UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)

Tesis doctoral

EVOLUTIONARY LEARNING OF FUZZY RULES FOR REGRESSION

Presentada por:

Ismael Rodríguez Fernández

Dirigida por:

Alberto J. Bugarín Diz
Manuel Mucientes Molina

Santiago de Compostela, septiembre de 2016

Alberto J. Bugarín Diz, Catedrático de Universidad del Área de Ciencia de la Computación e Inteligencia Artificial de la Universidade de Santiago de Compostela

Manuel Mucientes Molina, Profesor Contratado Doctor del Área de Ciencia de la Computación e Inteligencia Artificial de la Universidade de Santiago de Compostela

HACEN CONSTAR:

Que la memoria titulada **EVOLUTIONARY LEARNING OF FUZZY RULES FOR REGRESSION** ha sido realizada por **Ismael Rodríguez Fernández** bajo nuestra dirección en el Centro Singular de Investigación en Tecnoloxías da Información de la Universidade de Santiago de Compostela, y constituye la Tesis que presenta para obtar al título de Doctor.

Santiago de Compostela, septiembre de 2016

Alberto J. Bugarín Diz
Director de la tesis

Manuel Mucientes Molina
Director de la tesis

Ismael Rodríguez Fernández
Autor de la tesis

Alberto J. Bugarín Diz, Catedrático de Universidad del Área de Ciencia de la Computación e Inteligencia Artificial de la Universidade de Santiago de Compostela

Manuel Mucientes Molina, Profesor Contratado Doctor del Área de Ciencia de la Computación e Inteligencia Artificial de la Universidade de Santiago de Compostela

como Director/res de la tesis titulada:

EVOLUTIONARY LEARNING OF FUZZY RULES FOR REGRESSION

Por la presente DECLARAN:

Que la tesis presentada por Don **Ismael Rodríguez Fernández** es idónea para ser presentada, de acuerdo con el artículo 41 del *Regulamento de Estudos de Doutoramento*, por la modalidad de compendio de ARTÍCULOS, en los que el doctorando ha tenido participación en el peso de la investigación y su contribución fue decisiva para llevar a cabo este trabajo. Y que está en conocimiento de los coautores, tanto doctores como no doctores, participantes en los artículos, que ninguno de los trabajos reunidos en esta tesis serán presentados por ninguno de ellos en otras tesis de Doctorado, lo que firmamos bajo nuestra responsabilidad.

Santiago de Compostela, septiembre de 2016

Alberto J. Bugarín Diz
Director de la tesis

Manuel Mucientes Molina
Director de la tesis

Do or do not, there is no try.

Yoda

Now, I suppose this is the time for me to say something profound... Nothing comes to mind.

Jack O'Neill

Agradecimientos

En primer lugar, quiero expresar mi más sincero agradecimiento a mis directores, Dr. Alberto J. Bugarín Diz y Dr. Manuel Mucientes Molina, por su confianza y consejos que me han brindado durante estos años.

Al Departamento de Electrónica y Computación y al Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS) por proporcionar los recursos necesarios para la realización de esta tesis. A los miembros del centro con los que he aprendido y entablado una gran relación, especialmente estoy agradecido al director comisario Dr. Paulo Félix. También quisiera agradecer al Dr. Jaume Bacardit y al Newcastle Computer Science Department por acogerme durante los tres meses de estancia en los que he estado lejos da miña terra.

Me gustaría agradecer la financiación que me ha permitido desarrollar esta investigación: programa de ayudas Formación de Profesorado Universitario (FPU AP2010-0627) financiado por el Ministerio de Educación; proyectos TIN2008-00040, TIN2011-22935, TIN2011-29827-C02-02 y TIN2014-56633-C3-1-R financiados por el Ministerio de Economía y Competitividad y el Fondo Europeo para el Desarrollo Regional (FEDER); proyectos EM2014/12, GRC2014/030 y CN2012/151 financiados por la Xunta de Galicia y el FEDER.

Gracias también a mis compañeros del Laboratorio S2 y allegados por todas las experiencias compartidas tanto en los buenos como en los malos momentos.

Quisiera dar un agradecimiento especial a Belén, que siempre ha estado ahí en los momentos más difíciles y su constante apoyo me ha ayudado a seguir adelante.

Y por último querría dar las gracias a mi familia: a mis padres y mis hermanos. Gracias por ayudarme a crecer como persona, sin vosotros no podría haber llegado a donde estoy ahora.

Santiago de Compostela, septiembre de 2016

Resumen

Debido a la gran cantidad de datos que se generan todos los días, existe la necesidad de explotarlos para obtener información a partir de ellos, procesándolos y extrayendo automáticamente los patrones y tendencias relevantes presentes en los mismos. Esta tarea se denomina aprendizaje a partir de datos, siendo el aprendizaje máquina el campo de las ciencias de la computación que se centra en el desarrollo de los algoritmos que construyen automáticamente los modelos para predicción y toma de decisiones. Existen dos categorías principales de problemas de aprendizaje:

- Aprendizaje supervisado: el objetivo es crear modelos que predigan el valor de una variable de salida a partir de un conjunto de medidas de entrada.
- Aprendizaje no supervisado: no existe variable de salida, por lo que el objetivo es aprender cómo se organizan los datos en sí.

Dentro del aprendizaje supervisado, dependiendo del tipo de salida, existen dos tareas diferentes: clasificación y regresión. En clasificación, los datos se dividen entre varias categorías o clases, y el proceso de aprendizaje debe producir un modelo que asigne a las nuevas entradas su categoría correspondiente. Por otro lado, en un problemas de regresión, la variable de salida es un valor real continuo en vez de un valor discreto, por lo que el modelo debe generar un valor real de salida a partir de las entradas.

Los modelos obtenidos a través de técnicas de aprendizaje máquina suelen tener dos requisitos complementarios:

- Precisión: indica la capacidad del modelo de predecir valores cercanos a los verdaderos.
- Interpretabilidad: la capacidad del modelo de ser entendido por los seres humanos, la cual está relacionada con lo complejo que es el modelo.

El objetivo es encontrar un buen equilibrio entre la complejidad del modelo y la reducción del error de entrenamiento para obtener, finalmente, un buen error de test. El error de entrenamiento tiende a decrementar según el modelo se vuelve más complejo, esto es, cuando el modelo se ajusta más a los datos (sobreaprendizaje). Sin embargo, cuando la complejidad del modelo sobrepasa cierto umbral, aunque el error de entrenamiento disminuye, el modelo se ajusta tanto a los datos de entrenamiento que se equivoca en los datos de test. Por el contrario, si el modelo no es lo suficientemente complejo, no es capaz de representar el conocimiento intrínseco de los datos, resultando en una mala precisión y pobre generalización.

De entre todos los tipos de modelos utilizados en aprendizaje máquina, los más interpretables por el ser humano son los árboles de decisión y su representación como sentencias condicionales o reglas. Ambos representan el conocimiento a través de condiciones que, dada una entrada que las cumple, permiten estimar la salida. Además, el razonamiento realizado por humanos es impreciso por naturaleza y, en muchos casos, los datos contienen cierto grado de incertidumbre. La lógica borrosa es una técnica dentro del campo de la computación flexible que trabaja con este tipo de incertidumbres e imprecisiones, proporcionando un sistema donde la verdad de los valores se representa como un número real entre 0 y 1. Por ello, el uso de reglas borrosas está muy extendido, debido a la combinación de interpretabilidad y expresividad de las reglas con la capacidad de representar incertidumbre de la lógica borrosa.

Los Sistemas Basados en Reglas Borrosas (FRBS, *Fuzzy Rule Based Systems*) utilizan proposiciones borrosas tanto en el antecedente como el consecuente de las reglas. Los FRBS se componen de cuatro partes diferentes: la Base de Conocimiento (*Knowledge Base*, KB), la interfaz de borrosificación, el sistema de inferencia y la interfaz de desborrosificación. La KB contiene el modelo, compuesto por la Base de Datos (DB, *DataBase*) — la definición de los conjuntos borrosos utilizados en el sistema — y la Base de Reglas (RB, *Rule Base*) — las sentencias condicionales que utilizan la información de la DB en sus proposiciones. Los valores de entrada se introducen en el sistema a través de la interfaz de borrosificación, donde los valores en crudo son convertidos a conjuntos borrosos definidos en la DB. Después, el sistema de inferencia calcula el grado de pertenencia de la parte antecedente de las reglas y dispara los consecuentes correspondientes para calcular la salida. Finalmente, las salidas borrosas de la RB se transforman a valores crudos para obtener la salida final del sistema.

La definición de los conjuntos borrosos utilizados en la DB es uno de los aspectos más importantes de un sistema borroso en términos de interpretabilidad, la cual puede seguir dos aproximaciones diferentes:

- La aproximación lingüística define la división de cada variable en conjuntos borrosos de manera global, donde cada división se denomina partición borrosa. Los conjuntos borrosos dentro de una partición borrosa se pueden asociar a etiquetas lingüísticas, por lo que le da mayor interpretabilidad humana al sistema. Sin embargo, esto limita los grados de libertad del sistema y, por lo tanto, su capacidad de aproximar el problema de manera más precisa.
- La aproximación aproximativa utiliza diferentes definiciones de conjuntos borrosos por cada proposición de cada regla. Por lo tanto, no existe una definición global de partición borrosa de las variables, y no se puede asociar términos lingüísticos a las mismas. Gen-lingüística, esta aproximación se utiliza cuando es necesario soluciones muy precisas, aunque puede llevar a particiones complejas del espacio de entrada y dificultar la comprensión de la relación entre las entradas y la respuesta del sistema.

Takagi, Sugeno y Kang propusieron un modelo de reglas borrosas, denominado TSK, donde los antecedentes se componen de variables lingüísticas mientras que el consecuente se representa como una función polinómica de las entradas. La función polinómica más comúnmente utilizada en los consecuentes de reglas TSK es la combinación lineal de las entradas (TSK-1), donde el polinomio es de grado 1. Este tipo de reglas combinan la interpretabilidad de las reglas borrosas con la precisión de los modelos de regresión, mejorando la precisión del sistema. Los sistemas lingüísticos de reglas borrosas TSK representan un buen equilibrio entre precisión e interpretabilidad:

- El uso de términos lingüísticos en el antecedente de las reglas proporciona una descripción completa del espacio de entrada, dada la definición global de las particiones borrosas en la DB.
- La representación lineal de la salida puede obtener soluciones precisas utilizando métodos estadísticos conocidos en la literatura.
- El consecuente de las reglas, representado como una combinación lineal de las variables de entrada, permite una comprensión fácil de la relación entre las entradas y la salida.

Por ello, aunque los sistemas de reglas borrosas TSK son menos entendibles en términos de lenguaje natural, pueden proporcionar información útil y comprensiva, siendo la opción preferida en muchos campos, como la robótica, imagen médica, estimaciones industriales u optimización de procesos.

El aprendizaje automático de FRBS se ha aproximado en la literatura utilizando diferentes técnicas de aprendizaje máquina, como por ejemplo Redes Neuronales. Sin embargo, la aproximación más existosa ha sido mediante el uso de Algoritmos Evolutivos, particularmente Algoritmos Genéticos. Un algoritmo genético es una técnica de búsqueda heurística que imita el proceso de selección natural, utilizando operaciones como cruce, mutación y selección para generar un conjunto de soluciones posibles que son optimizadas a lo largo de varias iteraciones hasta que se alcanza algún criterio de convergencia. La característica más importante de los algoritmos genéticos es su capacidad de explorar espacios de búsqueda muy grandes y su flexibilidad a la hora de incorporar conocimiento a priori en prácticamente cualquier parte del algoritmo: la representación de las soluciones para parecerse a los modelos a aprender, los operadores genéticos para conseguir que algoritmo converja a soluciones prometedores, etc. La combinación de algoritmos genéticos con FRBS ha generado un nuevo campo dentro de la computación flexible denominado Sistemas Genético-Borrosos (GFS, *Genetic Fuzzy Systems*).

La flexibilidad de los algoritmos genéticos permite la codificación de cualquier parte de los FRBS y, según como se represente la RB, existen tres aproximaciónes de GFS diferentes:

- Michigan: cada solución individual representa una única regla, y la población en su conjunto representa la RB, evolucionando todas las reglas al mismo tiempo. Esta aproximación tiene la desventaja de la falta de colaboración entre cada una de las soluciones, lo que provoca que varias reglas pueden estar compitiendo por representar el mismo conocimiento.
- Pittsburgh: esta aproximación intenta resolver el problema de la cooperación-competición codificando la KB entera en una única solución individual. Sin embargo, esto puede generar cromosomas complejos, dificultando la definición de los operadores genéticos. Este problema fue solucionado recientemente representando únicamente la DB y generando la RB mediante un método ad-hoc.
- *Iterative Rule Learning*: en esta aproximación, cada individuo representa una única regla, pero en vez de generar la RB usando toda la población, solo obtiene una única regla al final del proceso evolutivo. Después, el algoritmo se repite quitando aquellos ejemplos del conjunto de entrenamiento cubiertos por las reglas ya generadas, hasta que ya no queden ejemplos. Esta aproximación suele generar muchas reglas que cubren muy poco espacio de entrada, y por lo tanto suele utilizarse un proceso de selección de reglas posteriormente.

Particularmente, la flexibiliadad de los algoritmos genéticos permite gestionar el equilibrio entre la precisión y la interpretabilidad del modelo de una manera efectiva. La intepretabilidad de un FRBS involucra dos cuestiones principales:

- Legibilidad: está relacionado con la simplicidad de la estructura del sistema borroso: número de variables, términos lingüísticos por variables, número de reglas, antecedentes por regla, etc. Representa la parte cuantitativa u objetiva de la interpretabilidad del modelo.
- Comprensibilidad: se determina por la semántica del sistema borroso y el mecanismo de inferencia. Está asociado con el particionamiento borroso de las variables y su significado para el usuario final, por lo que representa la parte cualitativa o subjetiva de la intepretabilidad.

La legibilidad se suele transformar en una medida cuantitativa que se incorpora a la función de evaluación dentro del GFS. Sin embargo, la comprensibilidad es mucho más sutil, y solo algunas características, como particiones borrosas fuertes o un bajo número de conjuntos borrosos por cada partición borrosa, pueden mejorarla.

En la actualidad no existen GFSs que aprendan FRBSs capaces de resolver problemas de regresión a gran escala de manera general y que tengan un nivel de interpretabilidad suficiente que los haga comprensibles para usuarios no expertos. La simplicidad de los modelos obtenidos por GFS para regresión suele alcanzarse a través del control del número de reglas y/o número de etiquetas utilizados en la RB mediante una aproximación multi-objetivo. Sin embargo, recientemente, también se utilizan técnicas de selección de instancias, ya que pueden afrontar dos problemas al mismo tiempo: disminuye la complejidad de los problemas a gran escala y reduce el sobreaprendizaje, ya que se puede generar la RB con una parte del conjunto de entrenamiento y evaluar su error con otra parte. Además del número de reglas y número de etiquetas, un aspecto muy importante en la intepretabilidad de los FRBS es la definición de las particiones borrosas de la DB. Una de las técnicas más utilizadas es la aproximación multi-granular, donde el universo de discurso de una variable se divide en un número diferente de etiquetas por cada granularidad, obteniendo una partición borrosa diferente según el número de etiquetas. Además, cuando no está disponible conocimiento experto que determine cómo deben ser las etiquetas borrosas, se pueden seguir dos aproximaciones: discretización uniforme combinada con desplazamiento lateral, o discretización no uniforme.

El uso de bases de reglas TSK implica otra dimensión de complejidad: el polinomio del consecuente — normalmente de grado 1 (TSK-1) o 0 (TSK-0). La aproximación más uti-

lizada en el aprendizaje de los coeficientes del polinomio es el uso del método de mínimos cuadrados. Sin embargo, esta opción suele obtener modelos sobreentrenados que se comportan errónamente en test. Este problema se puede resolver mediante el control del valor de los coeficientes para que no sea muy grande (regularización de Ridge) o poniendo algunos coeficientes a 0 (regularización de Lasso), disminuyendo la complejidad del modelo. Además, es posible realizar una combinación de ambas regularizaciones mediante el método Elastic Net.

Uno de los principales inconvenientes de los GFS es su coste computacional. En general, la adaptación de aproximaciones de aprendizaje máquina a grandes conjuntos de datos ha conllevado un enorme desafío en los últimos años. En una aproximación GFS, el tamaño de los datos tiene una gran influencia en el rendimiento de los modelos generados. Las reglas borrosas aprendidas sufren de una explosión exponencial en el número de reglas cuando incrementa el número de ejemplos o variables. Por ello, cuando el espacio de búsqueda es muy grande, el tiempo de convergencia para el aprendizaje de modelos simples incrementa de manera no sostenible. Además, los algoritmos evolutivos son costosos computacionalmente por si mismos debido al gran número de evaluaciones necesarias para converger y, en muchos casos, el proceso de evaluación puede llevar mucho tiempo. Las técnicas utilizadas para mejorar la escalabilidad de GFS se pueden clasificar en tres categorías:

- Orientadas al algoritmo, que adaptan la estructura del algoritmo evolutivo.
- Orientadas a los datos, que modifican el conjunto de entrenamiento para reducir el coste computacional del aprendizaje.
- Aproximaciones distribuídas, que aprovechan la capacidad de procesamiento de un conjunto de máquinas para reducir el tiempo de ejecución.

El escalado del proceso de aprendizaje orientado a algoritmos puede realizarse mediante el control del tamaño del espacio de búsqueda, disminuyendo el número de reglas o el número de etiquetas utilizadas mediante aproximaciones multi-objetivo. Por otro lado, en los últimos años, las aproximaciones orientadas a los datos han recibido mayor antención mediante el uso de técnicas de selección de instancias que disminuyen la complejidad de problemas de gran escala y evitan el sobreaprendizaje.

