

6. Rule induction: It is also called a rule-based classifier, which is an approach used to extract knowledge in *IF-THEN* rules. In RST, a rule for the decision table $\mathscr{A}$ is called a decision rule denoted by *IF* φ *THEN* $d = v$, where $\varphi \in \mathscr{C}(A, V_a)$. $\mathscr{C}(A, V_a)$ is a set of pairs of conditional attributes A and their corresponding values V_a that are connected by the propositional $\wedge$ (conjunction), $\vee$ (disjunction), and $\neg$ (negation). The decision rule is *true* in $\mathscr{A}$ if, and only if, $\| \varphi_{\mathscr{A}} \| \subseteq \| d = v_{\mathscr{A}} \|$ where in this case, $\| . \|$ is the set of objects *matching* the decision rule [216]. In the package we consider three different methods as shown in Table 4.6.

7. Nearest neighbor-based classifiers: The nearest neighbor-based classifiers were introduced first by Fix and Hodges in [79]. Then, in 1967 they were improved and made famous by Cover and Hart [57]. In supervised learning, the k-nearest neighbor-based classifiers are defined as methods that predict new data/patterns based on the most similar

Table 4.5: Functions included in the feature selection in "RoughSets" and their references.

Names of functions	Descriptions	
FS.quickreduct.RST	Feature selection based on QuickReduct	[254]
FS.quickreduct.FRST	Feature selection based on fuzzy QuickReduct	[141, 249, 30, 54, 143, 309, 56, 120]
FS.greedy.heuristic.superreduct.RST	Greedy heuristics for selecting a superreduct	[301, 260, 134]
FS.nearOpt.fvprs.FRST	Feature selection based on near-optimal reduction	[309]
FS.greedy.heuristic.reduct.RST	Greedy heuristics for selecting a reduct	[301, 260, 134]
FS.all.reducts.computation	Wrapper for computing all reducts	[258, 282, 309, 46, 45]
FS.permutation.heuristic.reduct.RST	Permutation heuristic for determining a reduct	[133]

Table 4.6: Functions included in the rule induction in "RoughSets" and their references.

Names of functions	Descriptions	
RI.indiscernibilityBasedRules.RST	Rule induction based on RST	[213]
RI.hybrid.FRST	QuickRules algorithm	[140]
RI.GFRS.FRST	Generalized fuzzy rough set rule induction	[310]

Table 4.7: Functions included in the nearest neighbor-based classifiers in "RoughSets" and their references.

Names of functions	Descriptions	
C.FRNN.FRST	Fuzzy-rough nearest neighbors	[139]
C.FRNN.O.FRST	Fuzzy-rough ownership nearest neighbors	[251]
C.POSNN.FRST	Positive region based fuzzy-rough nearest neighbors	[286]

or nearest k patterns in training data. This section attempts to illustrate enhancements of k-nearest neighbors based on FRST. Table 4.7 show the functions included in this group.

4.3 Examples of Usage

In general speaking, to analyze data using "RoughSets" we follow these steps:

1. Installation and loading the "RoughSets" package: In order to use the package, we need to install and load it as shown in Section 4.3.1.

2. Constructing datasets in the *DecisionTable* format: *DecisionTable* is a standard format expressing a decision table and an information system in "RoughSets". All functions included in the package require *DecisionTable* as the input data. Therefore, we must convert data into this format before calling any functions. Basically, the *DecisionTable* representation contains three attributes, as follows:

   (a) *nominal.attrs*: containing boolean values representing types of attributes whether nominal or not.

   (b) *desc.attrs*: containing two parts: names of attributes and range of data.

   (c) *decision.attr*: showing an index of the decision attribute. In the case of the information system, it is set by *NULL*.

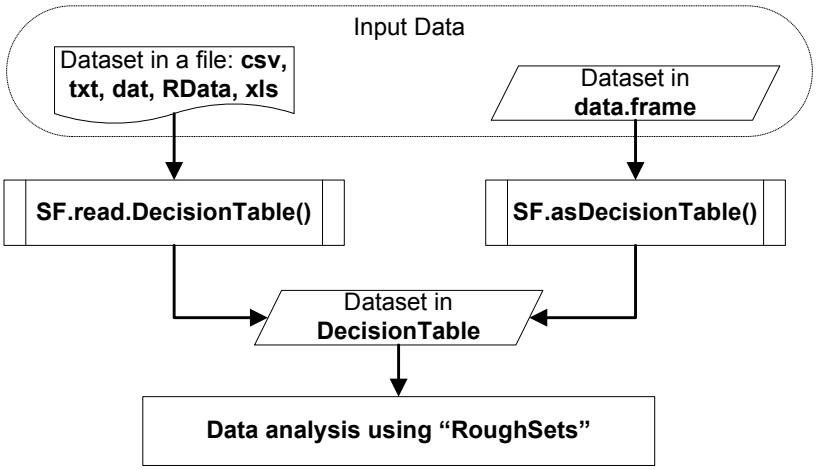


Figure 4.2: Constructing the *DecisionTable* format from a file and *data.frame*.

Figure 4.2 shows two ways to construct *DecisionTable*. First, we can obtain from a file with extensions *csv*, *txt*, *data*, *RData*, and *xls* by executing the function *SF.read.DecisionTable()*. Second, *DecisionTable* can be generated from a dataset represented by *data.frame*. Detailed illustrations showing how to build the format can be seen in Section 4.3.2.

3. Performing data pre-processing: In this part, we can perform missing value handling, discretization, feature selection, and instance selection by invoking associated functions. It should be noted that all functions in this step do not generate a new decision table (i.e., dataset), but they produce a model related to these tasks. For example, a function used for discretization returns cut values as the model. Therefore, as shown in Figure 4.3, we need to execute *SF.applyDecTable()* to materialize the decision table by considering an obtained model. Detailed illustrations showing how to perform data pre-processing can be seen in Section 4.3.4 and 4.3.5.

4. Generating learning models and predict new data: In order to construct a learning model, we provide two techniques: rule-based classifiers (i.e., rule induction) and instance-based classifiers (i.e., nearest neighbor-based classifiers). A simple difference between these two classifiers is that while instance-based classifiers produces a set of selected

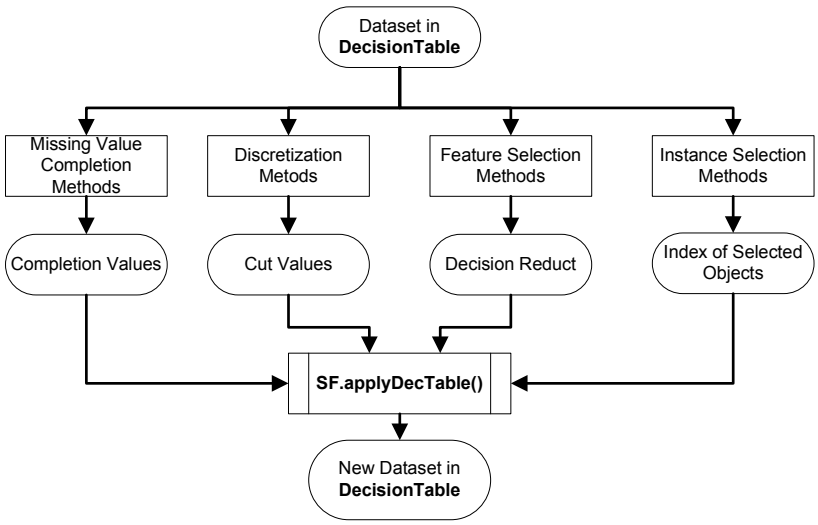


Figure 4.3: Generating a new decision table in the data pre-processing "RoughSets."

instances, rule-based classifiers generates a set of rules. Furthermore, rule-based classifiers performs learning and prediction sequentially, but the other one does these processes at the same time. Figure 4.4 and 4.5 emphasize the differences between the two classifiers. Furthermore, it can be seen in Figure 4.4, the output of learning functions based on rule-based classifiers is a model, called decision rules. Detailed illustrations showing how to perform the learning and prediction steps can be seen in Section 4.3.4 and 4.3.5.

4.3.1 Installation and Loading the "RoughSets" Package

To install "RoughSets" from CRAN, users simply enter the following command in the R environment:

```
R> install.packages(c("class", "RoughSets"))
```

Here, "class" is a package required for some functions of the "RoughSets" package.

The package installation has to be done only once. After that in any session using "RoughSets", we need to load it with the following command:

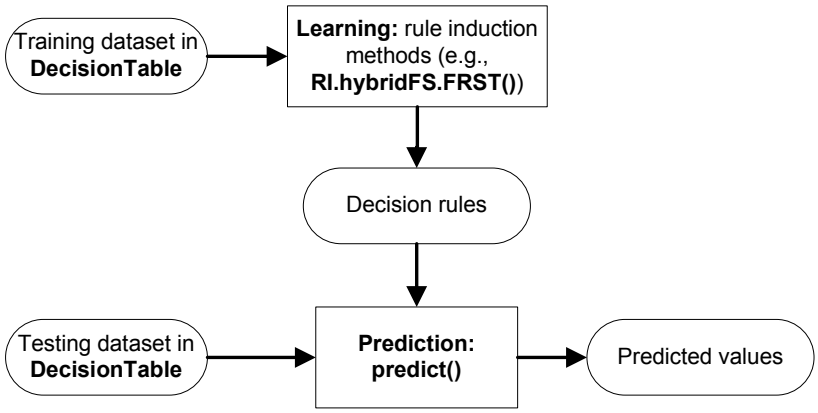


Figure 4.4: The learning and prediction steps based on rule-based classifiers in "RoughSets".

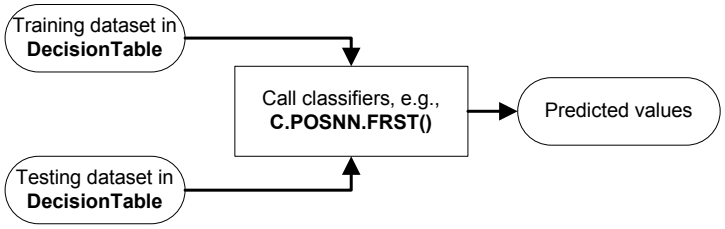


Figure 4.5: The learning and prediction steps based on nearest neighbor-based classifiers "RoughSets".

```
R> library(RoughSets)
```

The command loads the "RoughSets" package and makes its functions available in the R environment. We can see a list of functions included in "RoughSets" by running the following command:

```
R> library(help=RoughSets)
```

All R functions available in "RoughSets" are documented in the R help system in a hypertext format, and in the package manual in pdf format. The manual is available on the package website on CRAN. To get information on a particular function, we can call the help command, e.g.,

```
R> help(BC.IND.relation.RST)
```

In this section, we provide some examples showing how to use the "RoughSets" package. Simple data are used in the examples to give clear illustrations. It will be divided into three parts of examples: constructing the *DecisionTable* format, executing basic concepts, and applications.

4.3.2 Constructing Datasets in the *DecisionTable* Format

Every dataset used in the package has to be in *DecisionTable* format. Let *dt.ex1* be our dataset, the following code shows how to construct the *DecisionTable* of *dt.ex1*.

```
R> dt.ex1 <- data.frame(c(1,0,2,1,1,2,2,0), c(0.5,1.2,0.1,1.2,0.4,
+     2.2,1.1,1.5), c(2,1,0,0,2,0,1,1), c(0,2,1,2,1,1,2,1))
R> colnames(dt.ex1) <- c("a", "b", "c", "d")
R> DecTable.1 <- SF.asDecisionTable(dataset = dt.ex1,
+     decision.attr = 4, indx.nominal = c(1,3:4))
```

We can see *DecTable.1* by typing:

```
R> print.default(DecTable.1)

$a
[1] 1 0 2 1 1 2 2 0

$b
[1] 0.5 1.2 0.1 1.2 0.4 2.2 1.1 1.5

$c
[1] 2 1 0 0 2 0 1 1

$d
[1] 0 2 1 2 1 1 2 1

attr(,"class")
[1] "DecisionTable" "data.frame"
```

```
attr(,"nominal.attrs")
[1]  TRUE FALSE  TRUE  TRUE
attr(,"desc.attrs")
attr(,"desc.attrs")$a
[1] "0" "1" "2"

attr(,"desc.attrs")$b
[1] 0.1 2.2

attr(,"desc.attrs")$c
[1] "0" "1" "2"

attr(,"desc.attrs")$d
[1] "0" "1" "2"

attr(,"decision.attr")
[1] 4
```

Basically, *DecTable.1* (which is a *DecisionTable* object) contains the dataset as a *data.frame*, attribute descriptions (namely *desc.attrs*), type of attributes (namely *nominal.attrs*), and an index of the decision attribute (namely *decision.attr*). In *nominal.attrs*, we can see that the second attribute has real values and the others have nominal values. Furthermore, *desc.attrs* constitutes a set of attributes and their range of values, and *decision.attr* shows that the attribute on index 4 is the decision attribute.

In case we have data with many rows and columns, it may be more convenient to construct *DecisionTable* from files, executing *SF.read.DecisionTable*. An example of this case can be found in the package manual.

4.3.3 Examples of the Basic Concepts

In the following, we illustrate with some examples the basic concepts of RST and FRST.

Based on Rough Set Theory

In this example, we are using the *hiring.dt* dataset [159]. It has been included in the "RoughSets" package. In order to load the data, we need to type the following command:

```
R> data(RoughSetData)
R> hiring.dt <- RoughSetData$hiring.dt
```

First, we calculate the indiscernibility relation. For example, by considering the second and third attributes only, we obtain the indiscernibility relation *IND* with the command:

```
R> IND <- BC.IND.relation.RST(hiring.dt)
```

The *IND* object contains the equivalence classes of the objects.

After obtaining the relation, we can calculate the approximations (namely *roughset*) and positive region (namely *region*) as follows:

```
R> roughset <- BC.LU.approximation.RST(hiring.dt, IND)
R> region <- BC.positive.reg.RST(hiring.dt, roughset)
```

Furthermore, we can also construct the discernibility matrix as follows:

```
R> disc.Mat <- BC.discernibility.mat.RST(hiring.dt)
```

We can then show the results, e.g. the output of *BC.LU.approximation.RST* which is *roughset* by

```
R> print(roughset)
```

```
$lower.approximation
$lower.approximation$Accept
        MBA High No Good MBA Medium Yes Excellent
                       4                         1
  MSc High Yes Excellent     MSc High Yes Neutral
                       3                         2
```

```
$lower.approximation$Reject
   MBA Low Yes Neutral   MCE Low No Excellent       MCE Low Yes Good
                     5                      8                      6
MSc Medium Yes Neutral
                     7


$upper.approximation
$upper.approximation$Accept
        MBA High No Good MBA Medium Yes Excellent
                       4                         1
  MSc High Yes Excellent     MSc High Yes Neutral
                       3                         2

$upper.approximation$Reject
   MBA Low Yes Neutral   MCE Low No Excellent       MCE Low Yes Good
                     5                      8                      6
MSc Medium Yes Neutral
                     7


$type.model
[1] "RST"

attr(,"class")
[1] "LowerUpperApproximation" "list"
```

Based on Fuzzy Rough Set Theory

This example uses the dataset *pima7.dt* which contains seven objects of the pima dataset [11]. It is included in the "RoughSets" package. Therefore, we load it with:

```
R> data(RoughSetData)
R> pima7.dt <- RoughSetData$pima7.dt
```

As in the RST case, we need to compute the indiscernibility relation, approximations and the positive region in sequential order. First, we calculate the indiscernibility relation of the considered conditional and decision attributes. For example, in this case we consider the second and third attributes only (*condAttr*) and the last column as the decision attribute (*decAttr*).

```
R> condAttr <- c(2, 3)
R> decAttr <- ncol(pima7.dt)
```

Then, we need to assign values of the *control* parameter. For the indiscernibility on the conditional attributes (the second and third attributes only), we set *lukasiewicz* and *eq.1* to be our t-norm and relation, respectively. Detailed descriptions of the parameters can be found in the package manual.

```
R> control.ind <- list(type.aggregation =
+     c("t.tnorm", "lukasiewicz"),
+     type.relation = c("tolerance", "eq.1"))
R> IND.condAttr <- BC.IND.relation.FRST(pima7.dt,
+     attributes = condAttr, control = control.ind)
```

Since our data have nominal values in the decision attribute, we assign *crisp* for the type of aggregation and relation to calculate the indiscernibility relation on the decision attribute.

```
R> control.dec <- list(type.aggregation = "crisp",
+     type.relation = "crisp")
R> IND.decAttr <- BC.IND.relation.FRST(pima7.dt,
+     attributes = decAttr, control = control.dec)
```

After obtaining the relations on the conditional and decision attributes, we can calculate the lower and upper approximations as:

```
R> control <- list(t.implicator = "lukasiewicz")
R> imp.tnorm <- BC.LU.approximation.FRST(pima7.dt,
```

```
+     IND.condAttr, IND.decAttr,
+     type.LU = "implicator.tnorm", control = control)
```

It can be seen that in this case we use *implicator.tnorm* proposed by [238] with *lukasiewicz* as the implicator. The *imp.tnorm* object is a list showing each index of objects included in lower and upper approximations based on decision concepts.

The positive region and degree of dependency can be obtained with:

```
R> region <- BC.positive.reg.FRST(pima7.dt, imp.tnorm)
```

Finally, we can construct the decision-relative discernibility matrix. For example, we use the following parameters.

```
R> control.1 <- list(type.relation = c("tolerance", "eq.1"),
+     type.aggregation = c("t.tnorm", "min"),
+     t.implicator = "lukasiewicz", type.LU = "implicator.tnorm")
```

The detailed explanation can be found in the manual. Then, we construct the matrix based on the *standard.red* algorithm which is based on [282].

```
R> disc.Mat <- BC.discernibility.mat.FRST(pima7.dt,
+     type.discernibility = "standard.red",
+     control = control.1)
```

As we mentioned above, the user can also define his or her own functions. For example, for constructing the indiscernibility relation we create the function *FUN.average* to be our aggregation operator. *FUN.average* is defined as average of all data on each considered attribute. Using the same dataset, we calculate the indiscernibility relation *IND.custom* of the second and third attributes as follows:

```
R> FUN.average <- function(data){
+  return(Reduce("+", data)/length(data))
+  }
R> control.ind <- list(type.aggregation = c("custom", FUN.average),
+     type.relation = c("tolerance", "eq.1"))
```

```
R> IND.custom <- BC.IND.relation.FRST(pima7.dt,
+     attributes = condAttr, control = control.ind)
```

For calculating approximations, as in the indiscernibility relations, we can build new models. For example, we design a new model based on the OWA model by defining our own weight vectors (*w.vector*) [56].

```
R> w.vector <- matrix(0, nrow = nrow(pima7.dt))
R> m.owa <- round(nrow(pima7.dt)/2)
R> for (i in 1 : m.owa){
+     w.vector[i] <- (2^(m.owa - i))/(2^m.owa - 1)
+  }
```

Then, we assign *lukasiewicz* as the type of implicator and t-norm.

```
R> control <- list(t.implicator = "lukasiewicz",
+  t.tnorm = "lukasiewicz", w.owa = w.vector)
```

We use the same dataset and the conditional and decision relations: *pima7.dt*, *IND.condAttr*, and *IND.decAttr*. Lastly, we calculate the approximation by the following command:

```
R> owa.custom <- BC.LU.approximation.FRST(pima7.dt,
+  IND.condAttr, IND.decAttr, type.LU = "owa",
+  control)
```

To display the approximations *owa.custom*, we can execute:

```
R> print(owa.custom)

$fuzzy.lower
$fuzzy.lower$1
        1         2         3         4         5         6         7
1.0000000 0.3785507 0.6765217 0.3785507 0.9339130 0.9953623 1.0000000

$fuzzy.lower$2
```

```
        1         2         3         4         5         6         7
0.2291304 0.9026087 0.2856522 0.7055072 0.3124638 0.1307246 0.1442029


$fuzzy.upper
$fuzzy.upper$1
        1         2         3         4         5         6         7
0.7708696 0.0973913 0.7143478 0.2944928 0.6875362 0.8692754 0.8557971

$fuzzy.upper$2
          1           2           3           4           5
0.000000000 0.621449275 0.323478261 0.621449275 0.066086957
          6           7
0.004637681 0.000000000


$type.LU
[1] "owa"

$type.model
[1] "FRST"

attr(,"class")
[1] "LowerUpperApproximation" "list"
```

It can be seen that the *owa.custom* object is a list containing the lower and upper approximations separated by the concept classes. Additionally, other descriptions, such as *type.LU*, are presented as well.

Other examples showing the use of custom functions can be seen in the package manual and on our website. The manual has been designed to show how all approaches are executed with complete related parameters.

4.3.4 An Example using Rule-Based Classifiers

In this example, we only consider implementations based on RST for prediction while examples regarding FRST can be found in the manual at `cran.r-project.org/package=RoughSets`. The example uses a real dataset which is the *wine* data included in the package.

Figure 4.6 shows processes conducted in the example. Basically, we first construct the *DecisionTable* object. Then, we do discretization, feature selection, and learning using a rule-based classifier. Finally, we perform prediction over new data. In this case, we focus on executing functions based on RST.

First, we load the *wine* data from the package:

```
R> data(RoughSetData)
R> dataset <- RoughSetData$wine.dt
```

For simplicity, we divide the data into training and testing data in the following way. After shuffling the data, the training data called *wine.decTable* are 80% of all data while the rest is the testing data (*tst.wine*).

```
R> dt.Shuffled <- dataset[sample(nrow(dataset)),]
R> idx <- round(0.8 * nrow(dt.Shuffled))
R> wine.decTable <- SF.asDecisionTable(dt.Shuffled[1 : idx, ],
+     decision.attr = 14, indx.nominal = 14)
R> tst.wine <- SF.asDecisionTable(dt.Shuffled[(idx +
+     1):nrow(dt.Shuffled), -ncol(dt.Shuffled)])
R> real.val <- dt.Shuffled[(idx + 1):nrow(dt.Shuffled),
+     ncol(dt.Shuffled), drop = FALSE]
```

Since the wine dataset contains real-valued attributes, we need to discretize them. For example, we use the *global.discernibility* method. Then we obtain cut values (called *cut.values*) by the following command:

```
R> cut.values <- D.discretization.RST(wine.decTable,
+     type.method = "global.discernibility")
```

Then we apply the cut values to the training and testing data to generate new decision tables (called *d.tra* and *d.tst*):

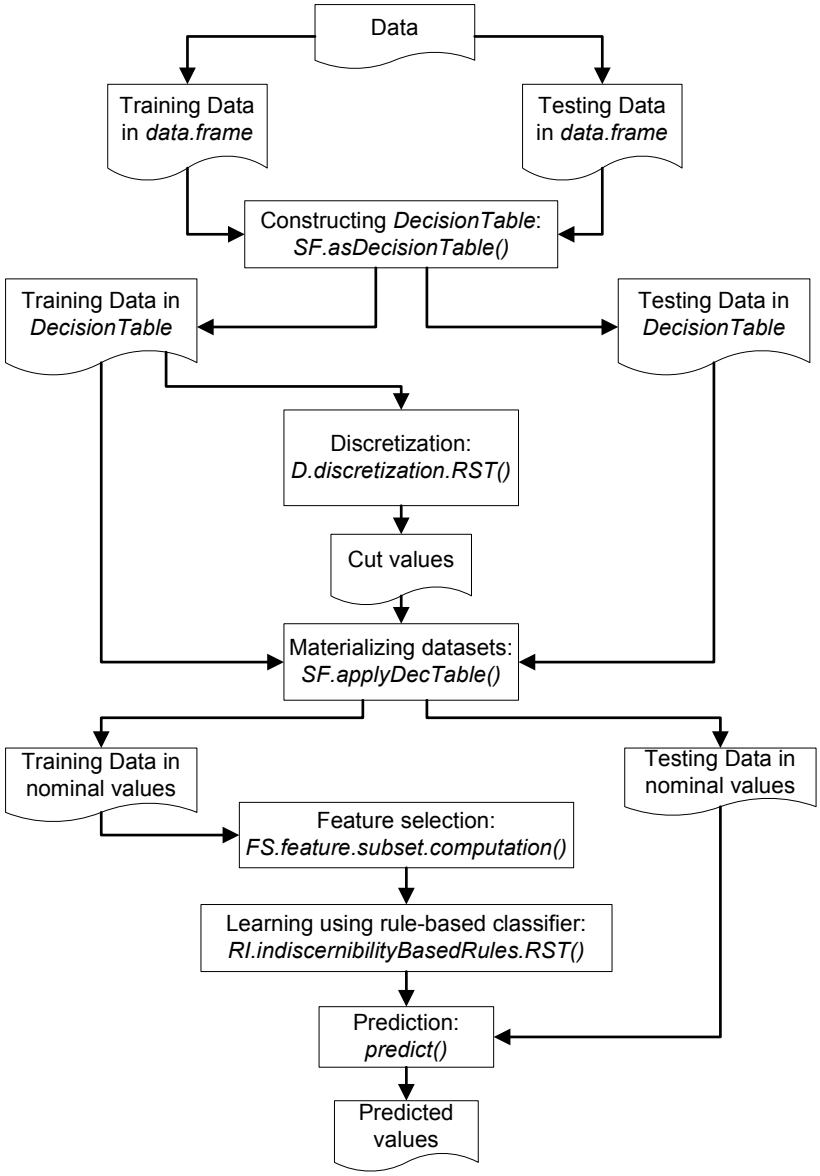


Figure 4.6: The workflow of data analysis using a rule-based classifier based RST.

```
R> d.tra <- SF.applyDecTable(wine.decTable, cut.values)
R> d.tst <- SF.applyDecTable(tst.wine, cut.values)
```

Now, we have the datasets containing nominal values.

We perform feature selection using the *quickreduct.rst* method as follows:

```
R> red.rst <- FS.feature.subset.computation(d.tra,
+    method = "quickreduct.rst")
```

A new decision table can be generated from the decision reduct *red.rst* with the following command:

```
R> fs.tra <- SF.applyDecTable(d.tra, red.rst)
```

After data preprocessing, a rule-based classifier can be induced from the training data:

```
R> rules <- RI.indiscernibilityBasedRules.RST(d.tra, red.rst)
```

It should be noted that in this case, the function needs the decision reduct as input data. Then, we can predict the testing data (*d.tst*) by considering the rules as follows:

```
R> pred.vals <- predict(rules, d.tst)
```

The predicted values *pred.vals* can be compared with the real values *real.val* as follows:

```
R> err <- 100 * sum(pred.vals != real.val)/nrow(pred.vals)
R> cat("The percentage error = ", err, "\n")

The percentage error =  11.11111
```

Detailed output for these examples and additional output is available at the project web page.

In summary, the examples illustrate the way in which the "RoughSets" package facilitates data analysis. Many options of parameter values are available to obtain good solutions. The functionality for defining our own functions as parameter values provides flexibility and opportunities for researchers to extend current models.

4.3.5 An Example Using Instance-Based Classifiers

In this example, we illustrate how to construct a model based on instances and predict new data. As the previous example, it uses a real dataset which is *wine* data included in the package.

Figure 4.7 depicts the processes considered in the example. First, we construct the *DecisionTable* object, and then feature and instance selections are conducted. After that, we consider two methods of instance-based classifiers in FRST: *C.FRNN.FRST* and *C.FRNN.O.FRST*.

First, we load the *wine* data from the package:

```
R> data(RoughSetData)
R> dataset <- RoughSetData$wine.dt
```

For simplicity, we divide the data into training and testing data in the following way. After shuffling the data, the training data called *wine.tra* are 80% of all data while the rest is the testing data (*wine.tst*).

```
R> dt.Shuffled <- dataset[sample(nrow(dataset)),]
R> idx <- round(0.8 * nrow(dt.Shuffled))
R> wine.tra <- SF.asDecisionTable(dt.Shuffled[1 : idx, ],
+     decision.attr = 14, indx.nominal = 14)
R> wine.tst <- SF.asDecisionTable(dt.Shuffled[(idx + 1):nrow(dt.Shuffled),
+     -ncol(dt.Shuffled)])
R> real.val <- dt.Shuffled[(idx + 1):nrow(dt.Shuffled), ncol(dt.Shuffled),
+     drop = FALSE]
```

The following command is to obtain a decision reduct by performing feature selection. For example, we consider the method *quickreduct.frst* which is the algorithm quick reduct based on FRST.

```
R> reduct <- FS.feature.subset.computation(wine.tra,
+    method = "quickreduct.frst")
```

After that, we generate new decision tables of the training and testing data by considering *reduct*, as follows:

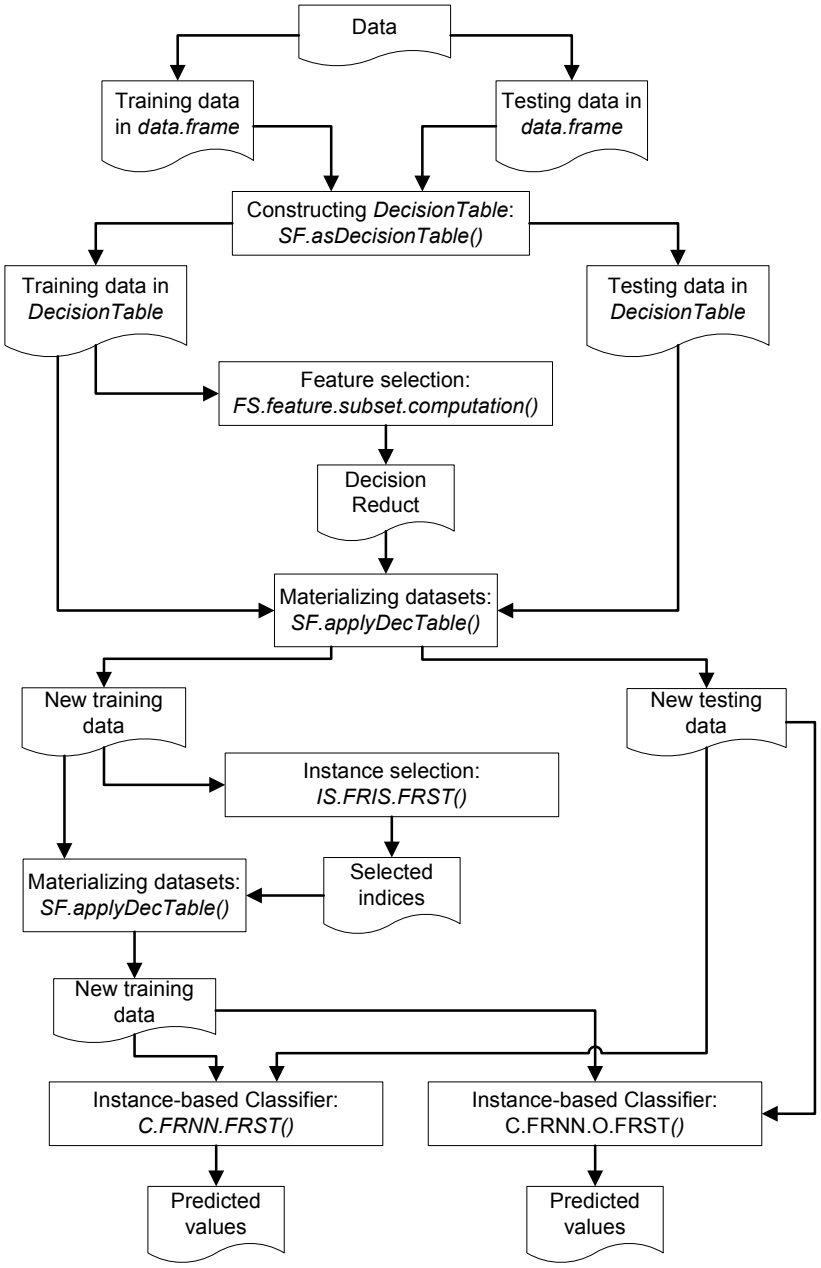


Figure 4.7: The workflow of data analysis using instance-based classifiers based FRST.

```
R> wine.tra.fs <- SF.applyDecTable(wine.tra, reduct)
R> wine.tst.fs <- SF.applyDecTable(wine.tst, reduct)
```

According to the scenario, we then perform instance selection as follows:

```
R> indx <- IS.FRIS.FRST(wine.tra.fs,
+     control = list(threshold.tau = 0.2, alpha = 1))
```

In this case, we use the method *IS.FRIS.FRST* with $\tau = 0.2$ and $\alpha = 1$. We materialize the decision table after instance selection by

```
R> wine.tra.is <- SF.applyDecTable(wine.tra.fs, indx)
```

Further, we perform learning and prediction by using two instance-based classifiers, which are *C.FRNN.O.FRST* and *C.FRNN.FRST*. Parameters in the methods are defined in the control arguments: *control.frnn.o* and *control.frnn*. Detailed description of the parameters can be seen in the manual [245].

```
R> ## Using C.FRNN.O.FRST
R> control.frnn.o <- list(m = 2, type.membership = "gradual")
R> predValues.frnn.o <- C.FRNN.O.FRST(wine.tra.is,
+     newdata = wine.tst.fs, control = control.frnn.o)
R> ## Using C.FRNN.FRST
R> control.frnn <- list(type.LU = "implicator.tnorm",k = 20,
+    type.aggregation = c("t.tnorm", "lukasiewicz"),
+    type.relation = c("tolerance", "eq.1"),
+    t.implicator = "lukasiewicz")
R> predValues.frnn <- C.FRNN.FRST(wine.tra.is,
+    newdata = wine.tst.fs, control = control.frnn)
```

We can compare the results by typing the following commands:

```
R> real.val <- dt.Shuffled[(idx+1):nrow(dt.Shuffled),
+    ncol(dt.Shuffled), drop = FALSE]
R> err.1 <- 100*sum(predValues.frnn.o!=real.val)/
```

```
+     nrow(predValues.frnn.o)
R> err.2 <- 100*sum(predValues.frnn!=real.val)/
+     nrow(predValues.frnn)
R> cat("The percentage error = ", err.1, "\n")

The percentage error =  2.777778

R> cat("The percentage error = ", err.2, "\n")

The percentage error =  0
```

4.4 A Comparison with Other Packages

This section reviews some existing software libraries implementing RST and FRST. A comparison according to the functionality and capability of other exemplary software systems is shown in Table 4.8. In particular, we compare the "RoughSets" library with the following four libraries:

Rough Set Data Explorer (ROSE) It is a software developed by the Laboratory of Intelligent Decision Support Systems of the Institute of Computing Science, Poznań Technical University. It is used to implement basic elements of the rough set theory and rule discovery technique [225, 226]. It allows to apply the variable precision rough set defined by Ziarko [312] and the classical model by Pawlak [213] for constructing the approximations. Both ROSE and "RoughSets" allow the user to perform the basic concepts of RST. For data processing, it provides a discretization method based on an entropy measure proposed by Fayyad and Irani [74]. Feature selection is done by algorithms based on the lattice search introduced by Romański [247] and the decision-relative discernibility matrix in [258]. Some algorithms are implemented to generate rules, such as LEM2 [95, 162] and the explore algorithm [194]. The software implements the L-metric or valued closeness relation based on [261] to predict new data. Even though ROSE provides more algorithms for rule induction than "RoughSets," in the case of discretization and feature selection "RoughSets" offers more functionality.

Rough Set Exploration System (RSES) It is a toolset for analyzing data implemented at the Group of Logic, Institute of Mathematics, University of Warsaw and the Group of Computer Science, Institute of Mathematics, University of Rzeszów, Poland [25]. It has an associated Java library called RSESlib. There are some algorithms included for handling data processing (e.g., missing value completion, discretization, feature selection, decomposition), rule induction, and classifiers based on nearest neighbor algorithms. For handling missing values, it provides four approaches: object removal, filling missing parts [99], analysis of data and treating the missing data as information. The discretization task is performed by algorithms proposed in [204]. Besides determining reducts based on exhaustive and genetic algorithms [23], it has implemented feature selection based on dynamic reducts [23, 22]. Furthermore, decomposition of tables and creation of new attributes are considered in this package as well. Implementation of the covering with reducts approach [300] and LEM2 [96] is used to generate rules. Although the following classifiers are not based on rough set theory, classifiers based on nearest neighbors [69] and local transfer function [298] are implemented as well in the package. It is clear that RSES is a comprehensive software, especially for tackling data preprocessing. However, it does not provide facilities for users to operate the basic concepts of RST as in "RoughSets."

Rough Set Toolkit for Analysis of Data (ROSETTA) It is a software package developed by a collaboration of researchers working in the Knowledge System Group at the Norwegian University of Science and Technology (NTNU), Norway, and the Group of Logic, University of Warsaw, Poland. It is an advanced system for data analysis based on RST [209, 210]. It consists of two parts which are a computational kernel and a graphical user interface (GUI) front end. Even though ROSETTA implements some algorithms as its computational kernel, it also uses the RSES library. For example, in discretization algorithms, it considers algorithms of boolean reasoning [206], entropy measure [67, 75] and χ-statistics [154, 174]. From a front end view, it provides a user friendly interface and allows to import datasets from external data sources, e.g., database management systems (DBMS), via the open database connectivity (ODBC) interface. Additionally, we can export results, e.g., decision rules, reducts, etc., to various formats such as XML, C++ and Prolog. In summary, the advantages of this software are that it provides a user friendly interface based on GUI and a connection with databases. "RoughSets" offers other benefits. For example, the consistency

and reliability of command line facilities for batch processing and heavy experimentation. Furthermore, R as the scientific environment provides many other packages implementing the connection with databases and various file formats.