Desde un punto de vista Big Data, las aproximaciones distribuídas son las más apropiadas para escalar algoritmos GFS. Sin embargo, muy pocos trabajos han utilizado plataformas Big Data para solucionar los problemas de escalado de los GFS. De entre los métodos más utilizados, los más populares son: i) MPI (*Message Passing Interface*) que explota efi-

cientemente arquitecturas de clústeres multi-núcleo, y ii) Apache Spark, una plataforma recientemente desarrollada que puede ser ejecutada sobre clústers tradicionales como Hadoop. El uso extendido de Spark está intrínsicamente ligado al éxito de Hadoop, el cual procesa grandes volúmenes de datos en paralelo mediante el uso de *Hadoop Distributed File System*. Hadoop también popularizó el uso de la aproximación MapReduce, una metodología de procesamiento distribuído basado en la definición de dos funciones diferentes: la función Map, que distribuye un bloque de datos entre varios nodos ejecutando el mismo proceso en cada partició; y la función Reduce, que agrega los resultados de las funciones Map a través de una representación Clave-Valor de los resultados. Spark añade a esta arquitectura la capacidad de utilizar otro tipo de flujos de datos con mejora de la computación en memoria, junto con un conjunto de funciones de alto nivel que facilitan la construcción de aplicaciones distribuídas.

El objetivo de esta tesis es el diseño de GFSs que aprendan FRBSs capaces de resolver problemas de regresión de una manera general. Particularmente, la meta es obtener modelos de baja complejidad a la vez que mantienen una gran precisión, sin hacer uso de conocimiento experto sobre el problema a resolver. Esto significa que los GFSs deben trabajar con datos en bruto, esto es, sin ningún tipo de preprocesamiento que ayude al proceso de aprendizaje a resolver un problema en concreto. Esto es particularmente interesante cuando no hay disponible conocimiento sobre los datos de entrada o para realizar una primera aproximación al problema. Además, los GFSs deben ser capaces de trabajar con problemas de gran escala, por lo que los algoritmos diseñados tendrán que ser capaces de escalar con los datos.

Como primera aproximación a los objetivos de la Tesis, en el Capítulo 2 se presenta un algoritmo de aprendizaje de controladores borrosos para robótica móvil con el preprocesamiento de variables de entrada embebido dentro del algoritmo de aprendizaje. Específicamente, se utilizan reglas borrosas cuantificadas para transformar las variables de bajo nivel a variables de alto nivel, reduciendo la dimensionalidad del problema a través de resumir los datos. Además, se propone un nuevo algoritmo, llamado *Iterative Quantified Fuzzy Rule Learning* (IQFRL, Aprendizaje Iterativo de Reglas Borrosas Cuantificadas, en castellano), basado en el *Iterative Rule Learning* y en programación genética para poder representar las estructuras de las reglas con una gramática. IQFRL también utiliza etiquetas lingüísticas definidas a través de granularidad múltiple sin restricciones, esto es, sin limitar los niveles de granularidad posibles.

A continuación, en el Capítulo 3 se presenta FRULER (*Fuzzy RUle Learning through Evolution for Regression*, Aprendizaje Evolutivo de Reglas Borrosas para Regresión en cas-

tellano), un nuevo GFS para la obtención de modelos lingüísticos de reglas borrosas TSK-1 simples y precisos capaces de resolver problemas de regresión. La baja complejidad de los sitemas borrosos tiene como objetivo mejorar tanto la capacidad de generalización como la legibilidad del modelo. Para ello, FRULER genera particiones borrosas lingüísticas con pocas etiquetas, un número bajo de reglas y regulariza los consecuentes — lo que reduce el número de variables de entrada utilizadas en la obtención de la salida. El algoritmo consiste en tres etapas: selección de instancias, discretización borrosas multi-granular y el aprendizaje evolutivo de bases de reglas, que utiliza regularización Elastic Net para obtener los consecuentes de las reglas.

El capítulo 4 se centra en la escabilidad de FRULER, obteniendo modelos con características similares — precisos y simples —, pero disminuyendo el tiempo de ejecución de convergencia del algoritmo. Para ello, el capítulo describe S-FRULER, una versión distribuída y escalable de FRULER, que divide el problema en particiones más pequeñas e incorpora un proceso de selección de características para reducir el número de variables utilizado en cada partición. Después, cada partición se resuelva independientemente usando el algoritmo FRULER, para, a continuación, utilizar una función de agregación que obtenga la base de reglas lingüísticas TSK final a partir de las bases de reglas generadas en cada partición.

Finalmente, el Capítulo 5 contiene las conclusiones de esta Tesis y sugiere posibles líneas de trabajo futuro.

Contents

CHAPTER 1

INTRODUCTION

1.1 Motivation

With the vast amounts of data that are being generated, there is a need of automatic processes that extract relevant patterns and trends from them, that allow users to understand what data says. This task is referred to as learning from data, being machine learning the field of computer science that focuses on the development of algorithms that automatically build a model for making data-driven predictions or decisions [15]. There are mainly two categories of learning problems:

- Supervised learning: the objective is to create a model that predicts the value of an output variable using a set of input measurements.
- Unsupervised learning: there is no output variable and the objective is to learn how data is organized by itself.

Inside supervised learning, depending on the type of output, there are two different problems: classification and regression. In classification, data is divided into several categories or classes, and the learning process must produce a model that assigns unseen inputs to their corresponding class. On the other hand, in a regression problem the output variable is a continuous real value rather than a discrete one, thus the obtained model generates a real-valued output from the input data.

The models obtained through machine learning techniques usually have two complementary requirements [56]:

- Accuracy or precision: indicates the ability of the model to predict values close to the

real ones.

- Interpretability: the capability of the model to be understood by a human being [7], which is related with how complex is the model.

The objective is to find a good balance between the complexity of the model and the reduction in the training error in order to, finally, get a good test error [47]. The training error tends to decrease as the model complexity increases, that is, when the model fits the data harder. However, when the complexity of the model exceeds a threshold, although the training error decreases, the model adapts itself too closely to the training data and fails in test. In contrast, if the model is not complex enough, it cannot represent the intrinsic knowledge of the data, resulting in poor precision and generalization. Figure 1.1 shows the typical behavior of the test and training errors, as model complexity is varied. With a low complexity both training and test errors remain high (underfitting). Then, as the complexity increases both errors decrease until some threshold where the test error worsens while the training error continues to improve (overfitting).

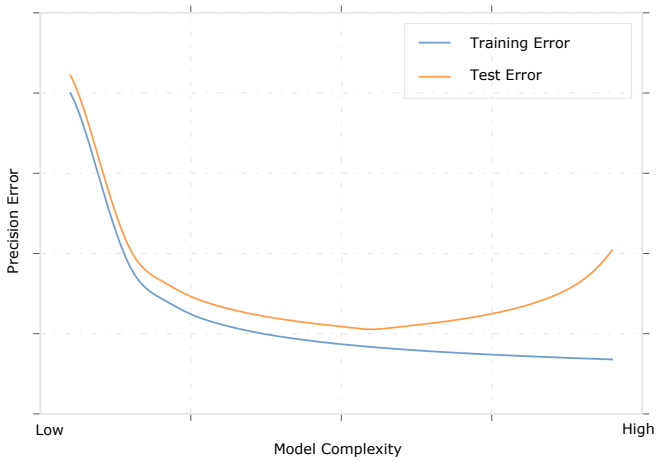


Figure 1.1: Test and training errors as a function of model complexity.

Within all type of models in machine learning, the most human interpretable ones are the decision trees and their equivalent as conditional statements or rules [47]. Both of them represent knowledge by means of conditions that, when the input data meet them, allow the output to be estimated. Moreover, the reasoning made by humans is imprecise by nature

and, in many real problems, the data has uncertainty in some degree. Fuzzy Logic [116] is a technique inside the field of soft computing that copes with this kind of uncertainty and imprecision, providing a framework where the truth values are real number between 0 and 1. Because of this, the use of fuzzy rules is much extended, since it combines the interpretability and expressiveness of the rules with the ability of fuzzy logic for representing uncertainty.

The automatic learning of Fuzzy Rule-Based Systems (FRBSs) has been approached in the literature using different Machine Learning techniques, for example with Neural Networks [57]. However, the most successful approach is the use of Evolutionary Algorithms, particularly Genetic Algorithms [61, 109]. These techniques have several characteristics that make them suitable for learning fuzzy rules. In particular, the flexibility of evolutionary algorithms allows to codify any part of the FRBS and, also, to manage the balance between accuracy and interpretability of the model in an effective way. The combination of these approaches led to a field inside soft computing called Genetic Fuzzy Systems (GFSs) [26].

One of the main drawbacks of GFSs is their computational cost. In general, the adaptation of Machine Learning approaches to large scale datasets has been a huge challenge addressed in the last years [43]. In a GFS approach, the size of the problem has a high influence in the performance of the obtained models [26, 49]. The learned fuzzy rule bases suffer from exponential rule explosion when the number of examples or variables increases. Thus, with huge search spaces, the convergence time for learning interpretable models rises in an unsustainable way. Moreover, evolutionary algorithms are computationally expensive by themselves due to the large number of evaluations needed to reach convergence and, in many cases, the evaluation process to obtain the fitness may take a long time.

The objective of this PhD Thesis is to design GFSs that learn FRBSs to solve regression problems in a general manner. Particularly, the aim is to obtain models with low complexity while maintaining high precision without using expert-knowledge about the problem to be solved. This means that the GFSs have to work with raw data, that is, without any preprocessing that help the learning process to solve a particular problem. This is of particular interest, when no knowledge about the input data is available or for a first approximation to the problem. Moreover, within this objective, GFSs have to cope with large scale problems, thus the algorithms have to scale with the data.

1.2 Fuzzy Rule-Based Systems

The input data of many real problems is uncertain and imprecise in some degree. The measurements can deviate from the actual value due to noise in the source or precision errors in the sensors. Moreover, the reasoning made by humans is imprecise by nature, without a clear boundary between qualitative or quantitative terms. Fuzzy logic is a technique inside the field of soft computing that copes with this kind of uncertainty [116]. Fuzzy logic provides a framework where the degree of truth of an expression is a real number between 0 and 1 —in classical logic it takes a value of true or false. Moreover, fuzzy sets are a generalization of the classical sets, and their elements have degrees of membership.

One of the most popular fuzzy logic based models in the literature are Fuzzy Rule-Based Systems (FRBS), where fuzzy propositions are stated in the antecedent and consequent part of the rules [114]. Thus, when dealing with multiple inputs-single output systems, these fuzzy rules are as follows:

$$\text{If } X_1 \text{ is } A_1 \text{ and } X_2 \text{ is } A_2 \text{ and } \ldots \text{ and } X_p \text{ is } A_p \text{ then } Y \text{ is } B \tag{1.1}$$

where X_j are the input variables, Y the output variable, A_j and B are the fuzzy sets associated with the input and output variables respectively, and p is the number of input variables. This type of knowledge representation models the interactions and relationships that exist between the input variables and the output. Moreover, the inference method of a FRBS is robust and flexible due to approximate reasoning.

The first type of FRBS was proposed by Mamdani [73] (Eq. 1.1). A Mamdani system is composed by four different parts (Fig. 1.2): the Knowledge Base, the Fuzzification Interface, the Inference System and the Defuzzification Interface. The Knowledge Base (KB) stores the model which is composed by the Data Base (DB) — the definition of the fuzzy sets used in the system — and the Rule Base (RB) — which are the conditional statements that use the DB information in their propositions.

The input values enter the system through a fuzzification interface, where the crisp input values are mapped into the fuzzy sets defined in the DB. Then, the Inference System calculates the fulfillment of the antecedent part of the rules and fires the corresponding consequents to calculate the estimated output. The inference of a particular fuzzy rule (Eq. 1.1) can be expressed as follows[73]:

$$\mu_B(y) = I(T(\mu_{A_1}(x_1), \mu_{A_2}(x_2), \ldots, \mu_{A_p}(x_p)), \mu_B(y)) \tag{1.2}$$

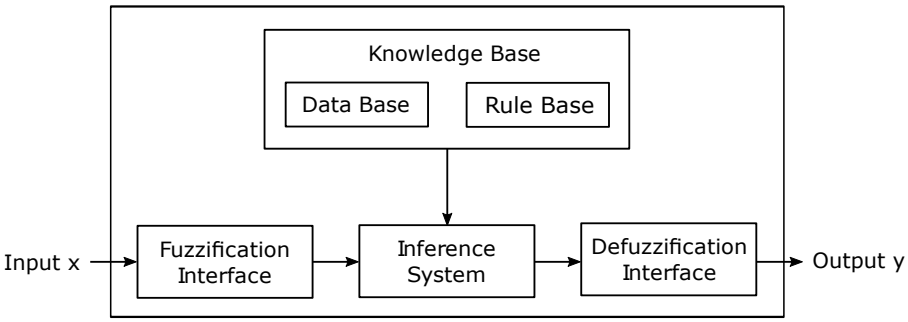


Figure 1.2: Structure of a Fuzzy Rule-Based System.

where μ_F is the membership function of the fuzzy set F, $x_1, x_2, \ldots, x_p$ are the values of each input variable, T is a fuzzy conjunctive operator and I is a fuzzy implication operator. The most common choice for both fuzzy operators is the minimum t-norm. Finally, the fuzzy outputs of the RB are transformed into a crisp value that is the final output of the FRBS. In regression problems, the defuzzification interface is usually implemented with the center of gravity of the union of all the output fuzzy sets for all rules.

The most important aspect of a fuzzy system in terms of interpretability is the definition of the fuzzy sets used in the DB. Two different approaches can be followed:

- The linguistic approach defines globally the division of each variable into fuzzy sets, called fuzzy partitions. The fuzzy sets inside a fuzzy partition can be associated to linguistic labels, thus giving more human interpretability to the system. However, this limits the degrees of freedom of the system, and, therefore, the ability to accurately approximate any problem.
- The approximative approach uses a different fuzzy set definition for each proposition of each rule. Therefore, there is no global partition of the variables into fuzzy sets, and there are no linguistic labels associated to them. Generally, this approach is used in problems where the system has to be really accurate. However, approximative approaches can lead to complex partitions of the input space that can make difficult to understand how the input is associated with the response.

Takagi, Sugeno, and Kang proposed in [108, 106] a fuzzy rule model, called TSK fuzzy rules, in which the antecedents are comprised of linguistic variables, as in the case of Mamdani [73, 74], but the consequent is represented as a polynomial function of the input variables.

This type of rules fuse the interpretability of fuzzy rules with the accuracy of regression, thus improving the precision of the Mamdani type systems. Although Mamdani systems are well-known for their semantic interpretability, the model of TSK rules is also a good choice since it is straightforward to understand the relationship between the output and input variables. This is of particular interest in many fields, such as robotics [93, 90, 85], medical imaging [91], industrial estimation [86] and optimization of processes [112].

The most common function for the consequent of a TSK rule is a linear combination of the input variables (TSK-1), and its structure is as follows:

$$\begin{aligned} &\text{If } X_1 \text{ is } A_1 \text{ and } X_2 \text{ is } A_2 \text{ and } \ldots \text{ and } X_p \text{ is } A_p \text{ then} \\ &Y = \beta_0 + X_1 \cdot \beta_1 + X_2 \cdot \beta_2 + \cdots + X_p \cdot \beta_p \end{aligned} \tag{1.3}$$

where β_j is the coefficient associated with X_j in the consequent part of the rule. The matching degree h_k between the antecedent of the rule r_k and the inputs to the system $(x_1, x_2, \ldots, x_p)$ is calculated as:

$$h_k = T(A_1^k(x_1), A_2^k(x_2), \ldots, A_p^k(x_p)) \tag{1.4}$$

where A_j^k is the linguistic fuzzy term for the j-th input variable in the k-th rule, and T is a t-norm conjunctive operator, usually the minimum function. The final output of a TSK fuzzy rule base system composed of m TSK fuzzy rules is computed as the average of the individual rule outputs Y_k weighted by the matching degree:

$$\hat{y} = \frac{\sum_{k=1}^{m} h_k \cdot Y_k}{\sum_{k=1}^{m} h_k} \tag{1.5}$$

Linguistic TSK fuzzy rule systems represent a good trade-off between accuracy and interpretability:

- The use of linguistic terms in the antecedent of the rules provides a full description of the input space due to the shared definition of the fuzzy partitions in the data base.
- The linear representation of the output allows to obtain accurate solutions using different well-studied statistical methods.
- The consequent of the rules represented by a linear combination of the input variables allows an easy understanding of the relationship between the inputs and the output.

Thus, even if the TSK fuzzy rule systems are less comprehensible in natural language terms than a Mamdani approach, they can provide useful and understandable information, and is the preferable choice in some domains.

1.3 Genetic Fuzzy Systems

The automatic learning of FRBSs has been tackled with different Machine Learning techniques, for example with Neural Networks [57]. However, the most successful technique are Evolutionary Algorithms, particularly Genetic Algorithms (GA)[61, 109]. Evolutionary algorithms have some features that make them suitable for learning fuzzy rules. The use of GA to learn FRBSs is very extended and gave birth to a novel field called Genetic Fuzzy Systems (GFS) [26].

A GA is a heuristic search technique that mimics the process of natural selection, using operations like crossover, mutation and selection to generate a set of possible solutions which are optimized through several iterations until a convergence criteria is reached. The most important characteristic of a GA is its capability to explore large search spaces — very useful in real-value optimizations — and its flexibility to incorporate prior knowledge in any part of the algorithm: the representation of the solutions to match the type of model to learn, the genetic operators to force the convergence of the algorithm towards promising solutions, etc. This flexibility allows to codify any part of the FRBS into the GA. Because of this, there are two different tasks that can be achieved[49]:

- In a learning process, the objective is to obtain a new FRBS model from the data. This approach learns the components of the KB from scratch, using only the information of the training data provided to the process.
- A tuning process starts from an existing KB and the GA optimizes some of the FRBS parameters: rule selection, rule weighting, membership functions optimization, etc.

There are different GFS approaches, depending on how the RB is represented [26]:

- Michigan: each individual solution represents a rule and the whole population represents the RB, evolving simultaneously. This approach has as drawback the lack of cooperation between solutions and that several rules may be competing to represent the same knowledge.
- Pittsburgh: this approach tries to solve the cooperative-competitive problem encoding the full KB into a single individual solution. However, this may generate overly complex chromosomes, and, therefore, the definition of the evolutionary operations becomes more difficult. Recently [2], this problem was solved representing only the DB and generating the RB with an ad-hoc method.

- Iterative Rule Learning (IRL): in this approach, each individual represents a rule like in a Michigan approach, but instead of generating the RB using the whole population, it only obtains one rule at the end of the evolutionary process. Then, the algorithm is repeated removing the training examples already covered by the previous generated rules, until no examples remain. This approach usually generates too many rules that cover a small search space, therefore is usually followed by a rule selection process.

In particular, the flexibility of evolutionary algorithms allows to manage the balance between accuracy and interpretability of the model in an effective way. The interpretability of a FRBS involves two main issues [7]:

- Readability: it is related with the simplicity of the fuzzy system structure, i.e., the number of variables, linguistic terms per variable, fuzzy rules, antecedents per rule, etc. It represents the quantitative or objective part of the interpretability of the model.
- Comprehensibility: it is determined by the general semantics of the fuzzy system and the fuzzy inference mechanism. It is associated with the fuzzy partitioning of the variables and its meaning for the user, thus representing the qualitative or subjective part of the interpretability.

Readability is usually transformed into a measurement that can be incorporated to the fitness function inside the GFS. However, comprehensibility is more subtle, and only some characteristics like strong fuzzy partitions [96] or a low number of fuzzy sets for each fuzzy partition can improve this measure.

The simplicity of the models obtained by regression GFSs has been mostly achieved in the literature through the control of the number of rules and/or the number of labels used in the rule base through a multi-objective approach [33, 54]. More recently, the use of instance selection techniques has received more attention in both classification [34, 41] and regression [92] problems. This approach faces two problems at the same time: decreases the complexity for large-scale problems and reduces the overfitting, as the rules can be generated with a part of the training data and the error of the rule base can be estimated with another part (or the whole training set).

The most important aspect of a fuzzy system in terms of interpretability is the definition of the fuzzy partition for each variable in the DB. One of the most used techniques is the multi-granularity approach. The universe of discourse of a variable is divided into a different number labels for each granularity, thus obtaining a different fuzzy partition depending on the number of labels needed. Moreover, when no expert knowledge is available to determine the

fuzzy labels, two different approaches can be applied: uniform discretization combined with lateral displacements [2], or non-uniform discretization [55]. Recently, [32, 42] have shown the application of non-uniform discretization techniques to classification problems.

The use of TSK fuzzy rule bases implies another complexity dimension: the polynomial in the consequent —usually with degree 1 (TSK-1) or 0 (TSK-0). The most widely used approach for learning the coefficients of the polynomial is the least squares method. However, that choice often leads to models that overfit the training data and misbehave in test. This problem can be solved by shrinking some coefficients (Ridge regularization) or setting them to zero (Lasso regularization), thus obtaining less complex models. Moreover, a combination of both regularizations (called Elastic Net [119]) can be used.

1.4 Scalability of Genetic Fuzzy Systems

In a GFS approach, the size of the problem has a huge influence in the performance of the obtained models [26, 49]. The fuzzy rule bases suffer from exponential rule explosion when the number of examples or variables increases. Thus, with huge search spaces, the convergence time rises exponentially. In particular, recent developments using multi-objective evolutionary fuzzy systems can be found in [3, 8, 39, 95, 94, 98], where both Mamdani and TSK systems were proposed to solve large-scale regression problems. Moreover, in [76] an adaptive fuzzy inference system was proposed to cope with high-dimensional problems. However, the number of variables and/or number of examples in the datasets used in these papers are still not high enough for properly labelling them as Big Data. Thus, there is also a need in the field to use Big Data regression datasets to benchmark the performance of the proposed GFSs.

Evolutionary algorithms are computationally expensive by themselves due to the large number of evaluations needed to reach convergence and, in many cases, the evaluation process to obtain the fitness may take a long time. The techniques to improve the scalability of GFS can be classified into three categories [36]:

- Algorithm-oriented, that adapt the structure of the evolutionary algorithm.
- Data-oriented, that modify the training data to reduce the computational cost of the learning process
- Distributed approaches, that take advantage of the availability of several machines to reduce the runtime.

To scale the learning process of a GFS in an algorithm-oriented manner, several papers in the literature focused in the control of the search space, by reducing the number of rules and/or the number of labels used in the rule base through a multi-objective approach [2, 3, 8, 33]. For example, rule explosion can be controlled by limiting the number of labels considered in the learning process [2, 95, 94].

Moreover, in recent years, the data-oriented approach has received increasing attention. The use of instance selection techniques decreases the complexity of large scale problems and reduces overfitting. For example, in [2] the error was estimated using a randomly selected subset of the examples, and the complete training dataset was only used for the most promising individuals. Also, in [92], a new instance selection method for regression was applied to generate the rules, while the error of the rule base was estimated with the whole training dataset.

From a Big Data point of view, the distributed approach is the most appropriate for scaling GFS. Among the most frequently used frameworks in Big Data analytics [88], the most popular ones are: i) MPI (Message Passing Interface) which efficiently exploits multi-core clusters architectures, and ii) Apache Spark [118], a recently developed platform that can be executed in traditional clusters such as Hadoop [115]. Spark was designed to perform distributed processing and other workloads like streaming, interactive queries, and machine learning focused algorithms. While MPI provides a solution mostly oriented to high performance computing, Spark also deals with failures and straggler nodes effectively but with an impact on speed. From the perspective of GFS, only a few works use Big Data frameworks to solve the scaling problem [36].

The extended use of Spark is closely linked to the success of Hadoop, which processes vast amounts of data in parallel on large clusters, usually implemented using the Hadoop Distributed File System. Hadoop popularized the MapReduce[28] approach, a distributed methodology based on the definition of two different functions: a Map function that distributes a block of data to several Worker Nodes, and executes the same process — called Task — in each data partition; and the Reduce function that aggregates the results of the Map functions by means of a Key-Value representation of the results, and performs some operations to obtain the final result. Spark adds to this framework the capability to use other data-flows with an improvement of in-memory computing and an easy of programming high-level functions that facilitate to build parallel applications.

1.5 Objectives

The objective of this PhD Thesis is to design GFSs that learn FRBSs to solve regression problems. Particularly, the aim is to obtain models with low complexity while maintaining high precision without using expert-knowledge about the problem to be solved. To achieve this, the following objectives have been pursued:

(a) Learning fuzzy controllers in mobile robotics with embedded preprocessing

As a first approximation to the objective of this PhD Thesis, the goal was to design a GFS that obtains a fuzzy controller for a well-studied behavior in mobile robotics. In this particular case, the aim was to automatically learn the fuzzy controller associated with the wall-following behavior of an autonomous mobile robot. The first step for designing controllers for mobile robots consists of the preprocessing of the raw sensor data, which usually is of high dimensionality. For example, a robot equipped with two laser range finders with 0.5 degrees of precision can provide 720 low-level distance values that must be aggregated into more significant variables, such as frontal distance. For this problem, no knowledge about how to transform the raw primitive sensor data into high-level input variables was considered in the learning process. Thus, the designed GFS should automatically perform the mapping between low-level and high-level input variables during the learning phase of the controller. The model learned should provide propositions that are able to summarize the data and model the underlying knowledge in a better way than just using average, maximum or minimum values. Therefore, in this work the FRBSs use a particular type of fuzzy proposition called Quantified Fuzzy Propositions (QFP), which provide a formal model that is capable to represent the knowledge involved in this grouping task.

(b) TSK Fuzzy Rule Evolutionary Learning for Regression

Starting from the development made in the previous work, the next step was to design a GFS that learns accurate and low complex FRBSs to solve regression problems. Since the objective is a general purpose algorithm, the approach cannot take into account any expert knowledge about a particular problem, and only used the example data provided to the system with no other information. Particularly, to meet this goal, the GFS should learn linguistic TSK KBs which represent a good trade-off between accuracy and interpretability. Also, the following issues should be considered:

- There is no previous definition of the fuzzy terms of the DB, hence the fuzzy partitions must be obtained automatically. For that, an automatic fuzzy discretization technique should be developed. Moreover, a multi-granularity approach is preferred to give the GFS the flexibility to choose a different granularity level for each input variable.
- In order to prevent overfitting, it is necessary to obtain low complex models which generalize well both the antecedent part — low granularity for each variable — and the consequent part — regularized regression.

(c) Scalability of TSK Fuzzy Rule Evolutionary Learning for Regression

The last objective was to adapt the previous algorithm to cope with large scale problems, thus giving the algorithm the capability to scale with the data size. This is of particular interest in the case of GFS, due to the rule explosion when the number of input variables increases. Moreover, the evolutionary algorithms are computationally expensive by themselves, because of the high number of iterations needed to converge and because the fitness function can take a long time to be computed. The proposal has to be able to deal with problems with a large number of examples and high dimensional inputs, while maintaining the ability to learn simple an precise TSK FRBSs.

1.6 Contributions

The main contributions of this PhD dissertation are:

- Iterative Quantified Fuzzy Rule Learning (IQFRL), an algorithm that is able to learn QFRs of variable structure for the design of controllers with embedded preprocessing in mobile robotics. It is based on the Iterative Rule Learning (IRL) approach and uses linguistic labels defined with unconstrained multiple granularity, i.e., without limiting the granularity levels. This proposal was designed to solve control (regression) problems in mobile robotics having as input variables the internal state of the robot and the sensors distance data. Expert knowledge is only used to generate the training data, and also to define the context-free grammar that specifies the structure of the rules.
  - The proposed algorithm is able to learn with the state of the robot and the raw sensors data, with no preprocessing. The mapping between low-level variables and high-level variables is done embedded in the algorithm.

  - The algorithm uses Quantified Fuzzy Propositions, a model able to summarize the low-level input data [117].
  - IQFRL uses linguistic labels with unconstrained multiple granularity. With this approach, the interpretability of the membership functions is unaffected while the flexibility of representation remains.
  - The proposal was validated in several simulated and real environments with the wall-following behavior. The approach was also tested in three real world behaviors: path tracking with obstacles avoidance, object tracking with fixed obstacles avoidance, and object tracking with moving obstacle avoidance.

- Fuzzy RUle Learning through Evolution for Regression (FRULER), a new GFS algorithm for obtaining accurate and simple linguistic TSK-1 fuzzy rule base models to solve regression problems. The simplicity of the fuzzy system aims to improve both the generalization ability and the readability of the model —and, therefore, the interpretability— by obtaining linguistic fuzzy partitions with few labels, a low number of rules, and the regularization of the consequents —which reduces the number of input variables that contribute to the output. It is made up by three components: a two-stage preprocessing —formed by the instance selection and multi-granularity fuzzy discretization modules—, and a genetic algorithm, which contains an ad-hoc TSK-1 rule generation module. Both preprocessing techniques are executed to improve the simplicity of the fuzzy rule bases generated by the evolutionary algorithm, while the evolutionary learning process obtains a definition of the data base of the knowledge system.
  - The algorithm incorporates a new instance selection method for regression, called Class Conditional Instance Selection for Regression (CCISR), that has a good balance between reduction and accuracy, with a low computational cost. CCISR uses the class conditional nearest neighbor relation over pairs of points in the training set, and uses this information to develop an effective large margin instance selection method. The instance selection process reduces the variance of the models focusing the generated rules on the representative examples.
  - The fuzzy partitions of the DB are obtained using a novel multi-granularity fuzzy discretization of the input variables, in order to generate non-uniform fuzzy partitions with different degrees of granularity. This process decreases the complexity

of the fuzzy partitions and, therefore, it is not necessary to establish an upper bound in the number of rules in the evolutionary stage.

  - The evolutionary algorithm uses a fast and scalable method with Elastic Net regularization solved through Stochastic Gradient Descent to generate accurate and simple TSK-1 fuzzy rules.
  - The automatic generation of the RB uses only the instances selected by CCISR, while the fitness function is calculated with all the examples in the training dataset. Thus, this evaluation can be seen, in some way, as a validation process, as the rule base was constructed with a subset of the examples.
  - Each stage was validated using 28 real-world datasets. Moreover, FRULER was-compared with three state of the art GFSs for regression.

- S-FRULER, a scalable version of FRULER, which allows to obtain models with similar characteristics than those obtained by FRULER —accurate and simple—, but reducing the runtime of the algorithm to converge. The algorithm divides the problem into a set of smaller problems that are solved independently using FRULER. Then, the solutions obtained in each partition are combined in order to get a final solution.
  - The developed GFS uses a distributed approach and the Spark software.
  - S-FRULER incorporates a random feature selection process to reduce the number of variables used in each partition of the problem.
  - The number of partitions was automatically calculated taking into account quantifiable characteristics of the dataset.
  - After obtaining the solutions for each partition, these are combined completing the variables not used in one partition with information of the others. Then, these solutions are evaluated using the combination of selected instances in each partition to generate the new KB. The solution with best performance using the full training dataset is selected as the final solution.
  - S-FRULER was validated in terms of scalability, precision and complexity using 10 large-scale datasets and it was compared with three state of the art GFSs.
  - Moreover, in order to validate the scalability of the algorithm, a real bioinformatic problem [13] that combines a high number of examples and high dimensional

inputs in a regression problem was confronted, obtaining good results in both accuracy and complexity.

In addition to the contributions to the GFS field, two web applications have been developed:

- TSK-View [22]: a web application that allows users to visualize regression models based on TSK rules, together with the underlying reasoning process. The software incorporates a visual analysis of:
  - the relative importance of the variables (numerical weights).
  - the fuzzy linguistic labels.
  - the rules of the knowledge-base.

  TSK-View users can analyze in the web their own TSK models, add new input values , and visualize the inference results.
- A web platform, called STAC (Stastical Tests for Algorithms Comparison) [19] that facilitates the application of statistical tests for the comparison of algorithms. Since the correct election of a test depends on a number of factors, such as the data distribution (parametrical or non-parametrical tests), the relationship among them (paired or unpaired) or the number of algorithms to be compared (pairwise or multiple), STAC includes an assistant for deciding among the existing alternatives.

The contributions of this PhD dissertation are included in the following publications:

Journal Papers

- I. Rodríguez-Fdez, M. Mucientes, and A. Bugarín. S-FRULER: Scalable Fuzzy Rule Learning through Evolution for Regression. *Knowledge-Based Systems*, Elsevier, Available online 26 July 2016. (DOI 10.1016/j.knosys.2016.07.034).
  Impact Factor (JCR 2015): 2.947
  Category: COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE. Rank: 16/123. Quartile 1.
- I. Rodríguez-Fdez, M. Mucientes, and A. Bugarín. FRULER: Fuzzy Rule Learning through Evolution for Regression. *Information Sciences*, Elsevier, No. 354, pp. 1-18, 2016. (ISSN 0020-0255, DOI 10.1016/j.ins.2016.03.012).
  Impact Factor (JCR 2015): 4.038

Category: COMPUTER SCIENCE, INFORMATION SYSTEMS. Rank 6/139. Decil 1. Quartile 1.

– I. Rodríguez-Fdez, M. Mucientes, and A. Bugarín. Learning Fuzzy Controllers in Mobile Robotics with Embedded Preprocessing. *Applied Soft Computing*, Elsevier, No. 26, pp. 123-142, 2015 (ISSN 1568-4946, DOI 10.1016/j.asoc.2014.09.021).
Impact Factor (JCR 2015): 2.810
Category: COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE. Rank: 17/123. Quartile 1.

Book Chapters

– I. Rodríguez-Fdez, M. Mucientes and A. Bugarín. Springer Handbook of Computational Intelligence, chapter Application of Fuzzy Techniques to Autonomous Robots. pages 313-328. Springer, 2015 (ISBN 978-3-662-43504-5, DOI 10.1007/978-3-662-43505-2).

Conference Papers

– I. Rodriguez-Fdez, M. Mucientes and A. Bugarín. A Genetic Fuzzy System for Large-scale Regression. *In Proceedings of the 25th IEEE International Conference on Fuzzy Systems*, Vancouver (Canada), 2016.
Conference Ranking (CORE 2014): A

– I. Rodríguez-Fdez, M. Mucientes, and A. Bugarín. Reducing the Complexity in Genetic Learning of Accurate Regression TSK Rule-Based Systems. *In Proceedings of the 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Istanbul (Turkey), 2015 (ISBN 978-1-4673-7428-6, DOI 10.1109/FUZZ-IEEE.2015.7337930).
Conference Ranking (CORE 2014): A
Best Student Paper Nomination Award (within the 4 finalists)

– I. Rodríguez-Fdez, M. Mucientes, and A. Bugarín. STAC: a web platform for the comparison of algorithms using statistical tests. *In Proceedings of the 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Istanbul (Turkey), 2015 (ISBN 978-1-4673-7428-6, DOI 10.1109/FUZZ-IEEE.2015.7337889).
Conference Ranking (CORE 2014): A

– I. Rodríguez-Fdez, M. Mucientes, and A. Bugarín. An Instance Selection Algo-

rithm for Regression and its Application in Variance Reduction. *In Proceedings of the 2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Hyderabad (India), 2013 (DOI 10.1109/FUZZ-IEEE.2013.6622486).
Conference Ranking (CORE 2014): A

- I. Rodríguez-Fdez, M. Mucientes, and A. Bugarín. Iterative rule learning of quantified fuzzy rules for control in mobile robotics. *In Proceedings of IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 111–118, Paris (France), 2011 (ISBN 978-1-61284-048-2, DOI 10.1109/GEFS.2011.5949500).
- M. Mucientes, I. Rodríguez-Fdez, and A. Bugarín. Evolutionary learning of quantified fuzzy rules for hierarchical grouping of laser sensor data in intelligent control. *In Proceedings of the IFSA-EUSFLAT 2009 conference*, pages 1559–1564, Lisbon (Portugal), 2009.
Conference Ranking (CORE 2008): B
- I. Rodríguez-Fdez, M. Mucientes, A. Bugarín. A MapReduce Implementation of a Genetic Fuzzy System for Regression. *In Actas del XVIII Congreso Español sobre Tecnologías y Lógica Fuzzy*, pp. 178-179. San Sebastián (España). 2016
- I. Rodríguez-Fdez, M. Mucientes, and A. Bugarín. Photons detection in positron emission tomography through iterative rule learning of tsk rules. *In Actas del VIII Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, pages 251-258, Albacete (Spain), 2012.