Waikato Environment for Knowledge Analysis (WEKA) It is a popular framework for data mining tasks such as data preprocessing, classification, regression, clustering, association rules and visualization [109]. It provides a sophisticated GUI containing four menus: *Explorer*, *Experimenter*, *KnowledgeFlow*, and *SimpleCLI*. Even though there are many packages included in WEKA, in this survey we focus on a package implementing FRST in WEKA. It was created by Jensen *et al.* [136]. For feature selection, it has implemented several methods, for example fuzzy QuickReduct using parameters γ [55], boundary region [143], δ [143], VQRS [54], and fuzzy discernibility matrix approach [144]. By employing search methods such as ant colony optimization (ACO) [142] and particle swarm optimization (PSO) [293], reducts can be generated. Fuzzy-rough instance selection is deployed to select qualified objects in the instance selection task [138]. Lastly, it also provides classifier methods based on nearest neighbors [137] and rule induction using hybrid fuzzy-rough rule induction and feature selection (called QuickRules) [140]. Some algorithms of feature selection, rule induction, and nearest neighbor-based classifier in the package are also implemented in "RoughSets." However, especially for constructing the approximations, in "RoughSets" we provide not only more approaches but also custom functions for some parameters. In addition, the R environment provides the possibility to convert the *data.frame* format from/to the WEKA Attribute-Relation File Format (ARFF) by employing the "foreign" package [233].

In summary, it can be seen that "RoughSets" offers some advantages that are not shared by others, such as the integration RST and FRST, a complete algorithms for missing value handling, discretization, feature selection, instance instance, rule-based classifiers, and instance-based classifiers, the facilities to build own models (i.e., the lower and upper approximations) by defining customized functions.

Table 4.8: A comparison of "RoughSets" with other packages (Part: 1 of 2).

Components	**ROSE**	**RSES**	**ROSETTA**	**WEKA**	**"RoughSets"**
General views:					
Theory	Rough set (RST)	RST	RST	Fuzzy rough set (FRST)	RST and FRST
Programming Language (mainly)	C++	C++ and Java	C++	Java	R
Operating System	MS Windows	MS Windows and Linux	MS Windows	MS Windows, Mac Os X, and Linux	MS Windows, Mac OS X, Linux, and Solaris
User Interface	Graphical User Interface (GUI)	GUI and scripting interface	GUI	GUI	Scripting interface
License	Use only for research, education, development, private, and non profit purposes	Freely distributed for non-commercial purposes	Use only for non-commercial purposes, a partially restricted use for algorithms of RSES library	The GNU General Public License	The GNU General Public License
Available features:					
Basic concepts	Yes	No	No	No	Yes
Discretization	Yes	Yes	Yes	No	Yes
Feature selection	Yes	Yes	Yes	Yes	Yes
Instance Selection	No	No	No	Yes	Yes
Missing value completion	Yes	Yes	Yes	Yes	No
Decomposition	No	Yes	No	No	No
Rule-based classifiers	Yes	Yes	Yes	Yes	Yes
Nearest neighbor-based classifiers	No	Yes	No	Yes	Yes
Cross validation	Yes	Yes	Yes	Yes	No
Algorithms of the basic concepts:					
Relations	Equivalence [213], similarity [262, 263]	Equivalence [213]	Equivalence [213]	Jensen [143]	**RST:** equivalence [213]; **FRST:** Naessend [200], Jensen [143], and kernel [119]
Approximations	Pawlak [213], Ziarko [312]	Pawlak [213], Ziarko [312]	Pawlak [213], Ziarko [312]	implicator/t-norm [238], VQRS [53], OWA [56]	**RST:** Pawlak [213]; **FRST:** implicator/t-norm [238], VQRS [53], OWA [56], FVPRS [309], SFRS [118], RFRS [120], and β-PFRS [249]
Positive Region	Pawlak [213]	Pawlak [213]	Pawlak [213]	Pawlak [213]	Pawlak [213]
Discernibility Matrix	Skowron [258]	Skowron [258]	Skowron [258]	Fuzzy discernibility matrix approach [144]	**RST:** Skowron [258]; **FRST:** Tsang [282], Zhao [309], and Chen [46, 45]

Table 4.9: A comparison of "RoughSets" with other packages (Part: 2 of 2).

Components	**ROSE**	**RSES**	**ROSETTA**	**WEKA**	**"RoughSets"**
Algorithms of applications:					
Discretization	Entroy measure [74]	Nguyen [204]	Nguyen [204], Boolean reasoning [206], entropy [67, 75], and χ-statistics [154, 174]	-	**RST:** maximal & local [23], global [205], and unsupervised [67]
Feature Selection	Lattice search [247], discernibility matrix [258]	Discernibility matrix [258], genetic algorithm [23], and dynamic reducts [23, 22]	Discernibility matrix [258], genetic algorithm [23], and dynamic reducts [23, 22]	Fuzzy QuickReduct γ [55], boundary region [143], δ [143], VQRS [54], fuzzy discernibility matrix approach [144], ACO [142], and PSO [293]	**RST:** QuickReduct [254], the greedy heuristics [301, 260, 134], permutation [133], discernibility matrix [258]; **FRST:** fuzzy QuickReduct [141, 249, 30, 54, 143, 309, 118, 56, 120], the near-optimal [309]
Rule Induction	LEM2 [95, 162], explore [194]	Covering reducts [300] and LEM2 [96]	Covering reducts [300] and LEM2 [96]	QuickRules [140]	**RST:** indiscernibility based rules [213]; **FRST:** QuickRules [140] generalized [310]
Instance Selection	-	-	-	FRIS [138]	**FRST:** FRIS [138] and FRPS [285]
Nearest Neighbor Classifiers	-	-	-	FRNN [137, 139]	**FRST:** FRNN [139], FRNN.O [251], and POSNN [286]

4.5 Summary

In order to fullfil Objective 1(b), this chapter has discussed the following points:

1. the "RoughSets" package, which is an R package integrating implementations of various algorithms based on RST and FRST for dealing with missing value, discretization, feature selection, instance selection, rule-based classifiers, and nearest neighbor-based classifiers,
2. the package architecture and implementation details explaining important components included in the package,
3. some guidance and examples for end users in order to use the package,
4. a comparison with other tools related to the concepts.

The "RoughSets" package is available in CRAN: `http://cran.r-project.org/package=RoughSets` and in the project website: `http://dicits.ugr.es/software/RoughSets/`. Additionally, a journal paper describing the package has been published:

> L.S. Riza, A. Janusz, C. Bergmeir, C. Cornelis, F. Herrera, D. Ślęzak, and J.M. Benítez. Implementing Algorithms of Rough Set Theory and Fuzzy Rough Set Theory in the R Package. Information Sciences, 2014, 287, 68-98.

Chapter 5

The "SparkFernTreeR" Package

This chapter aims to address the third objective, which is to develop an R package for Big Data processing based on RFs and RFe. The package is called "SparkFernTreeR". First, we introduce the "SparkFernTreeR" package, and then present the package architecture and implementation details. Some examples showing the usage of the package and a comparison with other tools are also provided. Finally, a short summary is presented in the last section.

5.1 Introduction

Recently, data exploding in high volume, speed, and variety offer challenges to data scientists since traditional tools cannot handle them efficiently. Data fulfilling these dimensions is termed Big Data. Basically, two main problems should be considered in Big Data processing: storage management systems and programming models. In Big Data frameworks, storage systems should be able to read, write, and manage Big Data efficiently besides providing fault tolerance, availability, consistency, scalability, and heterogeneity. From the computational-model perspective, we need to tackle data in parallel and distributed ways. These strategies should also consider several aspects, such as fault tolerance and recovery, task scheduling, etc.

Furthermore, to prevent network congestions instead of bringing data into applications, now we need to spread and execute programs in the place where data are.

There are two well-known Big Data frameworks: Apache Hadoop and Apache Spark. First, Apache Hadoop offers three main functionalities: HDFS for the distributed file systems, MapReduce for the programming model, and YARN for the resource management system. Apache Spark aims to extend and generalize the MapReduce paradigm, which provides computations in memory by employing the RDD abstraction. Apache Spark provides the following modules: SparkR, SparkSQL, Spark Streaming, GraphX, and MLlib. These frameworks have been attracting other researchers to develop other systems tackling specific problems on top of them.

Since the connection between R and Big Data platforms is an emerging topic no state-of-the-art overview could be found and we have gone through the topics thoroughly to develop an extensive overview. It can be found in Section 1.9.5. By focusing on R tools working with Apache Hadoop and Apache Spark only, a survey in Figure 1.18 shows that these exists some packages that are used to general and specific purposes. Basically, these packages attempt to resolve the drawbacks of R. The main drawback is that data are loaded into random access memory (RAM) before being processed by R. It means that the active data must be smaller than available RAM. Moreover, R is designed to work on single-threaded and standalone framework.

While some packages in R can be found to allow parallel and distributed computation, none of them offer the additional benefits of current Big Data platforms, such as scalability, fault tolerance, task scheduling, etc. Therefore, we aim to propose an R package to fulfill this hole so that the R community can directly use it for their tasks without programming from scratch. It calls the "SparkFernTreeR" package.

The "SparkFernTreeR" package is an R package implementing the following methods: DTs, RFs, and RFe, for dealing with Big Data. Actually, there are two modes for running the package: standalone/single machine and cluster mode. While on the standalone mode it can be executed in the R environment as the regular packages, we employ the Apache Spark framework in order to work on the cluster mode. In order to connect R and Apache Spark, we utilize the R package "SparkR". Furthermore, for storage management systems, "SparkFernTreeR" can load and save files from/to HDFS and other systems that are allowed by Apache Spark.

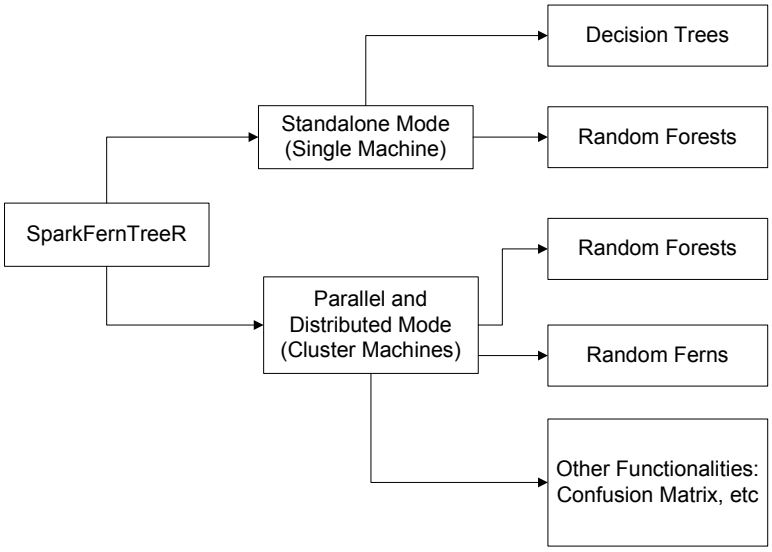


Figure 5.1: Main modules included in the "SparkFernTreeR" package.

Figure 5.1 shows a global feature included in the package as follows:

- Running on standalone/single machine:
  - Decision trees: As depicted in Section 1.7.1, DT is a basic algorithm used for building RFs. In this package, it can be used for handling classification, regression, clustering, and manifold learning tasks.
  - Random Forests: Based on the methods, we allow to deal with classification, regression, clustering, manifold learning/feature extraction problems. Furthermore, in this package, several parameters for setting a model, such as a number of trees, a number of features, maximum depth of trees, entropy methods, and splitting methods on nodes, are provided.
- Running on cluster mode (by using Apache Spark):
  - Random forests: Basically, RFs in this case are an extension of RFs on a single machine. The same as RFs on standalone, we allow to use them for handling classification, regression, clustering, and feature extraction.

- Random ferns: The RFe method is implemented by including "rFerns" [164, 212] into the package for dealing with Big Data analysis.
- Other utilities for experiments: We also embed the following functionalities for experiments:
  * Generating k-fold cross validation. In fact, this feature is useful when performing experiments with Apache Spark using not massive datasets so that we need to do cross validation on these data.
  * Calculating confusion matrix and other measures for classification adopted from [165]: OSR, TPR, TNR, PPV, NPV, F-measure, JCC, and ICSI. It is only used for classification tasks.
  * Calculating accuracy measures for regression (i.e., MSE and RMSE). It can be used for regression tasks.

It can be seen that instead of employing R packages available in CRAN for implementing RFs, we develop a new framework that attempts to unify RFs for dealing with several tasks (i.e., classification, regression, clustering, and manifold learning) by adopting the study in [58]. However, For the RFe method, we utilize the "rFerns" package for Big Data processing, so that users refers to the package when working on the standalone mode.

Furthermore, as the "SparkR" package, "SparkFernTreeR" gain the same benefits, such as running on Apache Hadoop, Mesos, and standalone, accessing input data from HDFS, Cassandra, HBase, and S3, supporting fault tolerance, etc.

5.2 The Package Architecture and Implementation Details

The "SparkFernTreeR" package is built on top of "SparkR" for parallel and distributed computing as shown in Figure 5.2. Therefore, before working with the package on the cluster mode, the "SparkR" must be available first. Moreover, if we only work on the R environment, it is not necessary to install the "SparkR" package.

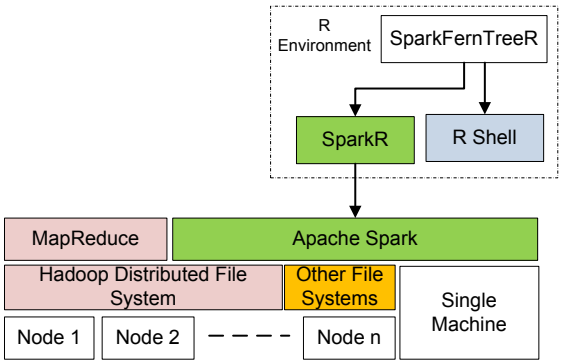


Figure 5.2: The general architecture of "SparkFernTreeR."

Table 5.1 and 5.2 show that there are three main parts in the package: learning, prediction, and other utilities. Learning aims to build a model that extracts knowledge from training data, whereas prediction is executed for obtaining predicted values based on the models. Other utilities provides some functions for helping in experimental phases.

As we mentioned above, R can only work over data in main memory. Therefore, we need to pay attention to the size of the following data when using the package: training, model, testing, and result/predicted values. Furthermore, in order to optimize the performance, there are several scenarios that should be taken into account, as follows:

- On the learning step: There are two possibilities whether training data are big or small. Therefore, the scenarios are as follows:

    1. Big training data: In this case, we have to parallelize the data through worker nodes as shown in Figure 5.3. Learning functions invoked in this case are *SparkRF()* for RFs and *SparkFerns()* for RFe. The output, which is a model, can be emitted as an R object, but we would recommend to produce it in an RDD object by defining *outputFile*. Furthermore, we also provide another algorithm to bootstrap datasets, named the Bag of Little Bootstraps (BLB) [157].
    2. Small training data: This is just provided for convenience of the users. While there are already available packages for regular ("small") datasets processing, by incorporating this functionality in the Big Data package, the user familiar with the

Table 5.1: Part 1: The functions included in "SparkFernTreeR."

Functions	Descriptions
Learning:	
decTree()	It is used to construct a DT model.
randForest()	It is used to construct a RF model.
SparkRF()	It is used to construct an RF model on Apache Spark.
SparkFerns()	It is used to construct an RFe model on Apache Spark.
Prediction:	
predict.decTree()	It is a method for prediction based on the DT method.
predict.randForest()	It is a method for prediction based on the RF method.
predict.SparkRF()	It is a method for prediction based on the RF method using Apache Spark.
predict.SparkFerns()	It is a method for prediction based on the RFe method using Apache Spark.
predictSparkForest()	It is a function used for prediction based on the RF method where the input are the RDD objects: *objectRDD* for the RF model and *newdataRDD* for new data.
predictSparkFerns()	It is a function used for prediction based on the RFe method where the input are the RDD objects: *objectRDD* for the RFe model and *newdataRDD* for new data.

Table 5.2: Part 2: The functions included in "SparkFernTreeR."

Functions	Descriptions
Other utilities:	
genCrossValidation()	It is used to generate k-fold cross validation.
sparkConfusionMatrix()	It is used to calculate confusion matrix of classification on big data.
sparkRegressionAcc()	It is used to calculate accuracy of regression on big data.
convertToObjectFile()	It is used to convert input data from text files into RDD objects.
splitTestingData()	It is used to get input and real data separately.

"SparkFernTreeR" package does not need to execute other packages for this. Moreover, we can also execute functions included in the standalone mode (i.e., *randForest()* and *decTree()*) besides *SparkRF()*. However, it should be noted that for RFe on the standalone mode, users refer to the "rFerns" package.

- On the prediction step: After doing the learning step, we predict new/testing data by considering the obtained model. In this step, mainly we need to consider the sizes of model and testing data. The following are possible scenarios:

  1. Big testing data and big model: As shown in Figure 5.4, both testing data and model will be divided according to a number of partitions. Then, a block of testing data broadcasts through all worker nodes. After that, prediction functions compute values based on the data and model. Predicted values of a block testing data are produced by aggregating all block of models. To perform this scenario, we need to execute *predictSparkForest()* for RFs and *predictSparkFerns()* for RFe. The resulting model or outputs can be saved in two alternative formats: an RDD object or an R object. The first one should be chosen if we suspect that the output is also big data.
  2. Big testing data and small model: In this case, testing data are divided based on a number of partitions and model is broad-

casted through worker nodes. Predicted values are calculated by aggregating results of all worker nodes. The diagram can be seen in Figure 5.5. To use this scenario, we just call the method *predict.SparkRF()* for RFs and *predict.SparkFerns()* for RFe by defining the parameter *isBigData* to be *TRUE*. Of course, in this case we can also choose the first scenario.

3. Small testing data and big model: It is also a common scenario when we have small testing data but huge numbers of trees because of big training data. Two parts that make this scenario shown by Figure 5.6 with the previous one are (i) in this schema we split model and broadcast testing data through worker nodes and (ii) here the aggregation function is not only to collect but also recapitulate results of each model. This scenario can be done by *predict.SparkRF()* and *predict.SparkFerns()* with *isBigData = FALSE*.

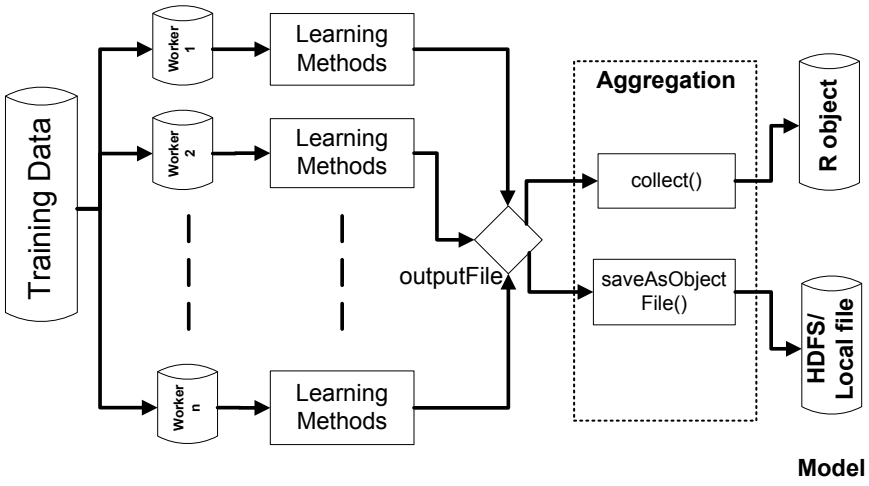


Figure 5.3: Big Data processing on learning: The scenario "big training data."