1.7 Dissertation Structure

This thesis consists of four further chapters. Chapters 2, 3 and 4 contain a full copy of each one of the main publications made during this PhD thesis, together with a brief introduction and contextualization of the papers.

Chapter 2 is focused on the learning of fuzzy controllers in mobile robotics with embedded preprocessing. Specifically, Quantified Fuzzy Rules (QFR) are used to transform low-level input variables into high-level input variables, reducing the input dimensionality through summarization. Moreover, the proposed algorithm, called Iterative Quantified Fuzzy Rule Learning (IQFRL), is based on the Iterative Rule Learning approach and on genetic programming in order to represent the valid rule structures with a grammar. IQFRL also uses

linguistic labels defined through unconstrained multiple granularity, i.e., without limiting the granularity levels.

Chapter 3 presents FRULER (Fuzzy RUle Learning through Evolution for Regression), a new GFS algorithm for obtaining accurate and simple linguistic TSK-1 fuzzy rule base models to solve regression problems. The simplicity of the fuzzy system aims to improve both the generalization ability and the readability of the model. For that, FRULER generates linguistic fuzzy partitions with few labels, a low number of rules, and regularizes the consequents — which reduces the number of input variables that contribute to the output. The algorithm consists of three stages: instance selection, multi-granularity fuzzy discretization of the input variables, and the evolutionary learning of the rule base that uses the Elastic Net regularization to obtain the consequents of the rules.

Chapter 4 focuses on the scalability of FRULER, obtaining models with similar characteristics —accurate and simple—, but reducing the runtime of the algorithm to converge. For that, this chapter describes S-FRULER, a scalable distributed version of FRULER, which focuses on splitting the problem into smaller partitions and incorporates a feature selection process for reducing the number of variables used in each partition. Each partition is then solved independently using the FRULER algorithm. Then, an aggregation function is used to obtain linguistic TSK fuzzy rule bases from the rule bases generated for each dataset partition.

Finally, Chapter 5 contains the conclusions of this PhD thesis and suggests future lines of work.

CHAPTER 2

LEARNING FUZZY CONTROLLERS IN MOBILE ROBOTICS WITH EMBEDDED PREPROCESSING

The first objective of the thesis was to develop an algorithm to learn fuzzy controllers for mobile robot behaviors. Usually, the automatic learning of controllers for mobile robots requires a first stage where sensorial data are preprocessed or transformed into high level variables which are usually defined from expert knowledge. One of the main challenges of the approach was to do the mapping between low-level and high-level input variables automatically during the learning phase of the controller.

This part of the thesis introduces the combination of evolutionary algorithms and FRBS. Specifically, in this work, Quantified Fuzzy Rules (QFR) are used to transform low-level input variables into high-level input variables, reducing the input dimensionality through summarization. Moreover, the proposed algorithm, called Iterative Quantified Fuzzy Rule Learning (IQFRL), is based on the Iterative Rule Learning approach and on genetic programming in order to represent the valid rule structures with a grammar. IQFRL also uses linguistic labels defined through unconstrained multiple granularity, i.e., without limiting the granularity levels.

The algorithm has been tested with the implementation of the wall-following behavior both in several realistic simulated environments with different complexity and on a *Pioneer 3-AT* robot in two real environments. Results have been compared with several state-of-the-

art learning approaches combined with different data preprocessing techniques, showing that IQFRL exhibits a better and statistically significant performance. Moreover, three real world applications for which IQFRL plays a central role are also presented: path and object tracking with static and moving obstacles avoidance.

In this chapter, a full copy of the following publication is presented:

> I. Rodríguez-Fdez[1], M. Mucientes[1], and A. Bugarín[1]. Learning Fuzzy Controllers in Mobile Robotics with Embedded Preprocessing. *Applied Soft Computing*, Elsevier, No. 26, pp. 123-142, 2015.

2.1 Abstract

The automatic design of controllers for mobile robots usually requires two stages. In the first stage, sensorial data are preprocessed or transformed into high level and meaningful values of variables which are usually defined from expert knowledge. In the second stage, a machine learning technique is applied to obtain a controller that maps these high level variables to the control commands that are actually sent to the robot. This paper describes an algorithm that is able to embed the preprocessing stage into the learning stage in order to get controllers directly starting from sensorial raw data with no expert knowledge involved. Due to the high dimensionality of the sensorial data, this approach uses Quantified Fuzzy Rules (QFRs), that are able to transform low-level input variables into high-level input variables, reducing the dimensionality through summarization. The proposed learning algorithm, called Iterative Quantified Fuzzy Rule Learning (IQFRL), is based on genetic programming. IQFRL is able to learn rules with different structures, and can manage linguistic variables with multiple granularities. The algorithm has been tested with the implementation of the wall-following behavior both in several realistic simulated environments with different complexity and on a *Pioneer 3-AT* robot in two real environments. Results have been compared with several well-known learning algorithms combined with different data preprocessing techniques, showing that IQFRL exhibits a better and statistically significant performance. Moreover, three real world applications for which IQFRL plays a central role are also presented: path and object tracking with static and moving obstacles avoidance.

[1]Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain.

2.2 Introduction

The control architecture of mobile robots usually includes a number of behaviors that are implemented as controllers, which are able to solve specific tasks such as motion planning, following a moving object, wall-following, avoiding collisions, etc. in real time. These behaviors are implemented as controllers whose outputs at each time point (control commands) depend on both the internal state of the robot and the environment in which it evolves. The robot sensors (e.g. laser range finders, sonars, cameras, etc.) are used in order to obtain the augmented state of the robot (internal state and environment). When the robot operates in real environments, both the data obtained by these sensors and the internal state of the robot present uncertainty or noise. Therefore, the use of mechanisms that manage them properly is necessary. The use of fuzzy rules is convenient to cope with this uncertainty, since it combines the interpretability and expressiveness of the rules with the ability of fuzzy logic for representing uncertainty.

The first step for designing controllers for mobile robots consists of the preprocessing of the raw sensor data: the low-level input variables obtained by the sensors are transformed into high-level variables that are significant for the behavior to be learned. Usually, expert knowledge is used for the definition of these high-level variables and the mapping from the sensorial data. After this preprocessing stage, machine learning algorithms can be used to automatically obtain the mapping from the high-level input variables to the robot control commands. This paper describes an algorithm that is able to perform the preprocessing stage embedded in the learning stage, thus avoiding the use of expert knowledge. Therefore, the mapping between low-level and high-level input variables is done automatically during the learning phase of the controller.

The data provided by the sensors is of high dimensionality. For example, a robot equipped with two laser range finders can generate over 720 low-level variables. However, in mobile robotics it is more common to work with sets or groupings of these variables, (e.g. “frontal sector”) that are much more significant and relevant for the behavior. As a result, it is necessary to use a model that is capable of grouping low-level variables, thus reducing the dimensionality of the problem and providing meaningful descriptions. The model should provide propositions that are able to summarize the data with expressions like “part of the distances in the frontal sector are high”. This kind of expressions can model the underlying knowledge in a better way than just using average, maximum or minimum values of sets of low level variables. Moreover, these expressions also include the definition of the set of low-level variables

to be used. Since these propositions involve fuzzy quantifiers (e.g. "part"), they are called Quantified Fuzzy Propositions (QFPs) [81]. QFP provide a formal model that is capable of modeling the knowledge involved in this grouping task.

Evolutionary algorithms have some characteristics that make them suitable for learning fuzzy rules. The well-known combination of evolutionary algorithms and fuzzy logic (genetic fuzzy systems) is one of the approaches that aims to manage the balance between accuracy and interpretability of the rules [23, 49]. As it was pointed out before, fuzzy rules can be composed of both conventional and QFPs (therefore, they will be referred to as QFRs). Furthermore, the transformation from low-level to high-level variables using QFPs produces a variable number of propositions in the antecedent of the rules. Therefore, genetic programming, where the structure of individuals is a tree of variable size derived from a context-free grammar, is here the most appropriate choice.

This paper describes an algorithm that is able to learn QFRs of variable structure for the design of controllers with embedded preprocessing in mobile robotics. This proposal, called Iterative Quantified Fuzzy Rule Learning (IQFRL), is based on the Iterative Rule Learning (IRL) approach and uses linguistic labels defined with unconstrained multiple granularity, i.e. without limiting the granularity levels. This proposal has been designed to solve control (regression) problems in mobile robotics having as input variables the internal state of the robot and the sensors data. Expert knowledge is only used to generate the training data for each of the situations of the task to be learned and, also, to define the context-free grammar that specifies the structure of the rules.

The main contributions of the paper are: (i) the proposed algorithm is able to learn using the state of the robot and the sensors data, with no preprocessing. Instead, the mapping between low-level variables and high-level variables is done embedded in the algorithm; (ii) the algorithm uses QFPs, a model able to summarize the low-level input data; (iii) moreover, IQFRL uses linguistic labels with unconstrained multiple granularity. With this approach, the interpretability of the membership functions used in the resulting rules is unaffected while the flexibility of representation remains. The proposal was validated in several simulated and real environments with the wall-following behavior. Results show a better and statistically significant performance of IQFRL over several combinations of well-known learning algorithms and preprocessing techniques. The approach was also tested in three real world behaviors that were built as a combination of controllers: path tracking with obstacles avoidance, object tracking with fixed obstacles avoidance, and object tracking with moving obstacle avoidance.

The paper is structured as follows: Section 2.3 summarizes recent work related with this proposal and Section 2.4 presents the QFRs model and its advantages in mobile robotics. Section 2.5 describes the IQFRL algorithm that has been used to learn the QFRs. Section 2.6 presents the obtained results, and Section 2.7 shows three real world applications of IQFRL in robotics. Finally, Section 2.8 points out the most relevant conclusions.

2.3 Related Work

The learning of controllers for autonomous robots has been dealt with by using different machine learning techniques. Among the most popular approaches can be found evolutionary algorithms [16, 77], neural networks [107] and reinforcement learning [1, 68]. Also hibridations of them, like evolutionary neural networks [65], reinforcement learning with evolutionary algorithms [97, 72], the widely used genetic fuzzy systems [102, 83, 79, 80, 67, 62, 84], or even more uncommon combinations like ant colony optimization with reinforcement learning [59] or differential evolution [53] or evolutionary group based particle swarm optimization [58] have been successfully applied. Furthermore, over the last few years, mobile robotic controllers have been getting some attention as a test case for the automatic design of type-2 fuzzy logic controllers [68, 77, 53].

An extensive use of expert knowledge is made in all of these approaches. In [102] 360 laser sensor beams are used as input data, and are heuristically combined into 8 sectors as inputs to the learning algorithm. On the other hand, in [65, 83, 79, 80, 67, 84, 59, 58] the input variables of the learning algorithm are defined by an expert. Moreover, in [83, 79, 67, 84, 53] the evaluation function of the evolutionary algorithm must be defined by an expert for each particular behavior. As in the latter case, the reinforcement learning approaches need the definition of an appropriate reward function using expert knowledge.

The approaches based on genetic fuzzy systems use different alternatives in the definition of the membership functions. In [97, 102, 67] the membership functions are defined heuristically. In [79, 80] labels have been uniformly distributed, but the granularity of each input variable is defined using expert knowledge. On the other hand, in [83, 62, 84, 59, 58] an approximative approach is used, i.e., different membership functions are learned for each rule, reducing the interpretability of the learned controller.

The main problem of learning behaviors using raw sensor input data is the curse of dimensionality. In [1], this issue has been managed from the reinforcement learning perspective, by

using a probability density estimation of the joint space of states. Among all the approaches based on evolutionary algorithms, only in [16] no expert knowledge has been taken into account. In this work, the number of sensors and their position are learned from a reduced number of sensors.

In [85] a Genetic Cooperative-Competitive Learning (GCCL) approach was presented. The proposal learns knowledge bases without preprocessing raw data, but the rules involved approximative labels while the IQFRL proposal uses unconstrained multiple granularity. Moreover, in this approach it is difficult to adjust the balance between cooperation and competition, which is typical when learning rules in GCCL. As a result, the obtained rules where quite specific and the performance of the behavior was not comparable to other proposals based on expert knowledge.

2.4 Quantified Fuzzy Rules (QFRs)

2.4.1 QFRs for robotics

Machine learning techniques in mobile robotics are used to obtain the mapping from inputs to outputs (control commands). In general, two categories can be established for the input variables:

- High-level input variables: variables that provide, by themselves, information that is relevant and meaningful to the expert for modeling the system (e.g. the linear velocity of the robot, or the right-hand distance from the robot to a wall).
- Low-level input variables: variables that do not provide by themselves information for the expert to model the system (e.g. a single distance measure provided by a sensor). Relevance of these variables emerge when they are grouped into more significant sets of variables. For example, the control actions cannot be decided by simply analyzing the individual distance values provided by each beam of a laser range finder, since noisy measurements or gaps between objects (very frequent in cluttered environments) may occur. Instead, more significant variables and models involving complex groupings and structures are used.

Usually, high-level variables, or sectors, consisting of a set of laser beam measures instead of the beam measures themselves (e.g., right distance, frontal distance, etc.) are used in mobile robotics. The low-level input variables are transformed into high-level input variables in

a preprocessing stage previous to the learning of the controller. Traditionally, this transformation and the resulting high-level input variables are defined using expert knowledge. Doing this preprocessing automatically during the learning phase demands a model that groups the low-level input variables in an expressive and meaningful way. Within this context Quantified Fuzzy Propositions (QFPs) such as *"part of the distances of the frontal sector are low"* are useful for representing relevant knowledge for the experts and therefore for performing intelligent control. Modeling with QFPs as in the previous example demands the definition of several elements:

- *part*: how many distances of the frontal sector must be low?
- *frontal sector*: which beams belong to the frontal sector?
- *low*: what is the actual semantics of low?

This example clearly sets out the need to use propositions that are different from the conventional ones. The use of QFPs in robotics eliminates the need of expert knowledge in two ways: i) the preprocessing of the low-level variables can be embedded in the learning stage; ii) the definition of the high-level variables obtained from low-level variables is done automatically, also during the learning stage. In this paper QFPs are used for representing knowledge about high-level variables that are defined as the grouping of low-level variables. Conventional fuzzy propositions are also used to represent conventional high-level variables, i.e., high-level variables not related to low-level ones (e.g. velocity).

2.4.2 QFRs model

An example of a QFR is shown in Fig. 2.1, involving both QFPs (2.1) and conventional ones (2.2); the outputs of the rule are also fuzzy sets. In order to determine the degree to which the output of the rule will be applied, it is necessary to reason about the propositions (using, for example, the Mamdani's reasoning scheme).

The general expression for QFPs in this case is:

$$d\,(h)\ is\, F_d^i\, in\, Q^i\, of\, F_b^i \tag{2.3}$$

where, for each i=1, ..., g_b^{max} (g_b^{max} being the maximum possible number of sectors of distances):

- $d\,(h)$ is the signal. In this example, it represents the distance measured by beam h.

IF	*part of distances of FRONTAL SECTOR are LOW*	(2.1)
	and	
	velocity is HIGH	(2.2)
THEN	*vlin is VERY LOW*	
	and	
	vang is TURN LEFT	

Figure 2.1: An example of QFR to model the behavior of a mobile robot.

- F_d^i is a linguistic value for variable $d(h)$ (e.g., *"low"*).
- Q^i is a (spatial, defined in the laser beam domain) fuzzy quantifier (e.g., *"part"*).
- F_b^i is a fuzzy set in the laser beam domain (e.g., the *"frontal sector"*).

Evaluation of the Degree of Fulfillment (DOF) for QFP (Eq. 2.3) is carried out using Zadeh's quantification model for proportional quantifiers (such as "most of", "part of", ...) [117]. This model allows to consider non-persistence, partial persistence and total persistence situations for the event "$d(h)$ is F_d^i" in the range of laser beams (spatial interval F_b^i). Therefore, for the considered example, it is possible to make a total or partial assessment on how many distances should be low, in order to decide the corresponding control action. This is a relevant feature of this model, since it allows to consider partial, single or total fulfillment of an event within the laser beams set.

The number of analyzed sectors of distances and their definition may vary for each of the rules. There can be very generic rules that only need to evaluate a single sector consisting of many laser beams, while other rules may need a finer granularity, with more specific laser sectors. Moreover, the rules may require a mix of QFPs and standard fuzzy propositions (for conventional high-level variables). Therefore, the automatic learning of QFRs demands an algorithm with the capability of managing rules with different structures.

2.5 Iterative Quantified Fuzzy Rule Learning of Controllers

2.5.1 Evolutionary learning of Knowledge Bases

Evolutionary learning methods follow two approaches in order to encode rules within a population of individuals [50, 27]:

- Pittsburgh approach: each individual represents the entire rule base.
- Michigan, IRL [25], and GCCL [46]: each individual codifies a rule. The learned rule base is the result of combining several individuals. The way in which the individuals interact during the learning process defines these three different approaches.

The discussion is focused on those approaches for which an individual represents a rule, discarding the Michigan approach as it is used in reinforcement learning problems in which the reward from the environment needs to be maximized [31]. Therefore, the IRL and GCCL approaches are analyzed.

In the IRL approach, the individuals compete among them but only a single rule is learned for each run (epoch) of the evolutionary algorithm. After each sequence of iterations, the best rule is selected and added to the final rule base. The selected rule must be penalized in order to induce niche formation in the search space. A common way to penalize the obtained rules is to delete the training examples that have been covered by the set of rules in the final rule base. The final step of the IRL approach is to check whether the obtained set of rules is a complete knowledge base. In the case it is not, the process is repeated. A weak point of this approach is that the cooperation among rules is not taken into account when a rule is evaluated. For example, a new rule could be added to the final rule base, deteriorating the behavior of the whole rule base over a set of examples that were already covered. The cooperation among rules can be improved with a posterior rules selection process.

In the GCCL approach the entire population codifies the rule base. That is, rules evolve together but competing among them to obtain the higher fitness. For this type of algorithm it is fundamental to include a mechanism to maintain the diversity of the population (niche induction). This mechanism must warrant that individuals of the same niche compete among themselves, but also has to avoid deleting those weak individuals that occupy a niche that remains uncovered. This is usually done using token competition [27].

Although GCCL works well for classification problems [81], the same does not occur for regression problems [85], mostly due to the difficulty of achieving in this realm an adequate

balance between cooperation and competition. It is frequent in regression that an individual tries to capture examples seized by other individual, improving the performance on many of the examples, but decreasing the accuracy on a few ones. In subsequent iterations, new and more specific individuals replace the rule that was weakened. As a result, the individuals improve their individual fitness, but the performance of the knowledge base does not increase. In particular, for mobile robotics, the obtained knowledge bases over-fit the training data due to a *polarization* effect of the rule base: few very general rules and many very specific rules. Moreover, many times, the errors of the individual rules compensate each other, generating a good output of the rule base over the training data, but not on test data.

This proposal, called IQFRL (Iterative Quantified Fuzzy Rule Learning), is based on IRL. The learning process is divided into epochs (set of iterations), and at the end of each epoch a new QFR (Sec. 2.4.2) is obtained. The following sections describe each of the stages of the algorithm (Fig. 2.2).

2.5.2 Examples and Grammar

The learning process is based on a set of training examples. In mobile robotics, each example can be composed of several variables that define the state of the robot (position, orientation, linear and angular velocity, etc.), and the data measured by the sensors. If the robot is equipped with laser range finders, the sensors data are vectors of distances. A laser range finder provides the distances to the closest obstacle in each direction (Fig. 2.3) with a given angular resolution (number of degrees between two consecutive beams). In this paper, each example e^l is represented by a tuple:

$$e^l = (d(1), \ldots, d(N_b), \mathit{velocity}, \mathit{vlin}, \mathit{vang}) \tag{2.4}$$

where $d(h)$ is the distance measured by beam h, N_b is the number of beams (e.g. 722 for a robot equipped with two Sick LMS200 laser range scanners as in Fig. 2.3), *velocity* is the measured linear velocity of the robot, and *vlin* and *vang* are the output variables (control commands for the linear and angular velocities respectively).

The individuals in the population include both conventional propositions and QFPs (Sec. 2.4.2). Also, the number of relevant inputs can be different. Therefore, genetic programming is the most appropriate approach, as each individual is a tree of variable size. In order to generate valid individuals of the population, and to produce right structures for the individuals after crossover and mutation, some constraints have to be added. With a context-free grammar

1: $KB_{cur} := \varnothing$
2: **repeat**
3: $it := 0$
4: $equal_{ind} := 0$
5: Initialization
6: Evaluation
7: **repeat**
8: Selection
9: Crossover and Mutation
10: Evaluation
11: Replacement
12: **if** $best_{ind}^{it-1} = best_{ind}^{it}$ **then**
13: $equal_{ind} := equal_{ind} + 1$
14: **else**
15: $equal_{ind} := 0$
16: $it := it + 1$
17: **until** $(it \geq it_{min} \wedge equal_{ind} \geq it_{check}) \vee (it \geq it_{max})$
18: $KB_{cur} := KB_{cur} \cup best_{ind}$
19: $uncov_{ex} := uncov_{ex} - cov_{ex}$
20: **until** $uncov_{ex} = \varnothing$

Figure 2.2: IQFRL algorithm.

all the valid structures of a tree (genotype) in the population can be defined in a compact form. A context-free grammar is a quadruple (V, Σ, P, S), where V is a finite set of variables, Σ is a finite set of terminal symbols, P is a finite set of rules or productions, and S the start symbol.

The basic grammar is described in Fig. 2.4. As usual, different productions for the same variable are separated by symbol "|". Fig. 2.5 represents a typical chromosome generated with this context-free grammar. Terminal symbols (leaves of the tree) are represented by ellipses, and variables as rectangles. There are two different types of antecedents:

- The sector antecedent. Consecutive beams are grouped into sectors in order to generate more general (high-level) variables (frontal distance, right distance, etc.). This

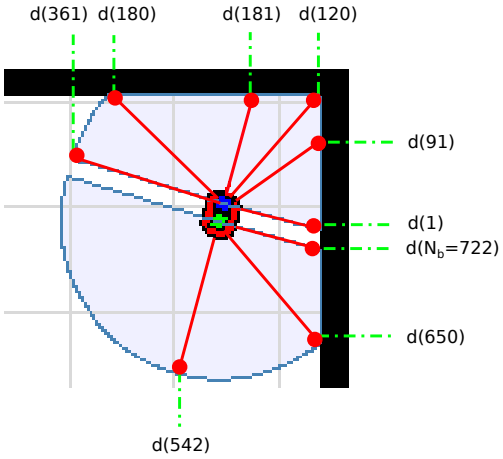


Figure 2.3: Some of the distances measured by a robot equipped with two laser range finders.

type of antecedent is defined by the terminal symbols F_d, F_b and Q: i) the linguistic label F_d represents the measured distances (*HIGH* in Fig. 2.1, prop. 2.1); ii) F_b is the linguistic label that defines the sector, i.e., which beams belong to the sector (*FRONTAL SECTOR* in Fig. 2.1, prop. 2.1); iii) Q is the quantifier (*part* in Fig. 2.1, prop. 2.1).

- The measured linear velocity of the antecedent is defined by the F_v linguistic label.

Finally, F_{lv} and F_{av} are the linguistic labels of the linear and angular velocity control commands respectively, which are the consequents of the rule.

The linguistic labels of the antecedent (F_v, F_d, F_b) are defined using a multiple granularity approach. The universe of discourse of a variable is divided into a different number of equally spaced labels for each granularity. Specifically, a granularity g^i_{var} divides the variable *var* in i uniformly spaced labels, i.e., $A^i_{var} = \{A^{i,1}_{var}, ..., A^{i,i}_{var}\}$. Fig. 2.6 shows a partitioning of up to granularity five. On the other hand, the linguistic labels of the consequents (F_{lv}, F_{av}) are defined using a single granularity approach[2].

[2]Multiple granularity makes no sense if the labels are defined as singletons, which is the usual choice for the output variables in control applications.

- V = { rule, antecedent, consequent, sector }
- Σ = { F_{lv}, F_{av}, F_v, F_d, F_b, Q }
- S = rule
- P:

  (a) rule $\longrightarrow$ antecedent consequent

  (b) antecedent $\longrightarrow$ sector F_v | sector

  (c) consequent $\longrightarrow$ F_{lv} F_{av}

  (d) sector $\longrightarrow$ F_d Q F_b sector | F_d Q F_b

Figure 2.4: Basic context-free grammar for controllers in robotics.

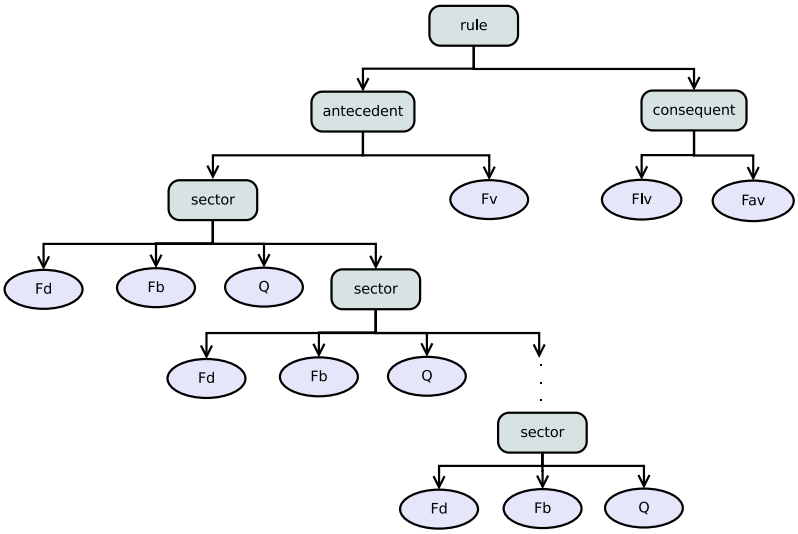


Figure 2.5: An individual representing a QFR that models the behavior of a robot.

2.5.3 Initialization

An individual (Fig. 2.5) is generated for each example in the training set. The consequent part (F_{lv} and F_{av}) is initialized as $F_{var} = A_{var}^{g_{var},\,\beta}$ where $\beta = argmax_j\ \mu_{var}^{g_{var},\,j}\left(e^l\right)$, i.e., the label

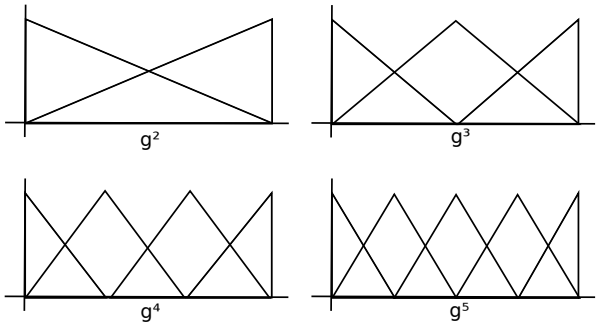


Figure 2.6: Multiple granularity approach from g_x^2 to g_x^5.

with the largest membership value for the example.

The initialization of the antecedent part of a rule requires obtaining the most similar linguistic label to a given fuzzy membership function (which is called mask label). As the maximum granularity of the linguistic labels in the antecedent part of a rule is not limited, the function `maskToLabel` (Fig. 2.7) is applied to obtain the most appropriate linguistic label. This function uses a similarity measure defined as [101]:

$$similarity(F_\phi,\ F_\psi) = 1\ -\ \frac{\sum_{x\in X} |\mu_\phi(x)\ -\ \mu_\psi(x)|}{|X|} \tag{2.5}$$

where F_ϕ and F_ψ are the labels being compared and X is a finite set of points x uniformly distributed on the support of $\phi \cup \psi$.

The `maskToLabel` function (Fig. 2.7) receives a triangular membership function ($mask_{var}$) and searches for the label $A_{var}^{i,\,j}$ with the highest similarity (Eq. 2.5, line 6) with less or equal support (line 5), starting from g_{var}^1 (line 1).

For the initialization of the quantified propositions (sectors), the distances measured in the example are divided into groups of consecutive laser beams whose deviation does not exceed a certain threshold (σ_{bd}). Each group represents a sector that is going to be included in the individual. Afterwards, for each of the previously obtained sectors, the components (F_b, F_d and Q) are calculated:

(a) $F_b = maskToLabel(mask_b)$, with $mask_b = (left_b,\ center_b,\ right_b)$ where $left_b$ is the lower beam of the group, $right_b$ is the higher beam, $center_b$ is the middle beam and the following properties are satisfied: $\mu(left_b) = \mu(right_b) = 0.5$ and $\mu(center_b) = 1$ as shown in Fig. 2.8(a).

Require: $mask_{var}$
1: $i := g^1_{var}$
2: $result := \varnothing$
3: **loop**
4: **for all** $j \in [1, i]$ **do**
5: **if** support($mask_{var}$) $\geq$ support($A^{i,\,j}_{var}$) **then**
6: **if** similarity($mask_{var}$, $A^{i,\,j}_{var}$) $>$ similarity($mask_{var}$, $result$) **then**
7: $result := A^{i,\,j}_{var}$
8: **else**
9: **break loop**
10: $i := i + 1$
11: **return** $result$

Figure 2.7: Function that searches for the most similar label to $mask_{var}$.

(b) $F_d = maskToLabel(mask_d)$, with $mask_d = (\bar{d} - \sigma_d,\ \bar{d},\ \bar{d} + \sigma_d)$ where $\bar{d}$ is the mean of the distances measured by the beams of the group, σ_d is the standard deviation of these distances and the following properties are satisfied: $\mu(\bar{d} - \sigma_d) = \mu(\bar{d} + \sigma_d) = 0.5$ and $\mu(\bar{d}) = 1$ as shown in Fig. 2.8(b).

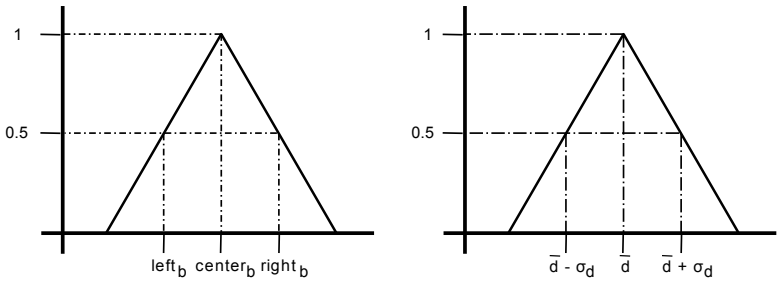
(c) Q (Fig. 2.9) is calculated as the percentage of beams of the sector ($h \in F_b$) that fulfill F_d:

$$Q = \frac{\sum_{h \in F_b} min\left(\mu_{F_d}(d(h)),\ \mu_{F_b}(h)\right)}{\sum_{h \in F_b} \mu_{F_b}(h)} \tag{2.6}$$

Finally, the velocity antecedent F_v is initialized as $F_v = A^{g^i_v,\,\beta}_v$ where $\beta = argmax_j\, \mu^{g^i_v,\,j}_v(e^l)$ and g^i_v is the granularity that satisfies that two consecutive linguistic labels have a separation of σ_v, where σ_v is a threshold of the velocity deviation.

2.5.4 Evaluation

The fitness of an individual of the population is calculated as follows. Firstly, it is necessary to estimate the probability that an example e^l matches the output (C_j) associated to the j-th



(a) $mask_b$ (b) $mask_d$

Figure 2.8: $mask_{var}$ representations for beam (b) and distance (d) variables.

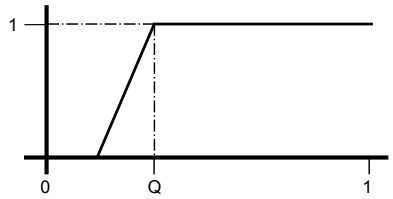


Figure 2.9: Example of a definition of the quantified label Q.

individual rule:

$$P\left(C_j \mid e^l\right) = \exp\left(-\frac{error_j^l}{ME}\right) \tag{2.7}$$

where ME is a parameter that defines the meaningful error and $error_j^l$ is the difference between output C_j and the output codified in the example:

$$error_j^l = \sum_k \left(\frac{y_k^l - c_{j,k}}{max_k - min_k}\right)^2 \tag{2.8}$$

where y_k^l is the value of the k-th output variable of example e^l, $c_{j,k}$ is the output of the k-th output variable associated to individual j, and max_k and min_k are the maximum and minimum values of output variable k. In regression problems, there can be several consequents that are different from the one codified in the example, but that produce small errors, i.e., that are very similar to the desired output. Thus, $P\left(C_j \mid e^l\right)$ can be interpreted as a normal distribution with

covariance ME, and $error_j^l$ is the square of the difference between the mean (output codified in the example) and the output value proposed in the rule codified by the individual.

In an IRL approach, $C_j = C_{R_j}$, i.e., the output coded in individual j is the output associated to rule j. The fitness of an individual in the population is calculated as the combination of two values. On one hand, the accuracy with which the individual covers the examples, called confidence. On the other hand, the ability of generalization of the rule, called support. The confidence can be defined as:

$$confidence = \frac{\rho_u}{\sum_l DOF_j(e_u^l)} \tag{2.9}$$

where $DOF_j(e_u^l)$ is the degree of fulfillment of e_u^l for rule j, and $e_u^l \in uncov_{ex}$, where $uncov_{ex}$ is defined as:

$$uncov_{ex} = \{e^l \ : \ DOF_{KB_{cur}}(e^l) < DOF_{min}\} \tag{2.10}$$

i.e., the set of examples that are covered with a degree of fulfillment below DOF_{min} by the current final knowledge base (KB_{cur}) (line 18, Fig. 2.2), and ρ_u can be defined as:

$$\rho_u = \sum_l DOF_j(e_u^l) \ : \ P\left(C_j \mid e_u^l\right) > P_{min} \quad and \ DOF_j(e_u^l) > DOF_{min} \tag{2.11}$$

where P_{min} is the minimum admissible accuracy. Therefore, the higher the accuracy over the examples covered by the rule (and not covered yet by the current knowledge base), the higher the confidence. Support is calculated as:

$$support = \frac{\rho_u}{\#uncov_{ex}} \tag{2.12}$$

Thus, support measures the percentage of examples that are covered with accuracy, related to the total number of uncovered examples. Finally, $fitness$ is defined as a linear combination of both values:

$$fitness = \alpha_f \cdot confidence + (1 - \alpha_f) \cdot support \tag{2.13}$$

which represents the strength of an individual over the set of examples in $uncov_{ex}$. $\alpha_f \in [0,\ 1]$ is a parameter that codifies the trade-off between accuracy and generalization of the rule.

2.5.5 Crossover

The matching of the pairs of individuals that are going to be crossed is implemented following a probability distribution defined as:

$$P_{close}(\alpha,\ \beta) = 1\ -\ \frac{\sum_{k=1}^{N_c}(\frac{c_{\alpha,\ k}\ -\ c_{\beta,\ k}}{max_k\ -\ min_k})^2}{N_c} \tag{2.14}$$

where $c_{\alpha,\ k}$ ($c_{\beta,\ k}$) is the value of the k-th output variable of individual α (β), and N_c is the number of consequents. With this probability distribution, the algorithm selects with higher probability mates that have similar consequents. The objective is to extract information on which propositions of the antecedent part of the rules are important, and which are not. Crossover has been designed to generate more general individuals, as the initialization of the population produces very specific rules. The crossover operator generates two offsprings:

$$\begin{aligned} offspring_1 &= crossover(ind_i,\ ind_j) \\ offspring_2 &= crossover(ind_j,\ ind_i) \end{aligned} \tag{2.15}$$

This operator modifies a single proposition in antecedent part of the rule. As individuals have a variable number of antecedents, the total number of propositions can be different for two individuals. Moreover, the propositions can be defined using different granularities. Therefore, the first step is to select the propositions (one for each individual) to be crossed between both individuals (Fig. 2.10) as follows:

(a) Get the most specific granularity of the sectors of the individuals to cross (g_b^{max}). Then, an antecedent $m \in [1,\ N_a]$ is selected, where N_a is g_b^{max} plus one, due to the velocity proposition.