It should be noted that the terms small and big in the above scenarios can be defined based on whether the data are less or greater than the amount of memory (RAM).

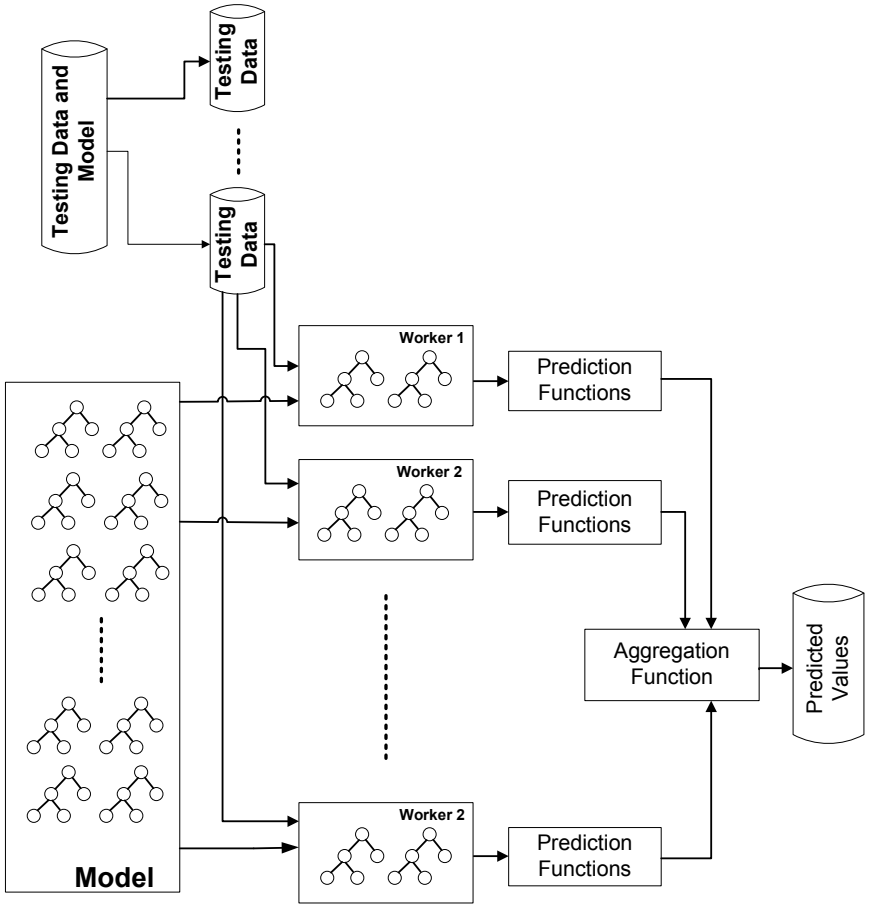


Figure 5.4: Big Data processing on prediction: The scenario "big testing data" and "big model."

5.3 Examples of Usage

In this part, we will explain several steps in order to use the package. First, we present a way to install and load the "SparkFernTreeR" package.

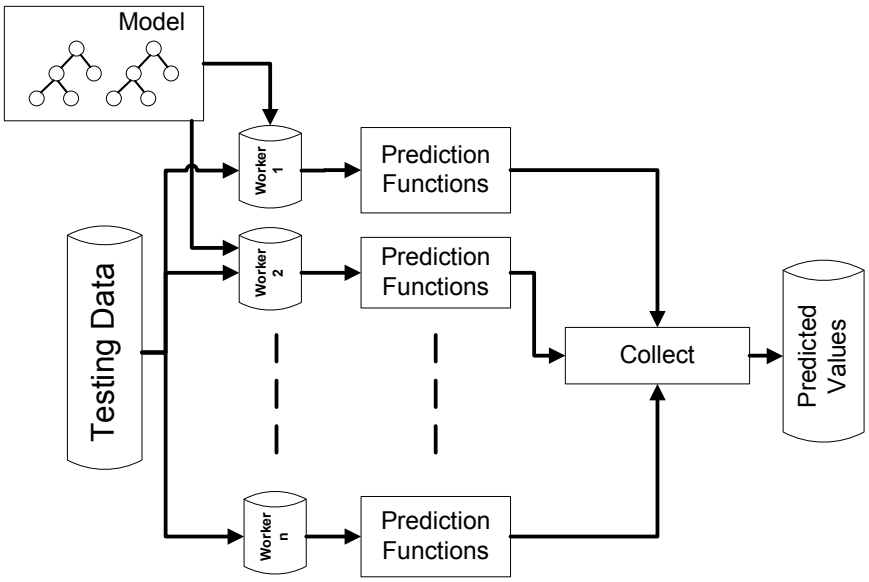


Figure 5.5: Big Data processing on prediction: The scenario "big testing data" and "small model."

5.3.1 Installation and Loading the "SparkFernTreeR" Package

To install "SparkFernTreeR" from CRAN, users simply enter the following command in the R environment:

```
R> install.packages("SparkFernTreeR")
```

The package installation has to be done only once. After that in any session using "SparkFernTreeR", we need to load it with the following command:

```
R> library(SparkFernTreeR)
```

The command loads the "SparkFernTreeR" package and makes its functions available in the R environment. We can see a list of functions included in "SparkFernTreeR" by typing the following command:

```
R> library(help=SparkFernTreeR)
```

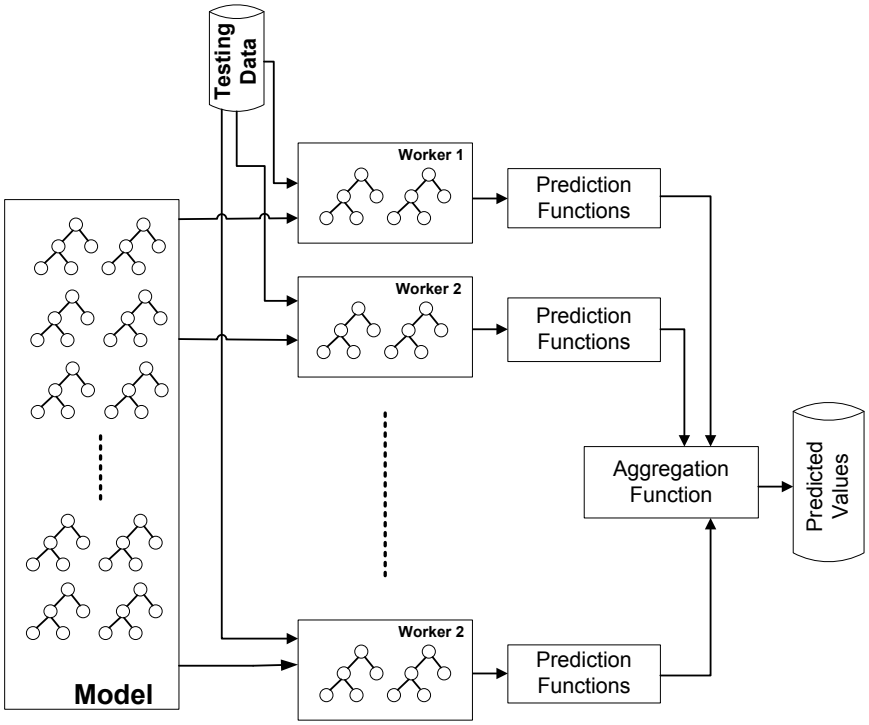


Figure 5.6: Big Data processing on prediction: The scenario "small testing data" and "big model."

All R functions available in "SparkFernTreeR" are documented in the R help system in a hypertext format, and in the package manual in pdf format. To get information on a particular function, we can call the help command, e.g.,

```
R> help(randForest)
```

If users use the package for dealing with big data on Apache Spark, the "SparkR" must be installed first. The following steps are an instruction to install the package: First, we need to clone the package from the github on the console:

```
> git clone https://github.com/amplab-extras/SparkR-pkg.git
```

Then, go to the directory SparkR-pkg, and execute the following command:

```
> ./install-dev.sh
```

It should be noted that SparkR by default uses Apache Spark 1.1.0. You can switch to a different Spark version by setting the environment variable SPARK_VERSION. For example, to use Apache Spark 1.3.0, you can run

```
> SPARK_VERSION=1.3.0 ./install-dev.sh
```

Furthermore, SparkR by default links to Hadoop 1.0.4. To use SparkR with other Hadoop versions, you will need to rebuild SparkR with the same version that Spark is linked to. For example to use SparkR with a CDH 4.2.0 MR1 cluster, you can run

```
> SPARK_HADOOP_VERSION=2.0.0-mr1-cdh4.2.0 ./install-dev.sh
```

5.3.2 Classification Using Random Forests with the Standalone Mode

For example, in this case we are using the iris dataset. First, we need to load the package:

```
R> library(SparkFernTreeR)
```

Then, prepare the data, as follows:

```
R> ## shuffle dataset
R> irisData <- iris[sample(nrow(iris)),]
R> ## set the first 120 rows to be training dataset
R> iris.tra <- irisData[1:120, ]
R> ## set testing dataset
R> iris.tst <- irisData[121:nrow(irisData), -ncol(irisData)]
R> ## get real values of testing dataset
R> real.iris <- irisData[121:nrow(irisData), ncol(irisData)]
```

After that, we construct models. In this example, two models are built, which as the DT and RF models.

```
R> ## construct a DT model
R> tree.iris <- decTree(iris.tra, typeTask = "classification",
```

```
+   controlTree = list(nameFeatures = NULL,
+   typeFeatures = c(1,1,1,1,0), paramPercent = 1,
+   typeEntropy = "ML", maxDepthTree = 20,
+   typeSplitting= "random", minNumData = 3))
R> ## construct a RF model
R> RF.iris <- randForest(iris.tra, typeTask = "classification",
+   controlRF = list(numTree = 20, numFeature = 2),
+   controlTree = list(typeFeatures = c(1,1,1,1,0),
+     paramPercent = 0.5,
+     typeEntropy = "ML", maxDepthTree = 20,
+     typeSplitting= "random",
+     minNumData = 3))
```

Prediction can be done through the following commands:

```
R> ## predict testing dataset using the DT model
R> res.test <- predict(tree.iris, iris.tst)
R> ## predict testing dataset using the RF model
R> res.test.RF <- predict(RF.iris, iris.tst)
```

Then, we can calculate the error:

```
R> ## error calculation
R> err.Tree = 100*sum(real.iris!=res.test)/length(real.iris)
R> err.RF = 100*sum(real.iris!=res.test.RF)/length(real.iris)
R> print("The result: ")

[1] "The result: "

R> print(err.Tree)

[1] 6.666667

R> print(err.RF)

[1] 6.666667
```

5.3.3 Classification Using Random Forest on Apache Spark

In this section, we explain the usage of the package for handling a classification task using RFs. In general speaking, the following are steps that should be taken into account:

1. **Installation of "SparkR":** It has been explained above. Furthermore, installation connecting with Apache Hadoop and other storage systems can be also found in the website.

2. **Loading "SparkR":** After installation, if we are working on single machine, we can load the package from the *SparkR-pkg* folder by:

   ```
   > ./sparkR
   ```

   Other ways to load the package, e.g., working on cluster and connecting with HDFS, can be read in the "SparkR" website.

3. **Installation of "SparkFernTreeR":** In order to install "SparkFernTreeR", we can do it from a local file or CRAN. For example, to install it from the local file *SparkFernTreeR_1.0-0.tar.gz*: in the "sparkR" shell, we execute the following command:

   ```
   R> install.packages("/home/lala/SparkFernTreeR_1.0-0.tar.gz",
   +   repos = NULL, type="source")
   ```

   Since the package requires "entropy" and "rFerns", we should install them previously.

4. **Load "SparkFernTreeR" and "SparkR":** As other R packages, they are loaded by:

   ```
   R> library(SparkR)
   R> library(SparkFernTreeR)
   ```

5. **Data preparation:** In order to execute the functions included in learning and prediction, data must be in RDD objects. So, if we have other formats, we should convert them first. In the package, we provide a function used for converting from text files to RDD objects, which is *convertToObjectFile()*. The return of the functions is an RDD object containing serialized R objects, which is a *list* consisting of *data.frame*. In other words, when we print the RDD object by calling *collect()* in

"SparkR", we have a *list* containing *data.frame*. For example, we have the training data in *iris.tra.txt* as follows:

```
R> trainRDD <- convertToObjectFile("iris.tra.txt",
+  controlSpark = list(master = "local[8]",
+   appName = "rForest", sparkHome = NULL),
+  controlBigData = list(
+   outputFile = "hdfs://hadoop-master:8020/xxxx/iris.tra",
+   numSlices = 4),
+  controlDataFile = list(nameFeatures = NULL, header = FALSE,
+   sep = " ", strip.white = TRUE))
```

The above command aims to convert the text file *iris.tra.txt* to an RDD file saved in HDFS: *hdfs://hadoop-master:8020/xxxx/iris.tra.5*. There are three control parameters, which are *controlSpark*, *controlBigData*, and *controlDataFile*. The *controlSpark* parameter is used to define Apache Spark configuration. For example, in this case we are working on the standalone mode by defining *master = "local[8]"*. Detailed description related this parameter can be seen in the "SparkR" website and the manual. Two elements in *controlBigData* are *outputFile* and *numSlices*, which are set the directory path of RDD objects and the number of partitions. We can also save the data into other system files (e.g., local file, Cassandra, etc). The last parameter is adopted from *read.delim()*, such as *header*, *sep*, etc.

As training data, we need to convert testing data as follows:

```
R> testRDD <- convertToObjectFile("iris.tst.txt",
+  controlSpark = list(master = "local[8]",
+   appName = "randForest", sparkHome = NULL),
+  controlBigData = list(
+   outputFile = "hdfs://hadoop-master:8020/xxxx/iris.tst",
+   numSlices = 4),
+  controlDataFile = list(nameFeatures = NULL, header = FALSE,
+   sep = " ", strip.white = TRUE))
```

Furthermore, it should be noted that testing data only contains input data/features. So, in the case testing data still involves the output data, we need to split the data by executing *splitTestingData()*.

If it is required to generate *k*-fold cross validation, we can also invoke *genCrossValidation()*.

6. **Learning:** As we mentioned in this step we have two scenarios. However, we focus on the first scenario on the learning step: big training data. The following command is an example on learning:

```
R> SparkRF("hdfs://hadoop-master:8020/xxxx/iris.tra",
+   typeTask = "classification",
+   controlSpark = list(master = "local",
+    appName = "randForest", sparkHome = NULL),
+   controlRF = list(numTree = 8, numFeature = 3),
+   controlTree = list(paramPercent = 0.5,
+    typeEntropy = "ML", maxDepthTree = 4,
+    typeSplitting = "ets", minNumData = 3),
+   controlBigData = list(
+    outputFile = "hdfs://hadoop-master:8020/xxxx/modIris",
+    numSlices = 4, numBLB = 4, useBLB = FALSE))
```

It can be seen that *SparkRF* has six parameters, as follows:

- *trainRDD*: In this case, it is obtained from HDFS: *hdfs://hadoop-master:8020/xxxx/iris.tra*. Other storage types supported by "SparkR" can be seen in the package website.
- *typeTask*: For *SparkRF*, there are four options of tasks: *"classification"*, *"regression"*, *"clustering"*, and *"manifold"*. In this example, we are performing the classification task.
- *controlSpark*: It is the same as the parameter used in *convertToObjectFile*. If the spark context has been running, we need to execute *sparkR.stop()* for setting a new configuration.
- *controlRF*: It contains parameters related to RFs, such as *numTree* and *numFeature* for defining a number of trees and a number of selected features.
- *controlTree*: It contains parameters related to decision tree. Detailed description regarding the parameters can be seen in the manual.
- *controlBigData*: It contains parameters used to configure an output file and a number of partitions. In this case, we define that the

output file will be saved in *hdfs://hadoop-master:8020/xxxx/modIris* and the number of partitions is 4. Furthermore, we can also set whether we are using the BLB algorithm or not.

Detailed description about the parameters can be seen in the manual.

7. **Prediction:** For example, in this case we are performing the first scenario on the prediction step by calling *predictSparkForest()* as follows:

```
R> predictSparkForest("hdfs://hadoop-master:8020/xxxx/modIris",
+   "hdfs://hadoop-master:8020/xxxx/iris.tst",
+     runSpark = TRUE,
+   controlSpark = list(master = "local",
+     appName = "rForestTest", sparkHome = NULL),
+   controlBigData = list(outputFile = "tstIris",
+     numSlices = 4))
```

By executing the above command, we save the return to *tstIris*, which is an RDD object saved in local file.

8. **Accuracy measurements:** After prediction new data, we can calculate the confusion matrix by the following command:

```
R> conList <- sparkConfusionMatrix("tstIris",
+   "hdfs://hadoop-master:8020/xxxx/iris.real",
+   controlSpark = list(master = "local",
+    appName = "randForest", sparkHome = NULL))
R> print(conList)
```

It should be noted that in this case the real data have been saved in HDFS: *hdfs://hadoop-master:8020/xxxx/iris.real*.

Additionally, we have provided some examples in the package or users can go to the project website: `http://dicits.ugr.es/software/SparkFernTreeR`.

5.3.4 Classification Using Random Ferns on Apache Spark

In this section, we explain the usage of the package for handling a classification task using RFe. Regarding steps that should be taken into account

are basically the same as the explanation in Section 5.3.3, especially for Step 1, 2, 3, and 5.

The following are steps to use RFe on Apache Spark that we begin with loading the related packages:

1. Load the related package: In this case, we also need to load the "rFerns" package as follows:

```
R> library(SparkR)
R> library(SparkFernTreeR)
R> library(rFerns)
```

2. **Data preparation:** Users can refers to Step 5 in Section 5.3.3.

3. **Learning:** As we mentioned in this step we have two scenarios. However, we focus on the first scenario on the learning step: big training data. The following command is an example on learning:

```
R> SparkFerns("hdfs://hadoop-master:8020/xxxx/iris.tra",
+        controlSpark = list(
+    master = "spark://hadoop-master:7077",
+    appName = "randFerns", sparkHome = NULL,
+    sparkEnvir=list(spark.executor.memory='45g')),
+   controlFern = list(ferns = 1000, depth = 10,
+    nameFeatures = c("Sepal.Length", "Sepal.Width",
+     "Petal.Length", "Petal.Width", "Species")),
+        controlBigData = list(
+    outputFile = "hdfs://hadoop-master:8020/xxxx/modIris",
+    numSlices = 3))
```

It can be seen that the *SparkFerns()* function has three the control parameters:

- *controlSpark:* It refers to the same parameter used on the *SparkRF()* function in Section 5.3.3.
- *controlFern:* It is a list containing the parameters related to construction ferns: *ferns*, *depth*, and *nameFeatures*. The first parameter expresses the number of ferns to build while *depth* represents

the depth of the ferns; it must be in 1–16 range. The last one is used to define names of attributes.

- *controlBigData:* It is used to define the output filename and the number of partitions.

Detailed description about the parameters can be seen in the manual and the project website.

4. **Prediction:** For example, in this case we are performing the first scenario on the prediction step by calling *predictSparkFerns()* as follows:

```
R> predictSparkFerns(
+    "hdfs://hadoop-master:8020/xxxx/modIris",
+         "hdfs://hadoop-master:8020/xxxx/iris.tst",
+   controlSpark = list("spark://hadoop-master:7077",
+    appName = "randFernTest", sparkHome = NULL),
+        controlBigData = list(outputFile = "tstIris"",
+    numSlices = 3))
```

It can be seen from the above command that we save the results to the local file *tstIris*.

5. **Accuracy measurements:** In this step, we can refers to Step 8 in Section 5.3.3.

Additionally, we have provided some examples in the package. The users can go to the project website: `http://dicits.ugr.es/software/SparkFernTreeR`.

5.4 Survey and Comparison with Other Software Libraries

In this section, we present a brief survey on software libraries that implement DTs, RFs, and RFe. These packages can be run on both standalone and cluster modes. Moreover, we compare the functionalities "SparkFernTreeR" with these packages as shown in Table 5.3.

In the R environment, several packages implementing RFs can be found. First, the "randomForest" package [169] focuses on implementing RFs for

dealing with classification and regression. In this package, we are allowed to define some parameters for constructing a model, such as a number of trees, missing value handling, and tuning procedures. Then, "bigrf" is an R package implementing RFs for classification and regression with optimization for performance and for handling of datasets that are too large to be processed in memory [170]. Moreover, in this package, there are two scenarios: growing trees in parallel on a single machine using the "foreach" package and constructing multiple forests in parallel on multiple machines, then merged into one. In order to manage the memory, "bigrf" utilizes the "bigmemory." Another package in R that implements RFs is "ipred" [222]. Besides being used for classification and regression, this package allows to deal with survival problems and resampling. Regarding the implementations of RFs in R, only the "bigrf" package is able to handle massive datasets as "SparkFernTreeR". By comparing with "bigrf", "SparkFernTreeR" offers other advantages, which are gained from the Big Data frameworks, such as the integration between distributed file systems and programming models, fault tolerance, scalability, etc.

Furthermore, from other platforms, there are many libraries running on a single machine. For example, RFs have been implemented in WEKA [109] in Java and the "scikit" package in Python [218]. Basically, these packages are quite similar to the implementations of RFs in R, and they are only used on the standalone mode.

Related to the Big Data processing, the following are packages implementing RFs:

- Mahout [211]: It is a software library containing many machine-learning algorithms that are used for Big Data on top of Apache Hadoop. In the library, there are several groups of tasks, such as collaborative filtering, classification, and clustering. For classification tasks, it provides RFs implemented with the MapReduce model. Furthermore, an extension of the implementation, which is used for imbalanced Big Data using RFs, has been proposed by the study [64].
- MLlib [189]: As explained above, it is a part of Apache Spark that implements a variety of machine-learning algorithms, such as linear support vector machine, k-means, DTs, RFs, classification and regression trees, etc. Because of MLlib integrated with Apache Spark, we allow to use it by using Java, Scala, and Python. For the implementation of RFs, it can be used for dealing with classification and regression tasks.

According to this survey, it can be seen that RFs have been implemented in well-known software libraries. However, an implementation of RFs used for Big Data processing in the R community is still missing. Therefore, the aim of this research is also to promote the implementation of RFs and RFe on top of the Big Data frameworks in the R community. In addition, the package "SparkFernTreeR" implements not only for Big Data processing on the cluster mode, but also for regular datasets on the standalone mode, where this feature is not available in Mahout and MLlib.

Table 5.3: A comparison of "SparkFernTreeR" with other packages.

Components	**randomForest [169]**	**bigrf [170]**	**ipred [222]**	**WEKA [109]**	**scikit [218]**	**Mahout [211]**	**MLlib [189]**	**SparkFernTreeR**
General views:								
References	[33]	[33]	[33]	[33]	[33]	[33]	[33]	[33, 58]
Prog. Languages	R	R	R	Java	Python	Java	Scala, Java, Python	R
Systems	Standalone	Parallel Computing	Standalone	Standalone	Standalone	Parallel Computing	Parallel Computing	Parallel Computing
Big Data Processing	No	Yes	No	No	No	Yes	Yes	Yes
Computational Models for Big Data	-	"foreach", "bigmemory"	-	-	-	Apache Hadoop (MapReduce)	Apache Spark (RDD)	Apache Spark (RDD)
Features:								
Tasks	Classification, Regression	Classification	Classification, Regression, Survival Problems, Resampling	Classification, Variable Importance	Classification, Regression	Classification	Classification, Regression	Classification, Regression, Clustering, Manifold Learning
Missing Value Handling	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Plotting Models	Yes	Yes	Yes	Yes	No	No	No	No
Availabilities of Paramaters:								
Number of Trees	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Number of Selected Features	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Maximum Depth of Tree	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Minimum Number of Data	Yes	Yes	No	Yes	Yes	No	Yes	Yes
Entropy Equations	No	No	Yes	No	Yes	No	Yes	Yes
Aggregation Functions	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Others:								
Cross Validation	Yes	No	Yes	Yes	No	No	No	Yes
Confusion Matrix	Yes	Yes	No	Yes	No	Yes	Yes	Yes
Outlier Handling	Yes	Yes	No	Yes	No	No	No	No
Mixed Variables	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

5.5 Summary

This chapter presents the implementation of the "SparkFernTreeR" package that fulfills the third objective. The package offers the following functionalities:

1. Implementations of the decision tree and random forest methods for dealing with classification, regression, clustering, and manifold learning on a single machine or the R environment.

2. Implementations of the random forest and random fern methods for Big Data analysis under Apache Spark. While random forests can be used for classification, regression, clustering, and manifold learning, random ferns is used for classification.

In this chapter, we also provide some example showing how to use the package.

The "SparkFernTreeR" package is available in CRAN: `http://cran.r-project.org/package=SparkFernTreeR` and in the project website: `http://sci2s.ugr.es/dicits/software/SparkFernTreeR`. Additionally, the journal paper describing the package is published:

> L.S. Riza, C. Bergmeir, B. Krawczyk, F. Herrera, and J.M. Benítez. "SparkFernTreeR": An R Package Based on Random Forest and Random Ferns for Big Data Analysis under Apache Spark. R Journal, 2015 (to be submitted).

Concluding Remarks

A. Summaries

To begin with, we should remark that all objectives set for this research have been successfully reached. The results of the research can be summarized as follows:

1. Implementing approaches based on fuzzy rule-based systems in the R package "frbs:" The package aims to construct FRBSs by learning from data with more than 10 well-known methods for handling classification and regression tasks. Furthermore, it allows human experts to build an FRBS model based on their knowledge. Besides three popular models: Mamdani, TSK, and FRBCS, various options of parameters are available as well, such as types of membership functions and conjunction, disjunction, and aggregation operators. The "frbs" package is available in CRAN: `http://cran.r-project.org/package=frbs` and in the project website: `http://sci2s.ugr.es/dicits/software/FRBS`.

2. Implementing approaches based on rough set theory (RST) and fuzzy rough set theory (FRST) in the package "RoughSets:" It provides the following functionalities: missing value handling, discretization, instance selection, feature selection, rule induction-based classifiers, and nearest neighbor-based classifiers. Moreover, for academic purposes, some functions used for performing the basic concepts of RST and FRST are available, such as indiscernibility relations, the lower and upper approximations, regions, and discernibility matrix. Users can also define their own functions for building the concepts. The "RoughSets" package is available in CRAN: `http://cran.r-project.org/package=RoughSets` and in the project website: `http://dicits.ugr.es/software/RoughSets/`.

3. Designing and implementing a universal representation framework of FRBS models, called frbsPMML: It is designated based on PMML by employing XML. The representation offers the following benefits: interoperability, reproducibility, interpretability, and flexibility. Additionally, the implementation of the representation has been embedded in the package "frbs" so that we allow to produce and consume FRBS models to/from the frbsPMML format. To further extend the platforms where the models can be deployed, we have developed the Java package "frbsJpmml" (`http://dicits.ugr.es/software/frbsJpmml/`) which can be used for importing an FRBS model in the frbsPMML format and applying it over new datasets.

4. Designing and implementing the package "SparkFernTreeR:" It is an R package used to implement decision tree, random forest, and random ferns for dealing with Big Data analysis. It can be used on standalone machine and cluster/parallel computing using the Apache Spark Framework. Using the random forest method we are allowed to deal with classification, regression, density estimation/clustering, and manifold learning, whereas random ferns can be used for classification.

B. The Associated Publications

The following is a list of publications related to the thesis:

- Publication on international journals:

    1. 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, Vol. 65(6), p. 1–30, 2015, `http://www.jstatsoft.org/v65/i06/`.

        - Status: **Published**
        - Impact Factor (JCR 2013): 3.801
        - Subject category:
            * COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS, Ranking: 9 of 102 (Q1)
            * STATISTICS & PROBABILITY, Ranking: 2 of 119 (Q1)

2. L.S. Riza, A. Janusz, C. Bergmeir, C. Cornelis, F. Herrera, D. Ślęzak, and J.M. Benítez. Implementing Algorithms of Rough Set Theory and Fuzzy Rough Set Theory in the R Package. Information Sciences, 287, p. 68-98, 2014.

   - Status: **Published**
   - Impact Factor (JCR 2013): 3.893
   - Subject category:
     * COMPUTER SCIENCE, INFORMATION SYSTEMS, Ranking: 8 of 135 (Q1)

3. L.S. Riza, C. Bergmeir, F. Herrera, and J.M. Benítez. A Universal Representation Framework for Fuzzy Rule-Based Systems Based on PMML. Information Sciences, 2015 (submitted)

   - Status: **Submitted**
   - Impact Factor (JCR 2013): 3.893
   - Subject category:
     * COMPUTER SCIENCE, INFORMATION SYSTEMS, Ranking: 8 of 135 (Q1)

4. L.S. Riza, C. Bergmeir, B. Krawczyk, F. Herrera, and J.M. Benítez. "SparkFernTreeR": An R Package Based on Random Forest and Random Ferns for Big Data Analysis under Apache Spark. R Journal, 2015 (To be submitted)

   - Status: **To be submitted**
   - Impact Factor (JCR 2013): 0.895
   - Subject category:
     * COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS, Ranking: 77 of 102 (Q4)
     * STATISTICS & PROBABILITY, Ranking: 59 of 119 (Q2)

- Dissemination on conferences and workshops:

  - The R user Conference 2013.
    * Title: Constructing Fuzzy Rule-based System with the R Package "frbs."

    - Place and date: University of Castilla-La Mancha, Albacete, Spain, July 10 - 12, 2013.
  - 2014 IEEE World Congress on Computational Intelligence.
    - Title: Learning from Data Using the R Package "frbs."
    - Place and date: Beijing, July 6 - 11, 2014.
  - The R User Conference 2014.
    - Title: R as a PaaS Cloud Computing Service for Computational Intelligence Tasks.
    - Place and date: UCLA, Los Angeles, California, June 30 - July 3, 2014.
  - 2014 Joint Rough Set Symposium.
    - Title: Workshop on Computational Intelligence in R: "RSNNS," "Rmalschains," "frbs," and "RoughSets."
    - Place and date: Granada and Madrid, Spain, July 9 - 13, 2014.
  - 2014 The Webinar of the Orange County R User Group.
    - Title: "RoughSets:" A Classification Tool Based on Rough Sets and Fuzzy Rough Sets in R.
    - Place and date: Webinar, June 24, 2014

C. Future Work

There are many opportunities to extend the work carried out in this thesis. Concretely, we plan for the future to continue work on the following:

1. For the package "RoughSets," we will extend it to deal with different specific data mining problems, such as imbalanced classification.

2. The package "frbs" will be improved by adding new methods to deal with regression and classification tasks.

3. Additionally, using frbsPMML, we will deploy FRBSs into Cloud Computing's schema so that users are able to use it as a Software as a Service (SaaS) and a Platform as a Service (PaaS).

4. We will develop a standard representation for RST and FRST based on PMML.

5. We plan to extend "frbs" and "RoughSets" so that the packages can be used for Big Data processing under Apache Spark.

6. We intend to enrich data preprocessing, such as feature extraction and feature selection, for Big Data within the R ecosystem.

Appendix

Appendix A. The "frbs" Package in CRAN

In this appendix we include the information for the "frbs" package in the CRAN repository as shown in Figure 7. In order to provide some hints on the development effort, we present Table 4 illustrating the following data: number of functions and number of source code lines.

frbs: Fuzzy Rule-Based Systems for Classification and Regression Tasks

An implementation of various learning algorithms based on fuzzy rule-based systems (FRBSs) for dealing with classification and regression tasks. Moreover, it allows to construct an FRBS model defined by human experts. FRBSs are based on the concept of fuzzy sets, proposed by Zadeh in 1965, which aims at representing the reasoning of human experts in a set of IF-THEN rules, to handle real-life problems in, e.g., control, prediction and inference, data mining, bioinformatics data processing, and robotics. FRBSs are also known as fuzzy inference systems and fuzzy models. During the modeling of an FRBS, there are two important steps that need to be conducted: structure identification and parameter estimation. Nowadays, there exists a wide variety of algorithms to generate fuzzy IF-THEN rules automatically from numerical data, covering both steps. Approaches that have been used in the past are, e.g., heuristic procedures, neuro-fuzzy techniques, clustering methods, genetic algorithms, squares methods, etc. Furthermore, in this version we provide a universal framework named 'frbsPMML', which is adopted from the Predictive Model Markup Language (PMML), for representing FRBS models. PMML is an XML-based language to provide a standard for describing models produced by data mining and machine learning algorithms. Therefore, we are allowed to export and import an FRBS model to/from 'frbsPMML'. Finally, this package aims to implement the most widely used standard procedures, thus offering a standard package for FRBS modeling to the R community.

Version: 3.1-0
Suggests: class, e1071, XML
Published: 2015-05-22
Author: Lala Septem Riza, Christoph Bergmeir, Francisco Herrera, and Jose Manuel Benitez
Maintainer: Christoph Bergmeir <c.bergmeir at decsai.ugr.es>
License: GPL-2 | GPL-3 | file LICENSE [expanded from: GPL (≥ 2) | file LICENSE]
URL: http://sci2s.ugr.es/dicits/software/FRBS
NeedsCompilation: no
Citation: frbs citation info
In views: MachineLearning
CRAN checks: frbs results

Downloads:

Reference manual: frbs.pdf
Package source: frbs_3.1-0.tar.gz
Windows binaries: r-devel: frbs_3.1-0.zip, r-release: frbs_3.1-0.zip, r-oldrel: frbs_3.1-0.zip
OS X Snow Leopard binaries: r-oldrel: frbs_3.0-0.tgz
OS X Mavericks binaries: r-release: frbs_3.1-0.tgz
Old sources: frbs archive

Reverse dependencies:

Reverse depends: fuzzyMM

Figure 7: The display of the "frbs" package in CRAN at http://cran.r-project.org/package=frbs.

No.	Filenames	Line of Code
1	FNN.FunctionCollection.R	299
2	GFS.Methods.R	996
3	FRBS.MainFunction.R	2415
4	FGradDescent.Methods.R	299
5	FCluster.Methods.R	267
6	GFS.Predict.R	112
7	FNN.Methods.R	279
8	frbs-package.R	454
9	FSpacePartition.Method.R	584
10	FRBS.pmml.R	623
11	FCluster.Predict.R	119
12	FCluster.FunctionCollection.R	201

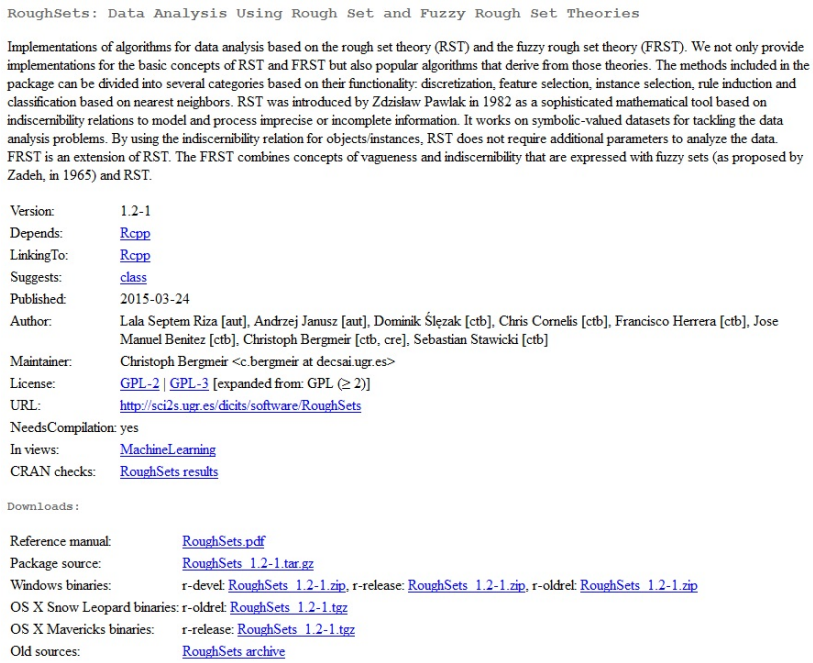
13	pmml.R	867
14	GFS.FunctionCollection.R	1885
15	pmml.frbs.R	573
16	docData.R	57
17	FGradDescent.FunctionCollection.R	209
18	FSpacePartition.Predict.R	214
19	FSpacePartition.FunctionCollection.R	1259
20	GFS.LT.RS.MG1000.R	57
21	HyFIS.MG1000.R	54
22	ANFIS.GasFur.R	54
23	FRBCS.W.Iris.R	31
24	FIR.DM.GasFur.R	55
25	FRBCS.CHI.Iris.R	31
26	ANFIS.MG1000.R	55
27	FIR.DM.MG1000.R	54
28	WM.GasFur.R	54
29	Thrift.MG1000.R	56
30	GFS.MEMETIC.GasFur.R	55
31	HyFIS.GasFur.R	55
32	DENFIS.MG1000.R	55
33	FRBS.Manual.R	88
34	FH.GBML.Iris.R	34
35	GFS.MEMETIC.MG1000.R	56
36	GFS.FR.MOGUL.GasFur.R	55
37	GFS.FR.MOGUL.MG1000.R	56
38	DENFIS.GasFur.R	54
39	ANFIS.GasFur.PMML.R	72
40	WM.MG1000.R	54
41	FRBS.FRBCS.Manual.R	65
42	SLAVE.Iris.R	33
43	GFS.LT.RS.GasFur.R	57
44	Thrift.GasFur.R	55
45	FS.HGD.MG1000.R	55
46	SBC.GasFur.R	53
47	FRBS.Mamdani.Manual.R	95
48	GFS.GCCL.Iris.R	34
49	SBC.MG1000.R	53
50	FS.HGD.GasFur.R	55
51	FRBS.TSK.Manual.R	74
52	WM.GasFur.PMML.R	75

53	GFS.GCCL.Iris.PMML.R	55
Total:		13606

Table 4: Notes on the engineering software process of “frbs.”