(b) Check the existence of this antecedent in both individuals, according to the following criteria:

a) If the antecedent m is a sector, then calculate for each proposition of each individual the similarity between the definition of the sector for the proposition and the linguistic label that defines sector m. Finally, select for each individual the proposition with the highest similarity.

b) If the antecedent m is the velocity, then the corresponding proposition is F_v (in case it exists).

Require: ind_α, ind_β

1: $a_\alpha = a_\beta = \varnothing$
2: $N_a = g_b^{max} + 1$
3: **repeat**
4: $m = random \in [1, N_a]$
5: **if** m is a *sector* **then**
6: $a_\alpha = argmax_r\ similarity(F_{b,r}, A_b^{g_b^{max},m}) \geq 0 : \forall r \in ind_\alpha$
7: $a_\beta = argmax_r\ similarity(F_{b,r}, A_b^{g_b^{max},m}) \geq 0 : \forall r \in ind_\beta$
8: **else**
9: $a_\alpha = F_v \in ind_\alpha$
10: $a_\beta = F_v \in ind_\beta$
11: **until** $(a_\alpha \neq \varnothing) \vee (a_\beta \neq \varnothing)$

Figure 2.10: Selection of antecedents for crossover.

Table 2.1: Crossover operations

Individual 1	Individual 2	Action
no	yes	copy proposition from individual 2 to 1
yes	no	delete proposition in individual 1
yes	yes	combine propositions

Once the propositions to be crossed have been selected, an operation must be picked depending on the existence of the antecedent in both parents (table 2.1):

- If the proposition does not exist in the first individual but exists in the second one, then the proposition of the second individual is copied to the first one, as this proposition could be meaningful.
- If the situation is the opposite to the previous one, then the proposition of the first individual is deleted, as it might be not important.
- If the proposition exists in both individuals, then both propositions are combined in order to obtain a proposition that generalizes both antecedents.

In this last case, the combination of propositions is done by taking into account the degree

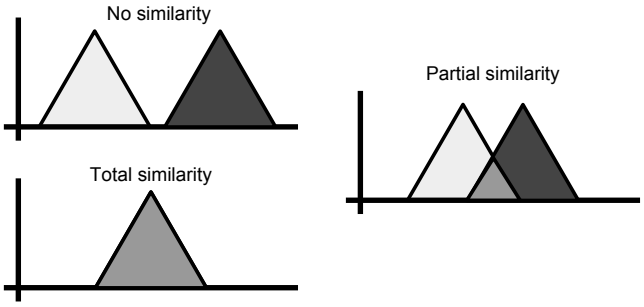


Figure 2.11: Different possibilities of similarity for the labels of equal proposition of two individuals used in the crossover operator.

of similarity (Eq. 2.5) between them (Fig. 2.11). If the proposition is of type sector, the similarity takes into account both F_b and F_d labels. Only when both similarities are partial, the propositions are merged:

- If there is no similarity, then the propositions correspond to different situations. For example, *"the distance is high in part of the frontal sector"* and *"the distance is low in part of the frontal sector"*. This means that the proposition of the first individual might not contain meaningful information and it could be deleted to generalize the rule. For example, both individuals have the proposition *"the distance is high in part of the frontal sector"*.
- If the similarity is total, then, in order to obtain a new individual with different antecedents, the proposition is eliminated.
- Finally, if the similarity is partial, then the propositions are merged in order to obtain a new one that combines the information provided by the two original propositions. For example, *"the distance is high in part of the frontal sector"* and *"the distance is medium-high in part of the frontal sector"*. Therefore, the individual is generalized. The merge action is defined as the process of finding the label with the highest possible granularity that has some similarity with the labels of both original propositions. This is done for both F_b and F_d labels. Q is calculated as the minimum Q of both individuals.

2.5.6 Mutation

If crossover is not performed, both individuals are mutated. Mutation implements two different strategies (Fig. 2.12): generalize or specialize a rule. The higher the value of confidence (Eq. 2.9), the higher the probability to generalize the rule by mutation. This occurs with rules that cover their examples with high accuracy and that could be modified to cover other examples. On the contrary, when the confidence of the individual is low, this means that it is covering some of its examples with a low performance. In order to improve the rule some of the examples that are currently covered should be discarded in order to get a more specific rule.

For generalization, the following steps are performed:

(a) Select an example $e^{sel} \in uncov_{ex}^{j}$, where $uncov_{ex}^{j} = \{e_u^l : DOF_j(e_u^l) < DOF_{min}\}$, i.e. the set of examples that belong to $uncov_{ex}$ and are not covered by individual j. The example is selected with a probability distribution given by $P\left(C_j \mid e_u^l\right)$ (Eq. 2.7). The higher the similarity between the output of the example and the consequent of rule j, the higher the probability of being selected.

(b) The individual is modified in order to cover e^{sel}. Therefore, all the propositions that are not covering the example (those with $\mu_{prop}\left(e^{sel}\right) < DOF_{min}$) are selected for mutation.

a) For sector propositions (Eq. 2.1), there are three different ways in which the proposition can be modified: F_d, F_b, and Q. The modification is selected among the three possibilities, with a probability proportional to the $\mu_{prop}\left(e^{sel}\right)$ value after applying each one.

i. F_d and F_b are generalized choosing the most similar label in the adjacent partition with lower granularity. The process is repeated until $\mu_{prop}\left(e^{sel}\right) \geq DOF_{min}$.

ii. On the other hand, Q is decreased until $\mu_{prop}\left(e^{sel}\right) \geq DOF_{min}$.

b) For velocity propositions (Eq. 2.2), generalization is done choosing the most similar label in the adjacent partition with lower granularity until $\mu_{prop}\left(e^{sel}\right) > DOF_{min}$.

For specialization, the process is equivalent:

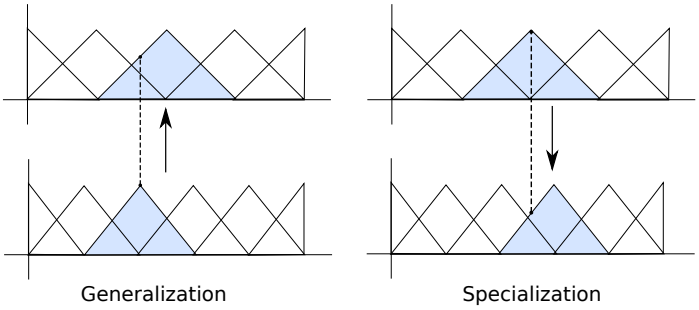


Figure 2.12: The strategies used for mutation for variables d, b and v.

(a) Select an example $e^{sel} \in cov_{ex}^{j}$, where $cov_{ex}^{j} = \{e_u^l : DOF_j(e_u^l) > DOF_{min}\}$, i.e. the set of examples that belong to $uncov_{ex}$ and are covered by individual j. The example is selected with a probability distribution that is inversely proportional to $P\left(C_j \mid e_u^l\right)$ (Eq. 2.7). The higher the similarity between the output of the example and the consequent of rule j, the lower the probability of being selected.

(b) Only one proposition needs to be modified to specialize the individual. This proposition is selected randomly.

a) For sector propositions there are, again, three different ways in which the proposition can be modified: F_d, F_b, and Q. The modification is selected among these three possibilities, with a probability that is inversely proportional to the $\mu_{prop}\left(e^{sel}\right)$ value after applying each one.

i. F_d and F_b are specialized, choosing the most similar label in the adjacent partition with higher granularity. The process is repeated until $\mu_{prop}\left(e^{sel}\right) < DOF_{min}$.

ii. On the other hand, Q is increased until $\mu_{prop}\left(e^{sel}\right) < DOF_{min}$.

b) For velocity propositions, specialization is done by choosing the most similar label in the adjacent partition with higher granularity until $\mu_{prop}\left(e^{sel}\right) < DOF_{min}$.

Finally, once the antecedent is mutated, the consequent also mutates. Again, this mutation requires the selection of an example. If generalization was selected for the mutation of the antecedent, then the example will be e^{sel}. On the other hand, for specialization an example is randomly selected from those currently in cov_{ex}^{j}. For each variable in the consequent part

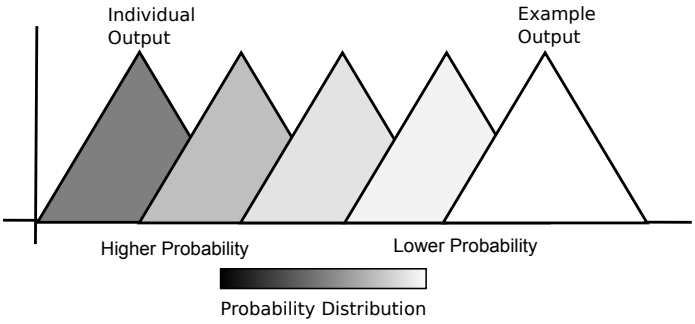


Figure 2.13: Probability distribution example for consequent mutation. Labels closest to the individual output have higher probability to be selected.

of the rule, the label of the individual is modified selecting a label following a probability distribution (Fig. 2.13):

$$P\left(A_{var}^{g_{var},\,\gamma} \mid A_{var}^{g_{var},\,\alpha},\, A_{var}^{g_{var},\,\beta}\right) = 1 \;-\; \frac{|\alpha \;-\; \gamma|}{|\alpha \;-\; \beta| \;+\; 1} \tag{2.16}$$

where $A_{var}^{g_{var},\,\alpha}$ is the label of each of the consequents of the individual, $A_{var}^{g_{var},\,\beta}$ is the label with the largest membership value for e^{sel} and $A_{var}^{g_{var},\,\gamma}$ is a label between them. Thus, the labels closer to the label of the individual have a higher probability to be selected, while the labels closer to the example label have a lower one.

2.5.7 Selection and replacement

Selection has been implemented following the binary tournament strategy. Replacement follows an steady-state approach. The new individuals and those of the previous population are joined, and the best pop_{max} individuals are selected for the next population.

2.5.8 Epoch loop

An epoch is a set of iterations at the end of which a new rule is added to KB_{cur}. The stopping criterion of each epoch (inner loop in Fig. 2.2) is the number of iterations, but this limit varies according to the following criteria: once the number of iterations (it) reaches it_{min}, the algorithm stops if there are it_{check} consecutive iterations (counted by $equal_{ind}$) with no change

in the best individual ($best_{ind}$). If the number of iterations reaches the maximum (it_{max}), then the algorithm stops regardless of the previous condition.

When the epoch ends, the rule defined in $best_{ind}$ is added to KB_{cur}. Moreover, the examples that are covered with accuracy (according to the criterion in Eq. 2.11) are marked as covered by the algorithm (line 19, Fig. 2.2). Finally, the algorithm stops when there are no uncovered examples.

2.5.9 Rule subset selection

After the end of the iterative part of the algorithm, the performance of the obtained rule base can be improved selecting a subset of rules with better cooperation among them. The rule selection algorithm described in [81] has been used. The rule selection process has the following steps:

(a) Generate $\#R_{gp}$ rule bases, where $\#R_{gp}$ is the number of rules of the population obtained by the IQFRL algorithm (RB_{gp}) Each rule base is coded as: $RB_i = r_1^i \cdots r_{\#R_{gp}}^i$, with:

$$r_j^i = \begin{cases} 0, & if\ j > i \\ 1, & if\ j \leq i \end{cases} \tag{2.17}$$

where r_j^i indicates if the j-th rule of RB_{gp} is included ($r_j^i = 1$) or not ($r_j^i = 0$) in RB_i. With this codification, RB_i will contain the best i rules of RB_{gp}, as these rules have been ranked in decreasing order of their individual fitness. Notice that $RB_{\#R_{gp}}$ is RB_{gp}

(b) Evaluate all the rule bases, and select the best one, RB_{sel}.

(c) Execute a local search on RB_{sel} to obtain the best rule set, RB_{best}.

The last step was implemented with the iterated local search (ILS) algorithm [71].

threshold (maxRestarts).

2.6 Results

2.6.1 Experimental setup

The proposed algorithm has been validated with the well-known in mobile robotics wall-following behavior. The main objectives of a controller for this behavior are: to keep a suitable

Figure 2.14: *Pioneer 3-AT* robot equipped with two laser range scanners.

distance between the robot and the wall, to move at the highest possible velocity, and to implement smooth control actions. The Player/Stage robot software [44] has been used for the tests on the simulated environments and also for the connection with the real robot *Pioneer 3-AT* (Fig. 2.14). This real robot was equipped with two laser range scanners with an amplitude of 180° and a precision of 0.5° (i.e. 361 measurements for each laser scan). Without loss of generality, all the examples and tests here described were made with the robot following the wall at its right.

The examples that have been used for learning were generated for three different situations (Fig. 2.15) that have been identified by an expert:

(a) Convex corner: it is characterized by the existence of a gap in the wall (like an open door) (labeled CX in Fig. 2.15).

(b) Concave corner: it is a situation in which the robot finds a wall in front of it (labeled CC in Fig. 2.15).

(c) Straight wall: any other situation (labeled SW in Fig. 2.15).

For each of the above situations, the robot was placed in different positions and the associated control order was the one that minimized the error. Therefore, each example consists of 722 distances (one for each laser beam), the current linear velocity of the robot, and the

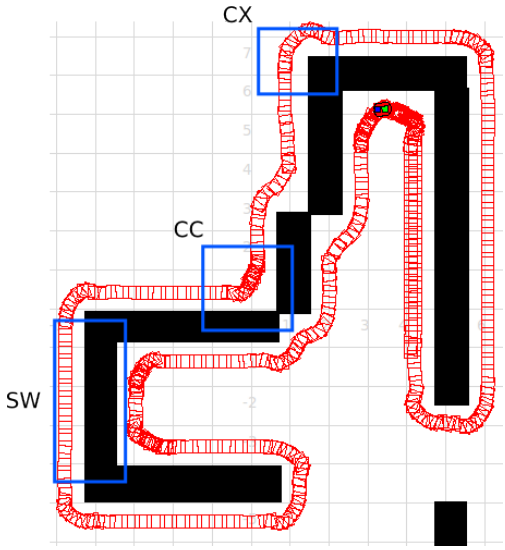


Figure 2.15: The three different situations for the wall-following behavior.

control commands (linear and angular velocity). The expert always tried to follow the wall at, approximately, 50 cm and the maximum values for the linear and angular velocities were 50 cm/s and $45^o s^{-1}$ respectively. 572 training examples were generated for the straight wall situation, 540 for the convex corner and 594 for the concave corner.

The IQFRL algorithm was used to learn a different controller for each of the three situations. In order to decide which knowledge base should be used at each time instant, the classification version of IQFRL (IQFRL-C, see 2.A) was used. In this way, IQFRL learning could be tested with three completely different controllers.

In order to analyze the performance of the proposed learning algorithm, several tests were done in 15 simulated environments and two real ones. Table 2.2 shows some of the characteristics of the environments: the dimensions of the environment, the path length, the number of concave (#CC) and convex (#CX) corners, and the number of times that the robot has to cross a door (#doors). The action of crossing a door represents a high difficulty as the robot has to negotiate a convex corner with a very close wall in front of it.

The simulated environments are shown in Figs. 2.16 and 2.17. The trace of the robot is represented by marks, and the higher the concentration of marks, the lower the velocity of the robot. Furthermore, Fig. 2.18 shows the real environments. Each of them represents an

Table 2.2: Characteristics of the test environments

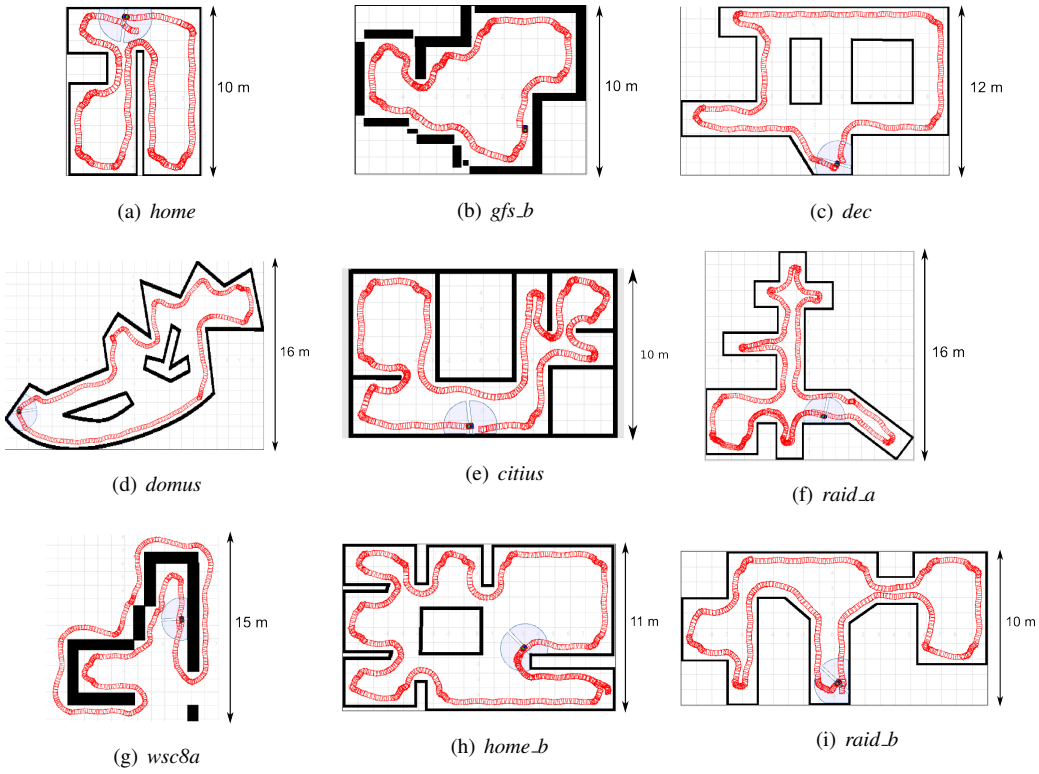
Environment	Dim. ($m \times m$)	Length (m)	#CC	#CX	#doors
home	8×10	20	8	3	1
gfs_b	14×10	43	10	6	0
dec	19×12	53	8	4	0
domus	26×16	60	9	6	3
citius	16×10	63	12	6	2
raid_a	16×16	66	16	12	0
wsc8a	15×15	70	4	7	1
home_b	18×11	76	17	6	2
raid_b	20×10	86	12	10	2
rooms	19×19	86	12	6	4
flower	22×20	98	9	6	1
office	26×26	146	23	10	8
autolab	26×28	154	21	11	10
maze	18×18	205	13	9	0
hospital	74×45	1046	98	69	43
real env 1	9×8	20	7	3	0
real env 2	10×5	26	7	3	0

occupancy grid map of the environment, together with the trajectory of the robot.