Appendix B. The "RoughSets" Package in CRAN

In this appendix we include the information for the "RoughSets" package in the CRAN repository as shown in Figure 8. In order to provide some hints on the development effort, we present Table 5 illustrating the following data: number of functions and number of source code lines.



RoughSets: Data Analysis Using Rough Set and Fuzzy Rough Set Theories

Implementations of algorithms for data analysis based on the rough set theory (RST) and the fuzzy rough set theory (FRST). We not only provide implementations for the basic concepts of RST and FRST but also popular algorithms that derive from those theories. The methods included in the package can be divided into several categories based on their functionality: discretization, feature selection, instance selection, rule induction and classification based on nearest neighbors. RST was introduced by Zdzisław Pawlak in 1982 as a sophisticated mathematical tool based on indiscernibility relations to model and process imprecise or incomplete information. It works on symbolic-valued datasets for tackling the data analysis problems. By using the indiscernibility relation for objects/instances, RST does not require additional parameters to analyze the data. FRST is an extension of RST. The FRST combines concepts of vagueness and indiscernibility that are expressed with fuzzy sets (as proposed by Zadeh, in 1965) and RST.

Version:	1.2-1
Depends:	Rcpp
LinkingTo:	Rcpp
Suggests:	class
Published:	2015-03-24
Author:	Lala Septem Riza [aut], Andrzej Janusz [aut], Dominik Ślęzak [ctb], Chris Cornelis [ctb], Francisco Herrera [ctb], Jose Manuel Benitez [ctb], Christoph Bergmeir [ctb, cre], Sebastian Stawicki [ctb]
Maintainer:	Christoph Bergmeir <c.bergmeir at decsai.ugr.es>
License:	GPL-2 | GPL-3 [expanded from: GPL (≥ 2)]
URL:	http://sci2s.ugr.es/dicits/software/RoughSets
NeedsCompilation:	yes
In views:	MachineLearning
CRAN checks:	RoughSets results

Downloads:

Reference manual:	RoughSets.pdf
Package source:	RoughSets_1.2-1.tar.gz
Windows binaries:	r-devel: RoughSets_1.2-1.zip, r-release: RoughSets_1.2-1.zip, r-oldrel: RoughSets_1.2-1.zip
OS X Snow Leopard binaries:	r-oldrel: RoughSets_1.2-1.tgz
OS X Mavericks binaries:	r-release: RoughSets_1.2-1.tgz
Old sources:	RoughSets archive

Figure 8: The display of the "RoughSets" package in CRAN at http://cran.r-project.org/package=RoughSets.

No.	Filenames	Lines of Code
1	BasicFuzzyRoughSets.R	1672
2	InstanceSelection.R	329
3	BasicRoughSets.OtherFuncCollections.R	1298
4	Discretization.R	674
5	InstanceSelection.OtherFuncCollections.R	63
6	IOFunctions.R	830
7	FeatureSelection.R	1582
8	MissingValue.R	402
9	FuzzyRoughSets-introduction.R	137
10	RoughSets-package.R	543
11	RuleInduction.OtherFuncCollections.R	475
12	FeatureSelection.OtherFuncCollections.R	936
13	RcppExports.R	11
14	Discretization.OtherFuncCollections.R	291
15	docData.R	144

16	RuleInduction.R	1051
17	NearestNeigbour.OtherFuncCollections.R	240
18	BasicRoughSets.R	402
19	NearestNeigbour.R	337
20	RoughSets-introduction.R	116
21	FS.permutation.heuristic.reduct.RST.R	14
22	FRNN.O.iris.R	33
23	FS.QuickReduct.FRST.Ex3.R	51
24	RI.classification.FRST.R	31
25	FS.QuickReduct.FRST.Ex5.R	57
26	FRNN.iris.R	38
27	DiscernibilityMatrix.FRST.R	28
28	FS.QuickReduct.FRST.Ex4.R	54
29	IS.FRPS.FRST.R	15
30	GettingStarted.B.R	136
31	FS.greedy.heuristic.reduct.RST.R	14
32	POSNN.iris.R	32
33	SimulationDataAnalysisWine.R	148
34	FS.QuickReduct.FRST.Ex2.R	51
35	RI.indiscernibilityBasedRules.RST.R	19
36	FS.greedy.heuristic.superreduct.RST.R	14
37	BasicConcept.FRST.R	47
38	D.discretize.quantiles.RST.R	14
39	IS.FRIS.FRST.R	19
40	BasicConcept.RST.R	25
41	RI.regression.FRST.R	19
42	FS.QuickReduct.FRST.Ex1.R	56
43	D.local.discernibility.matrix.RST.R	13
44	D.discretize.equal.intervals.RST.R	14
45	FS.nearOpt.fvprs.FRST.R	9
46	GettingStarted.A.R	103
47	MV.simpleData.R	30
48	D.global.discernibility.heuristic.RST.R	14
49	FS.QuickReduct.RST.R	10
50	DiscernibilityMatrix.RST.R	11
51	D.max.discernibility.matrix.RST.R	11
Total:		12663

Table 5: Notes on the engineering software process of "RoughSets."

Appendix C. The "SparkFernTreeR" Package

In order to provide some hints on the development effort, we present Table 6 illustrating the following data: number of functions and number of source code lines.

No.	Filenames	Lines of Code
1	RandomForest-introduction.R	231
2	SparkFernTreeR-package.R	421
3	SparkR-introduction.R	62
4	docData.R	57
5	modelFunctions.R	794
6	predictorFunctions.R	1342
7	supportingFunctionRF.R	394
8	utilityFunctions.R	670
9	FourHill_RF_ex1.R	75
10	GasFur_RF_ex1.R	54
11	Iris_RF_ex1.R	35
12	Iris_SRF_ex1.R	59
13	Iris_SRF_ex2.R	49
14	Iris_SRF_ex3.R	59
15	Iris_SRF_ex4.R	60
16	Iris_SRF_ex5.R	80
17	Iris_SRF_ex6.R	52
18	Iris_SRFerns_ex1.R	59
19	Iris_SRFerns_ex2.R	40
20	Iris_SRFerns_ex3.R	68
21	MackeyGlass_RF_ex1.R	54
22	MackeyGlass_SRF_ex1.R	61

23	Pima_SRF_ex1.R	51
24	Wine_RF_exCluster1.R	15
25	Wine_RF_exManifold1.R	11
26	Wine_SRF_ex1.R	53
27	Wine_SRF_exCluster1.R	45
28	Wine_SRF_exManifold1.R	46
Total:		4997

Table 6: Notes on the engineering software process of "SparkFernTreeR."

Bibliography

[1] B. Abdulhai and L. Kattan. Reinforcement learning: Introduction to theory and potential for transport applications. *Canadian Journal of Civil Engineering*, 30(6):981–991, 2003.

[2] Y.S. Abu-Mustafa, M. Magdon-Ismail, and H. Lin. *Learning from Data: A Short Course*. AMLbook.com, 2012.

[3] G. Acampora and V. Loia. Fuzzy control interoperability and scalability for adaptive domotic framework. *Industrial Informatics, IEEE Transactions on*, 1(2):97–111, 2005.

[4] G. Acampora, V. Loia, C-S. Lee, and M-H. Wang. *On the power of fuzzy markup language*. Springer, 2013.

[5] D. Adler, C. Gläser, O. Nenadic, J. Oehlschlägel, and W. Zucchini. *ff: Memory-efficient Storage of Large Data on Disk and Fast Access functions*, 2014. R package version 2.2-13, `http://cran.r-project.org/web/packages/ff/index.html`.

[6] J. Adler. *R in a Nutshell*. O'Reilly Media, 2012.

[7] K. Ahmed, Abdullah-Al-Emran, Abdullah-Al-Emran, T. Jesmin, R.F. Mukti, M.Z. Rahman, and F. Ahmed. Early detection of lung cancer risk using data mining. *Asian Pacific Journal of Cancer Prevention*, 14(1):595–598, 2013.

[8] M. Akın. A novel approach to model selection in tourism demand modeling. *Tourism Management*, 48:64–72, 2015.

[9] R. Alcala, J. Alcala-Fdez, and F. Herrera. A proposal for the genetic lateral tuning of linguistic fuzzy systems and its interaction with rule selection. *IEEE Transactions on Fuzzy Systems*, 15(4):616–635, 2007.

[10] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera. KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2–3):255–287, 2011.

[11] J. Alcalá-Fdez, L. Sánchez, S. García, M.J. del Jesus, S. Ventura, J.M. Garrell, J. Otero, C. Romero, J. Bacardit, V.M. Rivas, J.C. Fernández, and F. Herrera. KEEL: A software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2009.

[12] J.M. Alonso and L. Magdalena. Generating understandable and accurate fuzzy rule-based systems in a java environment. In *Lecture Notes in Artificial Intelligence - 9th International Workshop on Fuzzy Logic and Applications, Springer-Verlag, LNAI6857*, pages 212–219, 2011.

[13] J.M. Alonso, L. Magdalena, and S. Guillaume. KBCT: A knowledge extraction and representation tool for fuzzy logic based systems. In *Fuzzy Systems, 2004. Proceedings. 2004 IEEE International Conference on*, volume 2, pages 989–994. IEEE, 2004.

[14] X. Amatriain. Big & personal: Data and models behind Netflix recommendations. In *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 1–6. ACM, 2013.

[15] AMPLab UC Berkely. *SparkR: R Frontend for Spark*, 2015. R package, `http://amplab-extras.github.io/SparkR-pkg/`.

[16] B. Amrouche and X.L. Pivert. Artificial neural network based daily local forecasting for global solar radiation. *Applied Energy*, 130:333–341, 2014.

[17] R.C. Atkinson and R.M. Shiffrin. Human memory: A proposed system and its control processes. *Psychology of learning and motivation*, 2:89–195, 1968.

[18] H. Aydt, S.J. Turner, W. Cai, M.Y.H. Low, Y.S. Ong, and R. Ayani. Toward an evolutionary computing modeling language. *Evolutionary Computation, IEEE Transactions on*, 15(2):230–247, 2011.

[19] J.C. Backhus, H. Nonaka, T. Yoshikawa, and M. Sugimoto. Application of reinforcement learning to the card game wizard. In *2013 IEEE 2nd Global Conference on Consumer Electronics, GCCE 2013*, pages 329–333, 2013.

[20] P. Barbera. *streamR: Access to Twitter streaming API via R*, 2014. R package version 0.2.1, `http://cran.r-project.org/web/packages/streamR/index.html`.

[21] I. Baturone, F.J. Moreno-Velo, S. Sánchez-Solano, A. Barriga, P. Brox, A. Gersnoviez, and M. Brox. Using Xfuzzy environment for the whole design of fuzzy systems. In *Proc. IEEE International Conference on Fuzzy Systems*, pages 1–6, 2007.

[22] J. Bazan. A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision tables. In A. Skowron and L. Polkowski, editors, *Rough Sets in Knowledge Discovery 1*, volume 1, pages 321–365. Physica Verlag, Heidelberg, 1998.

[23] J.G. Bazan, H.S. Nguyen, S.H. Nguyen, P. Synak, and J. Wróblewski. Rough set algorithms in classification problem. In L. Polkowski, S. Tsumoto, and T.Y. Lin, editors, *Rough Set Methods and Applications*, pages 49–88, Heidelberg, New York, 2000. Physica-Verlag.

[24] J.G. Bazan and M. Szczuka. RSES and RSESlib–a collection of tools for rough set computations. In W. Ziarko and Y. Yao, editors, *Proceedings of the 2nd International Conference on Rough Sets and Current Trends in Computing (RSCTC'2000)*, volume 2005, pages 106–113, 2000.

[25] J.G. Bazan and M. Szczuka. The rough set exploration system. In J.F. Peters and A. Skowron, editors, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 3400 LNCS*, pages 37–56, 2005.

[26] D. Beaver, S. Kumar, H.C. Li, J. Sobel, P. Vajgel, et al. Finding a needle in haystack: Facebook's photo storage. In *OSDI*, volume 10, pages 1–8, 2010.

[27] C. Bergmeir and J.M. Benítez. Neural networks in R using the stuttgart neural network simulator: RSNNS. *Journal of Statistical Software*, 46(7):1–26, 2012.

[28] M.R. Berthold, N. Cebron, F. Dill, T.R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wisedel. *KNIME: The Konstanz Information Miner*. Springer, 2007.

[29] M.A. Beyer and D. Laney. The importance of 'big data': A definition, 2012. `https://www.gartner.com`.

[30] R.B. Bhatt and M. Gopal. On fuzzy-rough sets approach to feature selection. *Pattern Recognition Letters*, 26:965–975, 2005.

[31] B. Bischl and M. Lang. *parallelMap: Unified Interface to Some Popular Parallelization Back-ends for Interactive Usage and Package Development*, 2015. R package version 1.2, `http://cran.r-project.org/web/packages/parallelMap/index.html`.

[32] G. Box and G.M. Jenkins. *Time Series Analysis: Forecasting and Control*. CA: Holden Day, 1970.

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

[34] L. Breiman, A. Cutler, A. Liaw, and M. Wiener. *randomForest: Breiman and Cutler's Random Forest for Classification and Regression*, 2012. R package version 4.6-7, `http://www.stat-www.berkeley.edu/users/breiman/RandomForests`.

[35] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. 1984.

[36] J.J. Buckley and Y. Hayashi. Fuzzy neural networks: A survey. *Fuzzy Sets and Systems*, 66:1–13, 1994.

[37] J. Buckner, M. Seligman, and J. Wilson. *gputools: A few GPU enabled functions*, 2013. R package version 0.28, `http://cran.r-project.org/web/packages/gputools/index.html`.

[38] A. Bujard. *fugeR: Fuzzy genetic, a machine learning algorithm to construct prediction model based on fuzzy logic*, 2012. R package version 0.1.2, `http://CRAN.R-project.org/package=fugeR`.

[39] R. Buyya. Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility. In *Proc. of CCGRID '09, Proc. of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009.

[40] R. Buyya, C.S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities. In *Proc. of the Tenth IEEE International Conference on High Performance Computing and Communications*, 2008.

[41] Y. Cai, L. Song, T. Wang, and Q. Chang. Financial time series forecasting using directed-weighted chunking svms. *Mathematical Problems in Engineering*, 170424:1–7, 2014.

[42] J. Casillas, O. Cordon, F. Herrera, and L. Magdalena (Eds). *Interpretability Issues in Fuzzy Modeling*. Springer-Verlag Berlin Heidelberg, 2003.

[43] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[44] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. MIT Press, Cambridge, 2006.

[45] D. Chen, L. Zhang, S. Zhao, Q. Hu, and P. Zhu. A novel algorithm for finding reducts with fuzzy rough sets. *IEEE Transactions on Fuzzy Systems*, 20:385–389, 2012.

[46] D.G. Chen, Q.H. Hu, and Y.P. Yang. Parameterized attribute reduction with Gaussian kernel based fuzzy rough sets. *Information Sciences*, 181:5169–5179, 2011.

[47] W.C. Chen, G. Ostrouchov, D. Schmidt, P. Patel, and H. Yu. pbdMPI: Programming with big data – interface to MPI, 2012. R Package, `http://cran.r-project.org/package=pbdMPI`.

[48] W.C. Chen, G. Ostrouchov, D. Schmidt, P. Patel, and H. Yu. *A Quick Guide for the pbdMPI Package (Ver. 0.2-2)*, 2014. R Vignette, `http://cran.r-project.org/package=pbdMPI`.

[49] Z. Chi, H. Yan, and T. Pham. *Fuzzy Algorithms with Applications to Image Processing and Pattern Recognition*. World Scientific, 1996.

[50] S. Chiu. Method and software for extracting fuzzy classification rules by subtractive clustering. *Fuzzy Information Processing Society, NAFIPS*, pages 461–465, 1996.

[51] D. Conway. The data science venn diagram. http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram.

[52] O. Cordon, F. Herrera, F. Hoffmann, and L. Magdalena. *Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*. Singapore: World Scientific Publishing, 2001.

[53] C. Cornelis, M. De Cock, and A. Radzikowska. Vaguely quantified rough sets. In *Proceedings of 11th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC2007), Lecture Notes in Artificial Intelligence*, volume 4482, pages 87–94, 2007.

[54] C. Cornelis and R. Jensen. A noise-tolerant approach to fuzzy-rough feature selection. In *Proceedings of the 2008 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2008)*, pages 1598–1605, 2008.

[55] C. Cornelis, R. Jensen, G. Hurtado, and D. Ślęzak. Attribute selection with fuzzy decision reducts. *Information Sciences*, 180(2):209–224, 2010.

[56] C. Cornelis, N. Verbiest, and R. Jensen. Ordered weighted average based fuzzy rough sets. In *Proceedings of the 5th International Conference on Rough Sets and Knowledge Technology (RSKT 2010)*, pages 78–85, 2010.

[57] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[58] A. Criminisi and J. Shotton. *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media, 2013.

[59] P. Danenas and G. Garsva. Selection of support vector machines based classifiers for credit risk domain. *Expert Systems with Applications*, 42(6):3194–3204, 2015.

[60] S. Das, Y. Sismanis, K.S. Beyer, R. Gemulla, P.J. Haas, and J. McPherson. Ricardo: Integrating R and hadoop. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 987–998. ACM, 2010.

[61] S. Datoo. Big data: 4 predictions for 2014, 2014. http://www.theguardian.com/technology/datablog/2014/jan/14/big-data-4-predictions-for-2014.

[62] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[63] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[64] S. del Río, V. López, J.M. Benítez, and F. Herrera. On the use of MapReduce for imbalanced big data using random forest. *Information Sciences*, 285:112–137, 2014.

[65] Department of Finance and Deregulation. The Australian public service big data strategy, 2013. `http://www.finance.gov.au/sites/default/files/Big-Data-Strategy_0.pdf`.

[66] V. Dhar. Data science and prediction. *Communications of the ACM*, 56(12):64–73, 2013.

[67] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. pages 194–20, San Francisco, CA, 1995. Morgan Kaufmann.

[68] D. Dubois and H. Prade. Rough fuzzy sets and fuzzy rough sets. *International Journal of General Systems*, 17:91–209, 1990.

[69] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[70] K. Ebcioglu, V. Saraswat, and V. Sarkar. X10: Programming for hierarchical parallelism and non-uniform data access. In *Proceedings of the International Workshop on Language Runtimes, OOPSLA*, volume 30. Citeseer, 2004.

[71] D. Eddelbuettel, M. Stokely, and J. Ooms. RProtoBuf: Efficient cross-language data serialization in R. *arXiv preprint arXiv:1401.7372*, 2014. `http://arxiv.org/abs/1401.7372`.

[72] A.P. Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[73] Executive Office of the President of USA. Fact sheet: Big data across the federal government. Technical report, 2012. `https://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_fact_sheet_final.pdf`.

[74] U.M. Fayyad and K.B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87–102, 1992.

[75] U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous attributes as preprocessing for classification learning. In *Proceeding Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027. Morgan Kaufmann, 1995.

[76] J.A. Fernandes, X. Irigoien, J.A. Lozano, I. Inza, N. Goikoetxea, and A. Pérez. Evaluating machine-learning techniques for recruitment forecasting of seven North East Atlantic fish species. *Ecological Informatics*, 25:35–42, 2015.