2.6.2 Algorithms and parameters

The following values were used for the parameters of the evolutionary algorithm: $ME = 0.02$, $DOF_{min} = 0.001$, $\alpha_f = 0.99$, $P_{cross} = 0.8$, $pop_{max} = 70$, $it_{min} = 50$, $it_{check} = 10$, $it_{max} = 100$, $\sigma_{bd} = 0.01$, $\sigma_v = 0.1$ and $P_{min} = 0.17$. P_{min} is a parameter that has a high influence in the performance of the system. A single value of P_{min} was used in testing, obtained from Eqs. 2.7 and 2.8 for the case the error for each consequent is one label (Eq. 2.8). The granularities and the universe of discourse of each output of a rule are shown in table 2.3. For the rule subset selection algorithm, the parameters have values of $radius_{nbhood} = 1$ and $maxRestarts = 2$.

The fuzzy inference system used for the learned fuzzy rule sets uses the minimum t-norm for both the implication and conjunction operators, and the weighted average method as



(a) *home* (b) *gfs_b* (c) *dec*

(d) *domus* (e) *citius* (f) *raid_a*

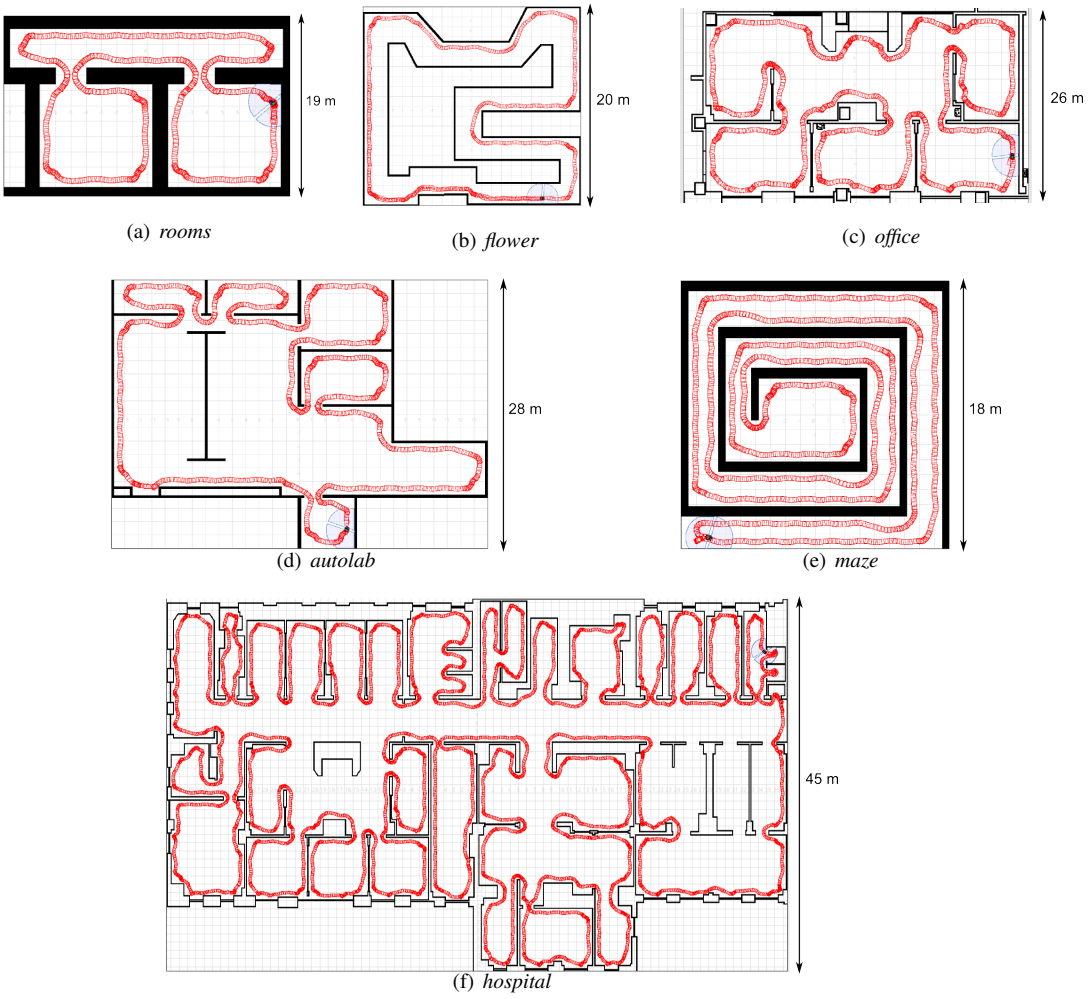
(g) *wsc8a* (h) *home_b* (i) *raid_b*

Figure 2.16: Path of the robot along the simulated environments (I).

Table 2.3: Universe of discourse and granularities

Variable	Min	Max	Granularities
Distance	0	1.5	All
Beam	0	721	All
Quantifier	10	100	–
Velocity	0	0.5	All
Lineal velocity	0	0.5	$\{9\}$
Angular velocity	$-\pi/4$	$\pi/4$	$\{19\}$

defuzzification operator.



(a) *rooms* (b) *flower* (c) *office*

(d) *autolab* (e) *maze*

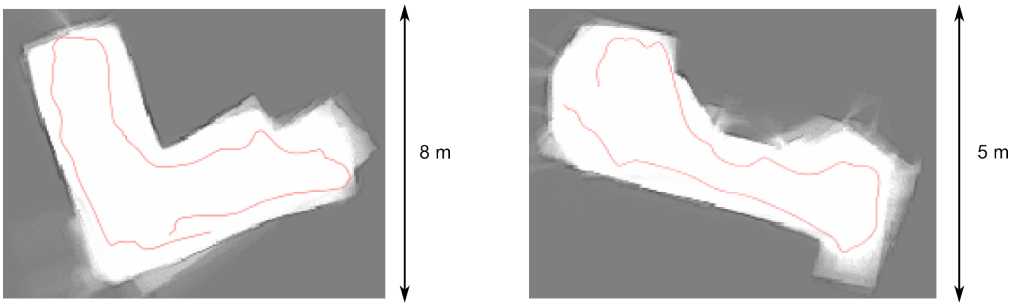
(f) *hospital*

Figure 2.17: Path of the robot along the simulated environments (II).

The IQFRL approach was compared with three different algorithms:

- Methodology to Obtain Genetic fuzzy rule-based systems Under the iterative Learning approach (MOGUL): a three-stage genetic algorithm [24]:

  (a) An evolutionary process for learning fuzzy rules, with two components: a fuzzy-rule generating method based on IRL, and an iterative covering method.



(a) *real environment 1* (b) *real environment 2*

Figure 2.18: Path of the robot along the real environments.

(b) A genetic simplification process for selecting rules.

(c) A genetic tuning process, that tunes the membership functions for each fuzzy rule or for the complete rule base.

The soft-constrained MOGUL was used, as it has better performance in very hard problems [25][3].

- Multilayer Perceptron Neural Network (MPNN): a single-hidden-layer neural network trained with the BFGS method [103] with the following parameters: $abstol = 0.01$, $reltol = 0.0001$ and $maxit = 500$. The number of neurons in the hidden layer varies from n to $2 \cdot n$, being n the number of inputs[4].
- ν-Support Vector Regression (ν-SVR)[5]: a ν-SVM [99] version for regression with a Gaussian RBF kernel. The parameter sigma is estimated based upon the 0.1 and 0.9 quantile of $||x - x'||^2$.

As mentioned before, in the IQFRL proposal the preprocessing of raw sensor data is embedded in the learning algorithm. Since the algorithms for the comparison need to preprocess the data before the learning phase, three different approaches were used for the transformation of the sensor data:

[3]The implementation in *Keel* [5], an open source (GPLv3) Java software tool to assess evolutionary algorithms for Data Mining problems, was used.

[4]The package *nnet* [111] of the statistical software R was used.

[5]The package *kernlab* [60] of the statistical software R was used.

Table 2.4: Different configurations for the preprocessing methods.

Preprocessing	Configuration
Min (n)	$\{4, 8, 16, 32, 64\}$
Sample (n)	$\{4, 8, 16, 32, 64\}$
PCA (σ_{PCA})	$\{0.90, 0.95, 0.975, 0.99, 0.999\}$

Table 2.5: Number of inputs obtained with PCA.

σ_{PCA}	Straight	Convex	Concave
0.90	35	15	27
0.95	51	24	40
0.975	66	35	53
0.99	85	57	68
0.999	127	99	109

- *Min*: the beams of the laser range finder are grouped in n equal sized sectors. For each sector, the minimum distance value is selected as input.
- *Sample*: n equidistant beams are selected as the input data.
- PCA: Principal Component Analysis computes the most meaningful basis to re-express the data. It is a simple, non-parametric method for extracting relevant information. The variances associated with the principal components can be examined in order to select only those that cover a percentage of the total variance.

Different parameters have been used for the preprocessing approaches. For *Min* and *Sample* methods, the number of obtained inputs (n) was changed. For *PCA*, the percentage of variance (σ_{PCA}) indicates the principal components selected as input data. Table 2.4 shows the parameters used for the preprocessing methods. Moreover, table 2.5 shows the number of inputs obtained with PCA for the three datasets with each configuration.

2.6.3 Comparison and statistical significance

Table 2.6 shows the training and test errors over a 5-fold cross-validation. For each algorithm and dataset the mean and standard deviation of the error (Eq. 2.8) were calculated.

Table 2.6: Training and test errors

Alg.	Preproc.	Dataset	Training	Test
IQFRL	–	Straight	0.11 ± 0.03	0.14 ± 0.03
		Convex	0.10 ± 0.01	0.12 ± 0.02
		Concave	0.04 ± 0.01	0.05 ± 0.01
MOGUL	min 16	other	0.01 ± 0.00	0.10 ± 0.01
		convex	0.01 ± 0.00	0.05 ± 0.01
		concave	0.00 ± 0.00	0.05 ± 0.01
	sample 16	other	0.01 ± 0.00	0.09 ± 0.02
		convex	0.02 ± 0.00	0.05 ± 0.01
		concave	0.00 ± 0.00	0.04 ± 0.01
MPNN	min 8	other	0.01 ± 0.00	0.06 ± 0.10
		convex	0.02 ± 0.01	0.03 ± 0.05
		concave	0.00 ± 0.00	0.04 ± 0.06
	sample 8	other	0.02 ± 0.00	0.18 ± 0.25
		convex	0.03 ± 0.01	0.02 ± 0.02
		concave	0.01 ± 0.00	0.17 ± 0.34
ν-SVR	min 16	other	0.01 ± 0.00	0.02 ± 0.02
		convex	0.03 ± 0.01	0.02 ± 0.01
		concave	0.01 ± 0.00	0.00 ± 0.00
	sample 16	other	0.02 ± 0.01	0.02 ± 0.02
		convex	0.04 ± 0.02	0.02 ± 0.01
		concave	0.01 ± 0.00	0.01 ± 0.00

For each preprocessing technique, a 5-fold cross-validation was performed for each combination of the parameters of the algorithms. For example, for the *Min* preprocessing with 16 equal size sectors, a 5-fold cross-validation was run for each number of neurons between 17 and 34 for the MPNN approach. Only the configuration of the algorithm with lowest test error for each configuration of the preprocessing methods was used for comparison purposes. Moreover, only those configurations of preprocessing techniques with the best results are

shown in the tables of this section. Results for *PCA* preprocessing have not been included, as the learning algorithms were not able to obtain adequate controllers.

Although, the MSE (Mean Squared Error) is the usual measure of the performance of the algorithms, this is not a sufficient criterion in mobile robotics. A good controller must be robust and able to provide a good and smooth output in any situation. The only way to validate the controller is to test it on environments (simulated and real) with different difficulties and assessing on these tests a number of quality parameters such as mean distance to the wall, mean velocity along the paths, ...

Table 2.7 contains the results of the execution of each of the algorithms for the different simulated environments (Figs. 2.16 and 2.17). Furthermore, table 2.8 shows the average results for the following five different indicators: the distance to the wall at its right (Dist.), the linear velocity (Vel.), the change in the linear velocity between two consecutive cycles (Vel.ch.) —which reflects the smoothness in the control—, the time per lap, and the number of blockades of the robot along the path and cannot recover.

The robot is blocked if it hits a wall or if it does not move for 5 s. In this situation the robot is placed parallel to the wall at a distance of 0.5 m. The average values of the five indicators are calculated for each lap that the robot performs in the environment. Results presented in the table are the average and standard deviation values over five laps of the average values of the indicators over one lap. The dash symbol in the results table indicates that the controller could not complete the path. This usually occurs when the number of blockades per meter is high (greater than 5 blockades in a short period of time) or when the robot completely deviates from the path.

Moreover, in order to evaluate the performance of a controller with a numerical value a general quality measure was defined. It is based on the error measure defined in [80], but including the number of blockades:

$$quality = \frac{1}{1 + (1 + \#Blockades) \cdot (0.9 \cdot |Dist - d_{wall}| + 0.1 \cdot |Vel - v_{max}|)} \tag{2.18}$$

where d_{wall} is the reference distance to the wall (50 cm) and v_{max} is the maximum value of the velocity (50 cm/s). The higher the quality, the better the controller. This measure takes the number of blockades into account in a linear form for comparison purposes. However, it should be noted that controllers with just a single blockade are not reliable and should not be implemented on a real robot.