[77] A. Fernández, S. del Río, V. López, A. Bawakid, M.J. del Jesus, J.M. Benítez, and F. Herrera. Big data with cloud computing: an insight on the computing environment, MapReduce, and programming frameworks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(5):380–409, 2014.

[78] A.R. Ferreira da Silva. cudaBayesreg: Parallel implementation of a bayesian multilevel model for fMRI data analysis. *Journal of Statistical Software*, 44(4):1–24, 2011. `http://www.jstatsoft.org/v44/i04/`.

[79] E. Fix and J. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. Technical report, 1951.

[80] M. Forina, E. Leardi, C. Armanino, and S. Lanteri. PARVUS: An extendable package of programs for data exploration, classification and correlation. *Journal of Chemonetrics*, 4(2):191–193, 1988.

[81] R. Francois, D. Eddelbuettel, M. Stokely, and J. Ooms. *RProtoBuf: R Interface to the Protocol Buffers API*, 2014. R package version 0.4.2, `http://CRAN.R-project.org/package=RProtoBuf`.

[82] Y. Freund and R.E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer, 1995.

[83] J.H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[84] A. Fu, S. Aiello, A. Rao, A. Wang, T. Kraljevic, and P. Maj. *h2o: H2OR Interface*, 2015. R package version 2.8.4.4, `http://cran.r-project.org/web/packages/h2o/index.html`.

[85] L.A. Gabralla, H. Mahersia, and A. Abraham. Ensemble neurocomputing based oil price prediction. *Advances in Intelligent Systems and Computing*, 334:293–302, 2015.

[86] E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V.l Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, et al. Open mpi: Goals, concept, and design of a next generation mpi implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 97–104. Springer, 2004.

[87] J. Gantz and D. Reinsel. Extracting value from chaos. Technical report, 2011. `http://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf`.

[88] Q. Gao, Y. Huang, X. Gao, W. Shen, and H. Zhang. A novel semi-supervised learning for face recognition. *Neurocomputing*, 152:69–76, 2015.

[89] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM, 2003.

[90] J. Ginsberg, M.H. Mohebbi, R.S. Patel, L. Brammer, M.S. Smolinski, and L. Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457(7232):1012–1014, 2009.

[91] A. Gonzalez and R. Peréz. Selection of relevant features in a fuzzy genetic learning algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 31(3):417–425, 2001.

[92] A. González, R. Pérez, and J.L. Verdegay. Learning the structure of a fuzzy rule: A genetic approach. In *Proc. First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, pages 814–819, 1993.

[93] Google Cloud Platform. What is BigQuery? `https://cloud.google.com/bigquery/what-is-bigquery`.

[94] J. Grzymala-Busse and W. Grzymala-Busse. Handling missing attribute values. In O. Maimon and L. Rokach, editors, *Data Mining and*

Knowledge Discovery Handbook, pages 33–51. New York: Springer, 2010.

[95] J.W. Grzymała-Busse. LERS - A system for learning from examples based on rough sets. In R. Słowiński, editor, *Intelligent Decision Support*, pages 3–18, 1992.

[96] J.W. Grzymała-Busse. A new version of the rule induction system LERS. *Fundamenta Informaticae*, 31(1):27–39, 1997.

[97] J.W. Grzymała-Busse. MLEM2: A new algorithm for rule induction from imperfect data. In *Proceedings of the 9th International Conference on Information Processing and Managament of Uncertainty in Knowledge-Based Systems (IPMU)*, pages 243–250, 2002.

[98] J.W. Grzymała-Busse. LERS - a data mining system. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 1347–1351. Springer US, 2005.

[99] J.W. Grzymała-Busse and M. Hu. A comparison of several approaches to missing attribute values in data mining. In *Proceesings of the Second International Conference on Rough Sets and Current Trends in Computing RSCTC'2000, LNAI 2005*, pages 378–385. Springer-Verlag, Berlin, 2000.

[100] J.W. Grzymała-Busse and W. Rzasa. A local version of the MLEM2 algorithm for rule induction. *Fundamenta Informaticae*, 100(1–4):99–116, 2010.

[101] A. Guazzelli. Representing predictive solutions in PMML: From raw data to predictions. Technical report, 2010. `http://www.ibm.com/developerworks/library/ba-ind-PMML2`.

[102] A. Guazzelli, K. Stathatos, and M. Zeller. Efficient deployment of predictive analytics through open standards and cloud computing. *ACM SIGKDD Explorations Newsletter*, 11(1):32–38, 2009.

[103] A. Guazzelli, M. Zeller, W. Lin, and G. Williams. PMML: An open standard for sharing models. *The R Journal*.

[104] S. Guha, R. Hafen, J. Rounds, J. Xia, J. Li, B. Xi, and W.S. Cleveland. Large complex data: Divide and recombine (d&r) with rhipe. *Stat*, 1(1):53–67, 2012.

[105] S. Guillaume and B. Charnomordic. Learning interpretable fuzzy inference systems with FisPro. *Information Sciences*, 181(20):4409–4427, 2011.

[106] P.D. Gutiérrez, M. Lastra, F. Herrera, and J.M. Benítez. A high performance fingerprint matching system for large databases based on GPU. *IEEE Transactions on Information Forensics and Security*, 9(1):62–71, 2014.

[107] M. Hahsler, M. Bolanos, and J. Forrest. *stream: Infrastructure for Data Stream Mining*, 2015. R package version 1.1-1,`http://cran.r-project.org/web/packages/stream/index.html`.

[108] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[109] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA data mining software: An update. In *ACM SIGKDD Explorations Newsletter*, volume 11, pages 10–18, 2009.

[110] S.B. Handurukande, M. Wang, and M. Nassar. RPig: A scalable framework for machine learning and advanced statistical functionalities. In *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 293–300. IEEE Computer Society, 2012.

[111] D. Harrison and D.L. Rubinfeld. Hedonic prices and the demand for clean air. *Journal of Environ. Economics and Management*, 5:81–102, 1978.

[112] I.A.T. Hashem, I. Yaqoob, N.B. Anuar, S. Mokhtar, A. Gani, and S.U. Khan. The rise of big data on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, 2015.

[113] C. Hayashi. What is data science? fundamental concepts and a heuristic example. In *Data Science, Classification, and Related Methods*, pages 40–51. Springer, 1998.

[114] J.M. Hellerstein, C. Ré, F. Schoppmann, D.Z. Wang, E. Fratkin, A. Gorajek, K.S. Ng, C. Welton, X. Feng, K. Li, et al. The MADlib analytics library: or MAD skills, the SQL. *Proceedings of the VLDB Endowment*, 5(12):1700–1711, 2012.

[115] F. Herrera. Genetic fuzzy systems: Taxonomy, current research trends and prospects. *Evolutionary Intelligence*, 1:27–46, 2008.

[116] F. Herrera, M. Lozano, and J. Verdegay. A learning process for fuzzy control rules using genetic algorithms. *Fuzzy Sets and System*, 100:143–158, 1998.

[117] F. Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, 2002.

[118] Q. Hu, S. An, and D. Yu. Soft fuzzy rough sets for robust feature evaluation and selection. *Information Sciences*, 180:4384–4407, 2010.

[119] Q. Hu, D. Yu, W. Pedrycz, and D. Chen. Kernelized fuzzy rough sets and their applications. *IEEE Transactions Knowledges Data Engineering*, 23:1649–1471, 2011.

[120] Q. Hu, L. Zhang, S. An, D. Zhang, and D. Yu. On robust fuzzy rough set models. *IEEE Transactions on Fuzzy Systems*, 20:636–651, 2012.

[121] H. Huang, S. Tata, and R.J. Prill. Bluesnp: R package for highly scalable genome-wide association studies using hadoop clusters. *Bioinformatics*, 29(1):135–136, 2013.

[122] P. Huijse, P. Estevez, P. Protopapas, J. Principe, and P. Zegers. Computational intelligence challenges and applications on large-scale astronomical time series databases. *Computational Intelligence Magazine, IEEE*, 9(3):27–39, 2014.

[123] IBM. *An Introduction to Big R*, 2015. R package version 1.0, `http://www-01.ibm.com/support/knowledgecenter/SSPT3X_3.0.0/com.ibm.swg.im.infosphere.biginsights.bigr.doc/doc/intro.html`.

[124] R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[125] Institute of Electrical and Electronics Engineers (IEEE). IEEE standard computer dictionary: a compilation of IEEE standard computer glossaries: 610. *Institute of Electrical and Electronics Engineers (IEEE)*, 1991.

[126] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In

ACM SIGOPS Operating Systems Review, volume 41, pages 59–72. ACM, 2007.

[127] H. Ishibuchi and T. Nakashima. Effect of rule weights in fuzzy rule-based classification systems. *IEEE Transactions on Fuzzy Systems*, 1:59–64, 2001.

[128] H. Ishibuchi, T. Nakashima, and T. Murata. Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(5):601–618, 1999.

[129] H. Ishibuchi, K. Nozaki, and H. Tanaka. Distributed representation of fuzzy rules and its application to pattern classification. *Fuzzy Sets and Systems*, 52:21–32, 1992.

[130] H. Ishibuchi, K. Nozaki, and H. Tanaka. Empirical study on learning in fuzzy systems by rice taste analysis. *Fuzzy Sets and Systems*, 64(2):129–144, 1994.

[131] H. Ishibuchi, T. Yamamoto, and T. Nakashima. Hybridization of fuzzy GBML approaches for pattern classification problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(2):359–365, 2005.

[132] J.S.R. Jang. ANFIS: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–685, 1993.

[133] A. Janusz and D. Ślęzak. Rough set methods for attribute clustering and selection. *Applied Artificial Intelligence*, 28(3):220–242, march 2014.

[134] A. Janusz and S. Stawicki. Applications of approximate reducts to the feature selection problem. In *Proceedings of International Conference on Rough Sets and Knowledge Technology (RSKT)*, volume 6954, pages 45–50, 2011.

[135] T. Jena, A. Guazzelli, W. Lin, and M. Zeller. The R pmmlTransformations package. In *Proc. of the 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2013.

[136] R. Jensen. Fuzzy-rough data mining with WEKA. Technical report, 2010. `http://users.aber.ac.uk/rkj/Weka.pdf`.

[137] R. Jensen and C. Cornelis. A new approach to fuzzy-rough nearest neighbour classification. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 5306 LNAI*, pages 310–319, 2008.

[138] R. Jensen and C. Cornelis. Fuzzy-rough instance selection. In *Proceedings of the 19th International Conference on Fuzzy Systems (FUZZ-IEEE 2010)*, pages 1776–1782, 2010.

[139] R. Jensen and C. Cornelis. Fuzzy-rough nearest neighbour classificition and prediction. *Theoretical Computer Science*, 412:5871–5884, 2011.

[140] R. Jensen, C. Cornelis, and Q. Shen. Hybrid fuzzy-rough rule induction and feature selection. In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1151–1156, 2009.

[141] R. Jensen and Q. Shen. Fuzzy-rough sets for descriptive dimensionality reduction. In *Proceedings of IEEE International Conference on Fuzzy System, FUZZ-IEEE*, pages 29–34, 2002.

[142] R. Jensen and Q. Shen. Fuzzy-rough data reduction with ant colony optimization. *Fuzzy Sets and Systems*, 149(1):5–20, 2005.

[143] R. Jensen and Q. Shen. New approaches to fuzzy-rough feature selection. *IEEE Transactions on Fuzzy Systems*, 19(4):824–838, 2009.

[144] R. Jensen, A. Tuson, and Q. Shen. Extending propositional satisfiability to determine minimal fuzzy-rough reducts. In *IEEE World Congress on Computational Intelligence, WCCI 2010*, pages 1–8, 2010.

[145] R. Jensen, A. Tuson, and Q. Shen. Finding rough and fuzzy-rough set reducts with SAT. *Information Sciences*, 255:100–120, 2014.

[146] W. Jin, L. Wang, X. Zeng, Z. Liu, and R. Fu. Classification of clouds in satellite imagery using over-complete dictionary via sparse representation. *Pattern Recognition Letters*, 49(1):193–200, 2014.

[147] M.J. Kane, J. Emerson, and S. Weston. Scalable strategies for computing with massive data. *Journal of Statistical Software*, 55(14):1–19, 2013. `http://www.jstatsoft.org/v55/i14/`.

[148] M.J. Kane, J.W. Emerson, and P. Haverty. *bigmemory: Manage massive matrices with shared memory and memory-mapped files*, 2013. R package version 4.4.6, `http://cran.r-project.org/web/packages/bigmemory/index.html`.

[149] A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

[150] H Karau, A Konwinski, P Wendell, and M Zaharia. *Learning Spark: Lighting-Fast Data Analysis*. O'Reilly, 2015.

[151] N.K. Kasabov and Q. Song. DENFIS: Dynamic evolving neural-fuzzy inference system and its application for time-series prediction. *IEEE Transactions on Fuzzy Systems*, 10(2):144–154, 2002.

[152] KDnuggets. Languages for analytics/data mining. Technical report, 2015. `http://www.kdnuggets.com/2015/05/poll-r-rapidminer-python-big-data-spark.html`.

[153] J.M. Keller, M.R. Gray, and J.R. Givens. A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:580–585, 1985.

[154] R. Kerber. ChiMerge: Discretization of numeric attributes. In *AAAI-92 Proc. Ninth National Conference on Artificial Intelligence*, pages 123–128. AAAI Press/MIT Press, 1992.

[155] J. Kim and N. Kasabov. HyFIS: Adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems. *Neural Networks*, 12(9):1301–1319, 1999.

[156] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. The emerging role of data scientists on software development teams. Technical report, 2015. `http://research.microsoft.com/pubs/242286/MSR-TR-2015-30.pdf`.

[157] A. Kleiner, A. Talwalkar, P. Sarkar, and M. Jordan. The big data bootstrap. *arXiv preprint arXiv:1206.6415*, 2012.

[158] G.J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall PTR, 1995.

[159] J. Komorowski, Z. Pawlak, L. Polwski, and A. Skowron. Rough sets: A tutorial. In S.K. Pal and A. Skowron, editors, *Rough Fuzzy Hybridization, A New Trend in Decision Making*, pages 3–98. Singopore, Springer, 1999.

[160] R. Kovahi. Glossary of terms. *Machine Learning*, 30:271–274, 1998.

[161] P. Kranen, H. Kremer, T. Jansen, T. Seidl, A. Bifet, G. Holmes, B. Pfahringer, and J. Read. Stream data mining using the MOA framework. In *Database Systems for Advanced Applications*, pages 309–313. Springer, 2012.

[162] K. Krawiec, R. Słowiński, and D. Vanderpooten. Learning of decision rules from similarity based rough approximations. In A. Skowron and L. Polkowski, editors, *Rough Sets in Knowledge Discovery vol 2*, pages 37–54. Physica Verlag, Heidelberg, 1998.

[163] M. Kuhn, S. Weston, N. Coulter, and R. Quinlan. *C50: C5.0 Decision Trees and Rule-Based Models*, 2012. R package version 0.1.0-013, `http://cran.r-project.org/web/packages/C50`.

[164] M.B. Kursa. rFerns: An implementation of the random ferns method for general-purpose machine learning. *Journal of Statistical Software*, 61(10):1–13, 2014.

[165] V. Labatut and H. Cherifi. Accuracy measures for the comparison of classifiers. *arXiv preprint arXiv:1207.3790*, 2012.

[166] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[167] V.G. Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8595–8598, 2013.

[168] A. Lewin. *fuzzyFDR: Exact Calculation of Fuzzy Decision Rules for Multiple Testing*, 2007. R package version 1.0.

[169] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[170] A. Lim, L. Breiman, and A. Cutler. *bigrf: Big Random Forests: Classification and Regression Forests for Large Data Sets*, 2014. R package version 0.1-11.

[171] H. Lin, S. Yang, and S.P. Midkiff. Rabid: A distributed parallel R for large datasets. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 725–732. IEEE, 2014.

[172] J. Lin and C. Dyer. Data-intensive text processing with MapReduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177, 2010.

[173] C. Liu, W. Hsaio, C. Lee, T. Chang, and T. Kuo. Semi-supervised text classification with universum learning. *IEEE Transactions on Cybernetics*, 2015. In Press.

[174] H. Liu and R. Setiono. Discretization of ordinal attributes and feature aelection. In *Proc. Seventh International Conference on Tools with Artificial Intelligence*, pages 388–391, Washington DC, 1995.

[175] Y. Liu, Q. Zhou, E. Rakus-Anderson, and G. Bai. A fuzzy-rough sets based compact rule induction method for classifying hybrid data. In *Rough Sets and Knowledge Technology, Lecture Notes in Computer Science*, volume 7414, pages 63–70, 2012.

[176] S. Lohr. In big data, shepherding comes first, 2014. `http://www.nytimes.com/2014/12/15/technology/in-big-data-shepherding-comes-first-.html`.

[177] J. Long. *An R language segue into parallel processing on Amazon's Web Services*, 2012. R package version 0.05, `http://https://code.google.com/p/segue/`.

[178] G. Louppe. Understanding random forests: From theory to practice. Master's thesis, Faculty of Applied Sciences, Department of Electrical Engineering & Computer Science, University of Liége, 2014. `http://arxiv.org/abs/1407.7502`.

[179] M. Lübbecke. Big data saves lives, 2015. `http://edition.cnn.com/2015/02/12/opinion/lbbecke-big-data-saves-lives/`.

[180] T. Lumley. *biglm: Bounded memory linear and generalized linear models*, 2013. R package version 0.9-1, `http://cran.r-project.org/web/packages/biglm/index.html`.

[181] M.C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289, 1977.

[182] G. Malewicz, M.H. Austern, A.J.C. Bik, J.C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[183] E.H. Mamdani. Applications of fuzzy algorithm for control a simple dynamic plant. *Proceedings of the Institution of Electrical Engineers*, 121(12):1585–1588, 1974.

[184] E.H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *Int. J. Man Mach. Stud*, 7:1–13, 1975.

[185] D.P. Mandal, C.A. Murthy, and S.K. Pal. Formulation of a multivalued recognition system. *IEEE Transactions on Systems, Man, and Cybernetics*, 22:607–620, 1992.

[186] McKinsey Global Institute (MGI). Big data: The next frontier for innovation, competition, and productivity. Technical report, 2011. `http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation`.

[187] K. McKusick and S. Quinlan. GFS: evolution on fast-forward. *Communications of the ACM*, 53(3):42–49, 2010.

[188] S. Melnik, A. Gubarev, J.J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: interactive analysis of web-scale datasets. *Communications of the ACM*, 54(6):114–123, 2011.

[189] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D.B. Tsai, M. Amde, S. Owen, et al. MLlib: Machine learning in Apache Spark. *arXiv preprint arXiv:1505.06807*, 2015.

[190] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2012. R package version 1.6-1, `http://CRAN.R-project.org/package=e1071`.

[191] D. Meyer and K. Hornik. Generalized and customizable sets in R. *Journal of Statistical Software*, 31(2):1–27, 2009.

[192] R.S. Michalski. A theory and methodology of inductive learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning*, pages 83–134. Morgan Kaufmann, 1983.

[193] Microsoft. The big bang: How the big data explosion is changing the world, 2013. `http://news.microsoft.com/2013/02/11/the-big-bang-how-the-big-data-explosion-is-changing-the-world/`.

[194] R. Mienko, J. Stefanowski, K. Tuomi, and D. Vanderpooten. Discovery-oriented induction of decision rules. *Cahier du Lamsade no. 141*, 1996.

[195] T. M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[196] F.J. Moreno-Velo, A. Barriga, S. Sanchez-Solano, and I. Baturone. XFSML: An XML-based modeling language for fuzzy systems. In *Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*, pages 1–8. IEEE, 2012.

[197] D. Morent, K. Stathatos, W. Lin, and M. Berthold. Comprehensive PMML preprocessing in KNIME. In *Proc. of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2011.

[198] C. Moretti, J. Bulosan, D. Thain, and P.J. Flynn. All-pairs: An abstraction for data-intensive cloud computing. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–11. IEEE, 2008.

[199] A.C. Murthy, V.K. Vavilapalli, D. Eadline, J. Niemiec, and J. Markham. *Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2*. Pearson Education, 2013.

[200] H. Naessend, H.D. Meyer, and B.D. Baets. Algorithms for the computation of T-transitive closures. *IEEE Transactions on Fuzzy Systems*, 10:541–551, 2002.

[201] E. Neisser. *Cognitive Psychology*. New York: Appleton-Century-Crofts, 1967.

[202] D. Newby, A.A. Freitas, and T. Ghafourian. Decision trees to characterise the roles of permeability and solubility on the prediction of oral absorption. *European journal of medicinal chemistry*, 90:751–765, 2015.

[203] NexR. *RHive: R and Hive*, 2014. R package version 2.0-0.2, `http://cran.r-project.org/web/packages/RHive/index.html`.

[204] H.S. Nguyen and S.H. Nguyen. Discretization methods in data mining. In A. Skowron and L. Polkowski, editors, *Rough Sets in Knowledge Discovery*, volume 1, pages 451–482. Physica Verlag, Heidelberg, 1998.

[205] S.H. Nguyen. On efficient handling of continuous attributes in large data bases. *Fundamenta Informaticae*, 48:61–81, 2001.

[206] S.H. Nguyen and A. Skowron. Quantization of real-valued attributes. In P.P. Wang, editor, *Second Annual Joint Conference on Information Sciences (JCIS'95)*, pages 34–37, Wrightsville Beach, North Carolina, 1995.

[207] NIST. Big data interoperability framework: Volume 1, definitions, 2013. `http://jtc1bigdatasg.nist.gov/_uploadfiles/N0028_NBD-PWG_Vol1-Definitions_V1Draft_Pre-release.pdf`.

[208] H. Nomura, I. Hayashi, and N. Wakami. A learning method of fuzzy inference rules by descent method. *IEEE International Conference on Fuzzy Systems*, pages 203–210, 1992.

[209] A. Øhrn. ROSETTA–A rough set toolkit for analysis of data. Technical report, 2009. `http://www.lcb.uu.se/tools/rosetta/`.

[210] A. Øhrn and J. Komorowski. ROSETTA–A rough set tool kit for analysis of data. In *Proceedings of the fifth International Workshop on Rough Sets and Soft Computing (RSSC'97) at the Third Joint Conference on Information Sciences (JCIS'97), Research Triangle Park, NC*, pages 403–407, 1997.

[211] S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in action*. Manning, 2011.

[212] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. Ieee, 2007.

[213] Z. Pawlak. Rough sets. *International Journal of Computer Sciences*, 11:341–356, 1982.

[214] Z. Pawlak. *Rough sets–Theoretical aspects of reasoning about data*. Kluwer Academic, 1991.

[215] Z. Pawlak and A. Skowron. Rough sets and boolean reasoning. *Information Sciences*, 177:41–73, 2007.

[216] Z. Pawlak and A. Skowron. Rudiments of rough sets. *Information Sciences*, 177:3–27, 2007.

[217] C.A. Peña Reyes. Coevolutionary fuzzy modelling. Master's thesis, Faculté Informatique et Communications, École Polytechnique Fédérale De Lausanne, 2002. http://library.epfl.ch/en/theses/?nr=2634.

[218] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[219] W. Pedrycz. *Fuzzy Modelling: Paradigms and Practice*. Kluwer Academic Press, 1996.

[220] W. Pedrycz and F. Gomide. *An Introduction to Fuzzy Sets: Analysis and Design*. The MIT Press, 1998.

[221] R.D Peng. Reproducible research in computational science. *Science*, 334:1226–1227, 2011.

[222] A. Peters, T. Hothorn, B.D. Ripley, T. Therneau, and B. Atkinson. *ipred: Improved Predictors*, 2015. R package version 0.9-4.

[223] J. Piaget. *Six Psychological Studies*. New York: Random House, 1967.

[224] N.M. Portela, G.D.C. Cavalcanti, and T.I. Ren. Semi-supervised clustering for MR brain image segmentation. *Expert Systems with Applications*, 41(4 PART 1):1492–1497, 2014.

[225] B. Predki, R. Słowiński, J. Stefanowski, R. Susmaga, and S. Wilk. ROSE - software implementation of the rough set theory. In L. Polkowski and A. Skowron, editors, *Proceedings of the Rough Sets and Current Trends in Computing'98 Conference, Lecture Notes in Artificial Intelligence*, volume 1424, pages 605–608. Springer, Berlin, 1998.

[226] B. Predki and S. Wilk. Rough set based data exploration using ROSE system. In Z.W. Ras and A. Skowron, editors, *Foundations of Intelligent Systems, Lecture Notes in Artificial Intelligence*, volume 1609, pages 172–180. Springer-Verlag, Berlin, 1999.

[227] Gill Press. A very short history of data science?, 2013. http://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/.

[228] H. Qian. PivotalR: A package for machine learning on big data. *The R Journal*, 6(1):57–67, 2014.

[229] J.R. Quinlan. *Discovering Rules by Induction from Large Collections of Examples*. Expert systems in the micro electronic age. Edinburgh University Press, 1979.

[230] J.R. Quinlan. *Learning Efficient Classification Procedures and Their Application to Chess End Games*. Springer Berlin Heidelberg, 1983.

[231] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[232] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993. `http://www.rulequest.com/see5-unix.html`.

[233] R Core Team, R. Bivand, V.J. Carey, S. DebRoy, S. Eglen, R. Guha, N. Lewin-Koh, M. Myatt, B. Pfaff, B. Quistorff, F. Warmerdam, S. Weigand, and Free Software Foundation Inc. *foreign: Read data stored by Minitab, S, SAS, SPSS, Stata, Systat, Weka, dBase*, 2014. R package version 0.8-61, `http://CRAN.R-project.org/package=foreign`.

[234] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008.

[235] R Development Core Team. *R Installation and Administration*. R Foundation for Statistical Computing, Vienna, Austria, 2008.

[236] R Development Core Team. *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria, 2008.

[237] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H. Jacobsen, and S. Mankovskii. Solving big data challenges for enterprise application performance management. *Proceedings of the VLDB Endowment*, 5(12):1724–1735, 2012.

[238] A.M. Radzikowska and E.E. Kerre. A comparative study of fuzzy rough sets. *Fuzzy Sets and Systems*, 126:137–156, 2002.

[239] Revolution Analytics. *Revolution R Enterprise DistributedR: Portable Power: Big Data Analytics For the Entire IT Infrastructure*, 2014. R package version 1.0.0, `http://www.revolutionanalytics.com/revolution-r-enterprise-distributedr`.

[240] Revolution Analytics. *RHadoop*, 2015. https://github.com/RevolutionAnalytics/RHadoop/wiki.

[241] Revolution Analytics and S. Weston. *foreach: Foreach looping construct for R*, 2014. R package version 1.4.2, http://cran.r-project.org/web/packages/foreach/index.html.

[242] B. Ripley. *nnet: Feed-forward neural networks and multinomial log-linear models*, 2012. R package version 7.3-5, http://www.stats.ox.ac.uk/pub/MASS4.

[243] B. Ripley. *tree: Classification and Regression Trees*, 2012. R package version 1.0-33, http://cran.r-project.org/web/packages/tree.

[244] L.S. Riza, C. Bergmeir, F. Herrera, and J.M. Benítez. *frbs: Fuzzy Rule-Based Systems for Classification and Regression Tasks*, 2014. R package version 3.0-0.

[245] L.S. Riza, A. Janusz, C. Bergmeir, C. Cornelis, F. Herrera, D. Ślezak, J.M. Benítez, et al. *RoughSets: Data Analysis Using Rough Set and Fuzzy Rough Set Theories*, 2015. R package version 1.2-0.

[246] M. Robnik-Sikonja and P. Savicky. *CORElearn: CORElearn Classification, Regression, Feature Evaluation and Ordinal Evaluation*, 2013. R package version 0.9.41, http://lkm.fri.uni-lj.si/rmarko/software/.

[247] S. Romanski. Operation on families of sets for exhaustive search, given a monotonic function. In W. Beeri, C. Schmidt, and N. Doyle, editors, *Proceedings of the 3rd Int. Conference on Data and Knowledge Bases*, pages 310–322, 1988.

[248] D.S. Rosenberg. *HadoopStreaming: Utilities for using R scripts in Hadoop streaming*, 2012. R package version 0.2, http://cran.r-project.org/web/packages/HadoopStreaming/index.html.

[249] J.M.F. Salido and S. Murakami. Rough set analysis of a general type of fuzzy data using transitive aggregations of fuzzy similarity relations. *Fuzzy Sets Systems*, 139:635–660, 2003.

[250] M. Sarkar. Fuzzy-rough nearest neighbors algorithm. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 5, pages 3556–3561, 2000.

[251] M. Sarkar. Fuzzy-rough nearest-neighbor algorithm in classification. *Fuzzy Sets and System*, 158:2123–2152, 2007.

[252] M. Sarkar. Fuzzy-rough nearest algorithms in classification. *Fuzzy Sets and Systems*, 158:2134–2152, 2012.

[253] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656, 1948.

[254] Q. Shen and A. Chouchoulas. A modular approach to generating fuzzy rules with reduced attributes for the monitoring of complex systems. *Engineering Applications of Artificial Intelligencee*, 13:263–278, 2000.

[255] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[256] B.F. Skinner. *The Behavior of Organisms: An Experimental Analysis*. Oxford, England: Appleton-Century, 1938.

[257] B.F. Skinner. The evolution of behavior. *Journal of the Experimental Analysis of Behavior*, 41:217–221, 1984.

[258] A. Skowron and C. Rauszer. The discernibility matrices and functions in information systems. In R. Słowiński, editor, *Intelligent Decision Support: Handbook of Applications and Advances of Rough Sets Theory*, pages 331–362, Dordrecht, Netherland, 1992. Kluwer Academic Publisherst.

[259] D. Ślęzak. Approximate bayesian networks. In B. Bounchon-Meunier, J. Gutierrez-Rios, L. Magdalena, and R.R. Yager, editors, *Technologies for Constructing Intelligent Systems 2: Tools*, pages 313–326. Springer Verlag, 2002.

[260] D. Ślęzak. Approximate entropy reducts. *Fundamenta Informaticae*, 53:365–390, 2002.

[261] R. Słowiński and J. Stefanowski. Rough set reasoning about uncertain data. *Fundamenta Informaticae*, 27(2–3):229–244, 1996.

[262] R. Słowiński and D. Vanderpooten. Similarity relation as a basis for rough approximations. In P.P. Wang, editor, *Advances in Machine Intelligence and Soft Computing*, pages 17–33. Bookwrights, Raleigh, NC, 1997.

[263] R. Słowiński and D. Vanderpooten. A generalized definition of rough approximations based on similarity. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):331–336, 2000.

[264] B.J. Smith. *magma: Matrix algebra on GPU and multicore architectures*, 2013. R package version 1.3.0-2, `http://cran.r-project.org/web/packages/magma/index.html`.

[265] D. Smith and J. Rickert. RevoScaleR: Big data analysis for R using Revolution R Enterprise, 2010. `http://www.revolutionanalytics.com/sites/default/files/big-data-wp.pdf`.

[266] J.M. Stanton. *Introduction to data science*. Syracuse University, 2013. `http://surface.syr.edu/istpub/165/`.

[267] J. Stefanowski. On rough set based approaches to induction of decision rules. In L. Polkowski and A. Skowron, editors, *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, pages 500–529. Heidelberg: Physica-Verlag, 1998.

[268] J. Stefanowski and D. Vanderpooten. A general two stage approach to rule induction from examples. In W. Ziarko, editor, *Rough Sets, Fuzzy Sets and Knowledge Discovery*, pages 317–325. Springer-Verlag, 1994.

[269] M. Stokely and T. Hesterberg. *HistogramTools: Utility Functions for R Histograms*, 2014. R package version 0.3.1, `http://cran.r-project.org/web/packages/HistogramTools/index.html`.

[270] P. Stone, R.S. Sutton, and G. Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[271] M. Sugeno and G.T. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, 28:15–33, 1988.

[272] M. Sugeno and T. Yasukawa. A fuzzy-logic-based approach to qualitative modeling. *IEEE Transactions on Fuzzy Systems*, 1(1):7–31, 1993.

[273] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 51(1):116–132, 1985.

[274] D.K. Tayal, A. Jain, S. Arora, S. Agarwal, T. Gupta, and N. Tyagi. Crime detection and criminal identification in india using data mining techniques. *AI and Society*, 30(1):117–127, 2014.

[275] The Apache Software Foundation. Apache HBase. http://hbase.apache.org/.

[276] The Apache Software Foundation. Apache HIVE. http://hive.apache.org/index.html.

[277] The MathWorks, Inc. The fuzzy logic toolbox for use with MATLAB version 2, 2002. http://www.mathworks.com/help/pdf_doc/fuzzy/fuzzy.pdf.

[278] P. Thrift. Fuzzy logic synthesis with genetic algorithms. In *Proc. of the Fourth International Conf. on Genetic Algorhtms (ICGA91)*, pages 509–513, 1991.

[279] L. Tierney, A.J. Rossini, N. Li, and H. Sevcikova. *snow: Simple network of workstations*, 2013. R package version 0.3-13, http://cran.r-project.org/web/packages/snow/index.html.

[280] Y. Tillé and A. Matei. *sampling: Survey sampling*, 2013. R package version 2.6, http://cran.r-project.org/web/packages/sampling/index.html.

[281] M. Troester. Big data meets big data analytics. Technical report, 2015. http://www.sas.com/resources/whitepaper/wp_46345.pdf.

[282] E.C.C. Tsang, D.G. Chen, D.S. Yeung, X.Z. Wang, and J.W.T. Lee. Attributes reduction using fuzzy rough sets. *IEEE Transactions on Fuzzy Systems*, 16:1130–1141, 2008.

[283] S. Tsumoto. Automated induction of medical expert system rules from clinical databases based on rough set theory. *Information Sciences*, 112:67–84, 1998.

[284] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[285] N. Verbiest, C. Cornelis, and F. Herrera. A fuzzy rough prototype selection method. *Pattern Recognition*, 46:2770–2782, 2013.

[286] N. Verbiest, C. Cornelis, and R. Jensen. Fuzzy-rough positive region based nearest neighbour classification. In *Proceedings of the 20th International Conference on Fuzzy Systems (FUZZ-IEEE 2012)*, pages 1961–1967, 2012.

[287] Voldemort. Project voldemort. http://project-voldemort.com/.

[288] M. Wall. Ebola: Can big data analytics help contain its spread?, 2014. http://www.bbc.com/news/business-29617831.

[289] P. Walmsley. *Definitive XML Schema*. Prentice Hall PTR, 2012.

[290] G. Wang, Z. Zheng, and Y. Zhang. RIDAS - a Rough Set Based Intelligent Data Analysis System. In *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, volume 2, pages 646–649 vol.2, 2002.

[291] L.X. Wang. *Adaptive Fuzzy Systems and Control: Design and Analysis*. Prentice-Hall, 1994.

[292] L.X. Wang and J.M. Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1414–1427, 1992.

[293] X. Wang, J. Yang, X. Teng, W. Xia, and R. Jensen. Feature selection based on rough sets and particle swarm optimization. *Pattern Recognition Letters*, 28(4):459–471, 2007.

[294] T. White. *Hadoop: The Definitive Guide*. " O'Reilly Media, Inc.", 2015.

[295] H. Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1):1–29, 2011. http://www.jstatsoft.org/v40/i01/.

[296] H. Wickham. *Advanced R*. CRC Press, 2015.

[297] J. Wijffels and BNOSAC. *RMOA: Connect R with MOA for massive online analysis*, 2014. R package version 1.0, http://cran.r-project.org/web/packages/RMOA/index.html.

[298] M. Wojnarski. LTF-C: Architecture, training algorithm and applications of new neural classifier. *Fundamenta Informaticae*, 54(1):89–105, 2003.

[299] World Wide Web Consortium. *XML 1.0 Specification*, 2014. Retrieved 2014-10-4, http://www.w3.org/TR/xml/.

[300] J. Wróblewski. Covering with reducts - a fast algorithm for rule generation. In *Proceeding of RSCTC'98, LNAI 1424*, pages 402–407. Springer Verlag, Berlin, 1998.

[301] J. Wróblewski. Ensembles of classifiers based on approximate reducts. *Fundamenta Informaticae*, 47:351–360, 2001.

[302] Z. Wu, G. Eadon, S. Das, E.I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan. Implementing an inference engine for RDFS/OWL constructs and user-defined rules in Oracle. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1239–1248. IEEE, 2008.

[303] D. Wuerts and et al. *fRegression: Regression based decision and prediction*, 2012. R package version 2160.77, `http://www.rmetrics.org`.

[304] X. Xie, J. Cao, H. Jin, X. Ke, and W. Cao. JRBridge: A framework of large-scale statistical computing for R. In *Services Computing Conference (APSCC), 2012 IEEE Asia-Pacific*, pages 27–34. IEEE, 2012.

[305] R. Yager and D. Filev. Generation of fuzzy rules by mountain clustering. *Journal of Intelligent and Fuzzy Systems*, 2(3):209–219, 1994.

[306] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[307] L.A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning - part i. *Information Sciences*, 8(3):199–249, 1975.

[308] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[309] S.Y. Zhao, E.C.C. Tsang, and D.G. Chen. The model of fuzzy variable precision rough sets. *IEEE Transactions on Fuzzy Systems*, 17:451–467, 2009.

[310] S.Y. Zhao, E.C.C. Tsang, D.G. Chen, and X.Z. Wang. Building a rule-based classifier – a fuzzy-rough set approach. *IEEE Trans. on Knowledge and Data Engineering*, 22:624–638, 2010.

[311] Q. Zhong, A.G. Busetto, J.P. Fededa, J.M. Buhmann, and D.W. Gerlich. Unsupervised modeling of cell morphology dynamics for time-lapse microscopy. *Nature Methods*, 9:711–713, 2012.

[312] W. Ziarko. Analysis of uncertain information in the framework of variable precision rough sets. *Foundation of Computing and Decision Sciences*, 18(3–4):381–396, 1993.