Table 2.7: Average results ($x \pm \sigma$) for each simulated environment

Alg.	Prepr.	Env.	Dist.(cm)	Vel.(cm/s)	Vel.ch.(cm/s)	Time(s)	# Blockades	*quality*
IQFRL	–	home	55.70 ± 0.25	27.00 ± 0.66	5.59 ± 0.14	164.63 ± 5.26	0.00 ± 0.00	0.12
		gfs_b	55.98 ± 1.70	22.37 ± 0.92	7.01 ± 0.74	163.90 ± 9.72	0.00 ± 0.00	0.11
		dec	57.33 ± 1.02	32.47 ± 0.67	5.83 ± 0.18	168.63 ± 1.76	0.00 ± 0.00	0.11
		domus	54.84 ± 0.53	29.97 ± 0.19	5.97 ± 0.49	198.80 ± 1.39	0.00 ± 0.00	0.14
		citius	54.80 ± 0.87	26.11 ± 0.78	6.28 ± 0.64	249.50 ± 8.25	0.00 ± 0.00	0.13
		raid_a	59.56 ± 0.72	25.35 ± 0.14	6.87 ± 0.55	262.00 ± 6.15	0.00 ± 0.00	0.08
		wsc8a	56.96 ± 1.00	27.45 ± 0.84	7.70 ± 0.33	233.10 ± 5.28	0.00 ± 0.00	0.11
		home_b	58.41 ± 0.72	25.53 ± 0.46	7.06 ± 0.36	300.07 ± 8.15	0.00 ± 0.00	0.09
		raid_b	58.22 ± 0.44	28.57 ± 0.40	6.60 ± 0.47	242.23 ± 3.82	0.00 ± 0.00	0.09
		rooms	57.38 ± 0.34	30.97 ± 0.34	6.38 ± 0.43	261.93 ± 4.60	0.00 ± 0.00	0.10
		flower	53.46 ± 0.25	33.85 ± 0.40	4.13 ± 0.34	290.77 ± 4.13	0.00 ± 0.00	0.17
		office	51.37 ± 0.57	24.20 ± 0.18	6.65 ± 0.25	578.27 ± 2.92	0.00 ± 0.00	0.21
		autolab	52.91 ± 0.20	28.75 ± 0.31	5.57 ± 0.48	499.33 ± 9.74	0.00 ± 0.00	0.17
		maze	52.43 ± 0.22	35.88 ± 0.40	3.64 ± 0.28	567.73 ± 5.29	0.00 ± 0.00	0.22
		hospital	51.09 ± 0.19	26.68 ± 0.10	6.18 ± 0.35	3608.07 ± 21.72	0.00 ± 0.00	0.23
MOGUL	min 16	home	55.12 ± 0.69	30.43 ± 1.30	5.40 ± 0.45	181.10 ± 12.70	7.33 ± 2.05	0.05
		gfs_b	54.75 ± 0.96	24.44 ± 1.02	6.81 ± 0.39	208.87 ± 12.86	14.00 ± 2.16	0.04
		dec	55.46 ± 1.14	36.13 ± 0.43	5.17 ± 0.15	190.50 ± 6.29	8.67 ± 1.25	0.07
		domus	55.75 ± 0.64	31.44 ± 1.80	5.47 ± 0.33	224.60 ± 10.25	8.33 ± 0.47	0.09
		citius	53.47 ± 1.27	29.48 ± 0.13	5.60 ± 0.53	302.57 ± 13.58	18.33 ± 2.62	0.05
		raid_a	57.53 ± 0.32	26.53 ± 0.28	6.41 ± 0.09	363.87 ± 10.21	27.33 ± 3.40	0.02
		wsc8a	54.80 ± 0.35	27.57 ± 0.63	6.26 ± 0.59	346.90 ± 29.40	26.67 ± 5.44	0.02
		home_b	56.75 ± 0.49	27.49 ± 0.62	6.58 ± 0.37	379.57 ± 2.05	22.00 ± 1.41	0.05
		raid_b	57.48 ± 0.70	32.38 ± 0.17	5.84 ± 0.38	280.17 ± 14.29	14.67 ± 3.40	0.03
		rooms	54.79 ± 0.38	30.57 ± 1.04	5.14 ± 0.37	350.33 ± 28.04	20.00 ± 5.89	0.02
		flower	53.33 ± 0.39	38.05 ± 1.00	3.70 ± 0.71	310.27 ± 13.41	11.67 ± 4.03	0.05
		office	51.48 ± 0.29	25.09 ± 0.49	6.77 ± 0.20	762.20 ± 5.28	49.67 ± 1.25	0.10
		autolab	51.95 ± 0.71	30.54 ± 1.24	5.23 ± 0.27	612.00 ± 23.46	31.00 ± 2.45	0.07
		maze	52.25 ± 0.55	37.55 ± 1.53	2.87 ± 0.25	690.93 ± 53.92	32.00 ± 6.38	0.04
		hospital	51.33 ± 0.06	26.86 ± 0.08	5.89 ± 0.29	4908.07 ± 56.12	313.33 ± 10.34	0.02
	sample 16	home	56.76 ± 0.20	29.57 ± 0.35	4.73 ± 0.15	161.97 ± 0.69	1.67 ± 0.47	0.08
		gfs_b	56.16 ± 1.62	23.66 ± 0.88	8.16 ± 0.51	160.80 ± 9.65	1.67 ± 1.25	0.05
		dec	57.69 ± 0.66	37.95 ± 0.96	6.03 ± 0.25	148.87 ± 3.94	0.67 ± 0.47	0.08
		domus	56.04 ± 0.20	36.61 ± 1.14	6.35 ± 0.57	165.63 ± 8.93	1.00 ± 0.82	0.08
		citius	51.38 ± 1.72	27.40 ± 0.21	6.13 ± 0.52	241.53 ± 3.38	1.00 ± 0.82	0.14
		raid_a	57.44 ± 0.51	26.18 ± 1.39	6.61 ± 0.44	275.63 ± 18.82	3.67 ± 2.05	0.03
		wsc8a	54.67 ± 0.24	30.18 ± 0.65	9.03 ± 0.42	220.87 ± 2.26	1.67 ± 0.94	0.08
		home_b	57.17 ± 0.36	26.80 ± 0.91	6.73 ± 0.72	303.77 ± 9.52	3.00 ± 0.82	0.06
		raid_b	60.38 ± 0.38	34.73 ± 1.26	6.15 ± 0.41	206.20 ± 10.12	0.33 ± 0.47	0.06
		rooms	56.05 ± 0.09	32.11 ± 1.49	6.39 ± 0.64	254.50 ± 18.03	0.67 ± 0.94	0.07
		flower	55.24 ± 0.58	41.58 ± 0.32	3.92 ± 0.27	244.67 ± 6.56	2.00 ± 1.41	0.07
		office	50.33 ± 0.10	22.76 ± 0.56	6.22 ± 0.46	655.40 ± 21.32	11.33 ± 2.36	0.09
		autolab	50.57 ± 0.27	29.62 ± 0.69	5.29 ± 0.32	498.67 ± 8.12	2.33 ± 1.25	0.15
		maze	55.05 ± 0.34	40.22 ± 0.78	3.25 ± 0.18	512.33 ± 8.86	0.67 ± 0.47	0.11
		hospital	51.54 ± 0.25	25.97 ± 0.14	6.11 ± 0.34	3964.70 ± 15.81	64.67 ± 8.18	0.03
MPNN	min 8	home	58.34 ± 1.07	29.34 ± 2.66	4.12 ± 0.11	122.73 ± 6.36	6.33 ± 0.47	0.07
		gfs_b	58.56 ± 1.41	28.14 ± 0.71	7.66 ± 0.28	149.20 ± 11.03	4.33 ± 1.70	0.04
		dec	56.10 ± 0.32	35.13 ± 0.38	4.28 ± 0.19	173.37 ± 5.58	3.33 ± 0.94	0.07
		domus	–	–	–	–	–	0.00
		citius	55.73 ± 0.90	27.22 ± 0.68	4.97 ± 0.28	324.37 ± 31.51	15.00 ± 3.56	0.03
		raid_a	57.90 ± 0.61	23.05 ± 1.01	6.85 ± 0.26	403.60 ± 13.17	27.33 ± 1.70	0.04

		wsc8a	56.24 ± 0.81	30.96 ± 1.03	7.97 ± 0.11	238.77 ± 5.45	7.33 ± 0.47	0.08
		home_b	58.02 ± 0.54	24.19 ± 1.81	7.13 ± 0.45	922.87 ± 568.90	66.33 ± 41.02	0.00
		raid_b	59.99 ± 1.57	27.98 ± 2.64	5.07 ± 0.63	1137.37 ± 842.79	38.33 ± 20.53	0.00
		rooms	–	–	–	–	–	0.00
		flower	–	–	–	–	–	0.00
		office	55.84 ± 0.48	28.48 ± 0.19	8.18 ± 0.32	626.33 ± 5.59	32.00 ± 1.41	0.05
		autolab	–	–	–	–	–	0.00
		maze	52.75 ± 0.32	42.53 ± 1.09	2.81 ± 0.38	621.07 ± 45.13	28.00 ± 3.56	0.06
		hospital	55.94 ± 0.02	28.45 ± 0.33	7.47 ± 0.17	3730.00 ± 166.69	205.00 ± 13.93	0.01
		home	–	–	–	–	–	0.00
		gfs_b	62.11 ± 0.47	21.96 ± 0.35	7.22 ± 0.14	172.00 ± 0.78	1.00 ± 0.00	0.07
		dec	64.67 ± 2.80	30.23 ± 4.21	6.03 ± 0.49	285.70 ± 118.55	1.00 ± 0.82	0.04
		domus	–	–	–	–	–	0.00
		citius	68.36 ± 5.48	20.98 ± 2.23	6.69 ± 0.18	603.33 ± 243.12	16.33 ± 11.81	0.00
		raid_a	74.58 ± 8.62	19.36 ± 3.61	6.41 ± 1.03	450.60 ± 259.59	5.00 ± 3.56	0.01
		wsc8a	61.04 ± 0.62	23.70 ± 0.41	7.90 ± 0.52	279.37 ± 6.70	1.00 ± 0.82	0.04
	sample 8	home_b	85.20 ± 8.36	16.95 ± 3.09	6.40 ± 0.62	2477.03 ± 1471.27	26.33 ± 15.15	0.00
		raid_b	70.10 ± 4.50	21.41 ± 1.56	7.18 ± 0.48	1780.97 ± 1445.69	49.00 ± 43.69	0.00
		rooms	60.75 ± 0.47	34.39 ± 0.65	6.81 ± 0.22	237.53 ± 5.88	0.00 ± 0.00	0.08
		flower	61.77 ± 2.11	28.82 ± 0.35	7.54 ± 0.21	912.67 ± 231.62	51.33 ± 13.02	0.01
		office	57.22 ± 1.31	21.01 ± 0.31	5.58 ± 0.15	783.03 ± 7.34	28.33 ± 0.47	0.07
		autolab	–	–	–	–	–	0.00
		maze	–	–	–	–	–	0.00
		hospital	74.51 ± 11.21	21.74 ± 3.14	5.33 ± 1.18	555.43 ± 721.71	16.33 ± 23.10	0.00
		home	–	–	–	–	–	0.00
		gfs_b	57.82 ± 0.58	26.35 ± 0.81	8.67 ± 0.52	140.20 ± 3.94	0.00 ± 0.00	0.10
		dec	59.14 ± 0.05	39.01 ± 0.98	6.59 ± 0.48	143.03 ± 3.39	0.00 ± 0.00	0.10
		domus	–	–	–	–	–	0.00
		citius	55.14 ± 0.78	25.65 ± 0.29	5.90 ± 0.28	258.87 ± 2.23	0.00 ± 0.00	0.12
		raid_a	58.52 ± 0.29	30.31 ± 0.67	9.36 ± 0.28	208.97 ± 4.84	0.00 ± 0.00	0.09
		wsc8a	58.33 ± 0.24	30.09 ± 0.27	10.70 ± 0.38	218.33 ± 1.56	0.00 ± 0.00	0.10
	min 16	home_b	61.26 ± 0.46	27.87 ± 0.50	8.05 ± 0.10	289.40 ± 2.94	1.00 ± 0.00	0.07
		raid_b	–	–	–	–	–	0.00
		rooms	–	–	–	–	–	0.00
		flower	58.66 ± 0.11	38.42 ± 0.64	4.76 ± 0.22	257.10 ± 4.64	0.00 ± 0.00	0.10
		office	51.37 ± 0.47	23.92 ± 0.40	7.67 ± 0.10	582.23 ± 5.47	0.00 ± 0.00	0.21
		autolab	54.18 ± 0.25	28.04 ± 0.22	6.38 ± 0.19	522.07 ± 3.83	0.67 ± 0.47	0.10
		maze	61.07 ± 0.70	32.60 ± 1.02	3.06 ± 0.17	675.67 ± 63.68	1.00 ± 0.82	0.04
v-SVR		hospital	53.42 ± 0.36	25.42 ± 0.08	6.58 ± 0.21	3833.90 ± 13.48	6.67 ± 1.70	0.06
		home	–	–	–	–	–	0.00
		gfs_b	57.63 ± 0.24	27.77 ± 0.64	8.21 ± 0.62	132.20 ± 3.00	0.00 ± 0.00	0.10
		dec	57.31 ± 0.03	39.00 ± 0.29	5.72 ± 0.10	142.47 ± 0.66	0.00 ± 0.00	0.12
		domus	–	–	–	–	–	0.00
		citius	55.31 ± 0.82	29.69 ± 0.58	6.28 ± 0.24	221.00 ± 3.99	0.00 ± 0.00	0.13
		raid_a	56.89 ± 0.09	31.38 ± 0.20	8.37 ± 0.35	201.27 ± 0.59	0.00 ± 0.00	0.11
		wsc8a	56.27 ± 0.03	32.09 ± 0.21	9.37 ± 0.31	203.50 ± 1.84	0.00 ± 0.00	0.12
	sample 16	home_b	60.62 ± 0.70	29.24 ± 0.06	8.27 ± 0.09	275.70 ± 6.91	0.33 ± 0.47	0.06
		raid_b	57.54 ± 1.24	36.64 ± 0.70	6.25 ± 0.44	273.67 ± 9.39	12.33 ± 1.25	0.05
		rooms	57.95 ± 0.49	34.88 ± 0.26	6.37 ± 0.26	233.53 ± 0.12	0.00 ± 0.00	0.10
		flower	56.64 ± 0.14	40.23 ± 0.13	5.16 ± 0.06	244.13 ± 1.13	0.00 ± 0.00	0.13
		office	51.34 ± 0.13	26.54 ± 0.11	7.65 ± 0.16	522.00 ± 3.41	0.00 ± 0.00	0.22
		autolab	53.26 ± 0.17	31.50 ± 0.38	6.19 ± 0.04	462.23 ± 3.94	0.00 ± 0.00	0.17
		maze	–	–	–	–	–	0.00
		hospital	52.54 ± 0.13	28.57 ± 0.15	6.71 ± 0.19	3359.17 ± 13.58	0.00 ± 0.00	0.18

Table 2.8: Average results ($x \pm \sigma$) for all simulated environments

Alg.	Prepr.	Dist.(cm)	Vel.(cm/s)	Vel.ch.(cm/s)	# Blockades	*quality*
IQFRL	–	55.36 ± 2.57	28.34 ± 3.60	6.10 ± 1.04	0.00 ± 0.00	0.14 ± 0.05
MOGUL	min 16	54.42 ± 1.99	30.30 ± 4.13	5.54 ± 1.05	40.33 ± 73.78	0.05 ± 0.02
	sample 16	55.10 ± 2.83	31.02 ± 5.76	6.07 ± 1.39	6.42 ± 15.78	0.08 ± 0.03
MPNN	min 16	56.86 ± 1.87	29.59 ± 5.09	6.05 ± 1.76	39.39 ± 55.37	0.03 ± 0.03
	sample 16	67.30 ± 7.88	23.69 ± 4.99	6.64 ± 0.76	17.79 ± 18.12	0.02 ± 0.03
ν-SVR	min 16	57.17 ± 3.05	29.79 ± 4.83	7.07 ± 2.05	0.85 ± 1.88	0.07 ± 0.06
	sample 16	56.11 ± 2.49	32.29 ± 4.27	7.05 ± 1.23	1.05 ± 3.40	0.10 ± 0.06

Table 2.9: Non-parametric test for *quality* of table 2.7.

Alg.	Preprocessing	Ranking	Holm p-value
IQFRL	–	1.53	–
MOGUL	min 16	4.9	0.012
	sample 16	3.47	0.025
MLPNN	min 8	5.57	0.010
	sample 8	6	0.008
ν-SVR	min 16	3.83	0.017
	sample 16	2.7	0.05
Friedman p-value = 0.00			
Holm's rejects hypothesis with p-value $<= 0.05$			

Table 2.10: Average results ($x \pm \sigma$) of IQFRL for the real environments

Env.	Dist.(cm)	Vel.(cm/s)	Vel.ch.(cm/s)	Time(s)	# Blockades	*quality*
real env 1	54.13 ± 2.59	19.86 ± 1.52	1.36 ± 0.21	100.70	0.00 ± 0.00	0.13
real env 2	59.29 ± 2.74	21.94 ± 1.43	1.72 ± 2.50	118.50	0.00 ± 0.00	0.08

In general, all the algorithms except MPNN with *Sample 16* preprocessing, produced a distance that is very close to the reference (between 40 cm and 60 cm to the wall at its right).

Note that in cases where the best distance is very different from that obtained by IQFRL, this is because several blockades happened. Therefore, those controllers have the advantage of being continually repositioned into the perfect situation. The best results in speed are those obtained by ν-SVR and MOGUL but, in general, due to a worsening in the distance to the wall or an increase in the number of blockades. The same applies to the speed change. In those cases where it is too low, like in some cases for MOGUL or MPNN, the robot is not able to trace some curves safely. IQFRL is the approach that gets the best quality values, reflecting not only the adequate values for the distance, velocity and smoothness in all the environments but, also, its robustness: it is the unique approach that never blocked or failed to complete the laps in any of the environments.

In order to compare the experimental results, non-parametric tests of multiple comparisons have been used. Their use is recommended in those cases in which the objective is to compare the results of a new algorithm against various methods simultaneously. The Friedman test with Holm post-hoc test was selected as the method for detecting significant differences among the results. The test is performed for the *quality* indicator in table 2.7.

The statistical test (table 2.9) shows that the difference of the quality of the IQFRL approach is statistically significant. Only ν-SVR and MOGUL with sample 16 preprocessing are comparable to IQFRL, as the number of blockades is very low or null in some environments.

Additionally, table 2.10 shows the results obtained by IQFRL in two real environments. As in the previous tables, the results are the average and standard deviation over 5 laps. The distance to the wall is lower than 60 cm, showing a good behavior, although the velocity seems to be low, this is because corners are very close to each other and the robot does not have time to accelerate. Also, the velocity change reflects a very smooth movement as changes in velocity take more time in the real robot.

Finally, the IQFRL proposal was compared with the proposals presented in [80] for learning rules for the wall-following behavior. The purpose of this comparison is to check if IQFRL is competitive against other methods which use expert knowledge for sensor data preprocessing. Four different approaches were used: the COR methodology, the weighted COR methodology (WCOR), Hierarchical Systems of Weighted Linguistic Rules (HSWLR) and a local evolutionary learning of Takagi-Sugeno rules (TSK). For these approaches, four input variables were defined by an expert: right distance, left distance, velocity, and the orientation (alignment) of the robot to the wall at its right. Moreover, the granularities of each variable were also defined by the expert. Table 2.12 presents the comparison between these approaches

and the IQFRL proposal on those environments which are common.

The IQFRL approach exhibited the highest quality in the two most complex environments (office and hospital). Moreover, table 2.11 shows the non-parametric tests performed over *quality*. The Friedman p-value is higher than in table 9, due to the low number of environments available for comparisons. As can be seen, there is no statistically significant difference regarding the *quality*. That is, the controllers learned with embedded preprocessing has similar performance to the methods that use expert knowledge to preprocess the data.

2.6.4 Complexity of the Rules

An example of a rule learned by IQFRL is presented in Fig. 2.19. The antecedent part is composed of a single QFP. The linguistic value $A_d^{5,\,1}$ indicates a low distance, while $A_b^{4,\,1}$ denotes that the beams sector of the proposition is formed by the frontal and right parts of the robot. Therefore, the rule describes a situation where the robot is too close to the wall and, if it continues, it will collide. Because of that, the consequent indicates a zero linear velocity and a turn of the robot to the left, in order to get away from the wall without getting the robot into risk.

Table 2.13 shows the number of rules learned for the different situations by each of the methods based on rules. MOGUL is implemented as a multiple-input single-output (MISO) algorithm, therefore for each output, different rule bases were learned. Moreover, table 2.14 shows the complexity of the learned rules in terms of mean and standard deviation of the number of propositions and granularities for each input variable.

Table 2.11: Non-parametric test for *quality* of table 2.12.

Alg.	Ranking	Holm p-value
IQFRL	2.9	–
COR	3.9	0.01
WCOR	3.7	0.017
HSWLR	2.8	0.05
TSK	1.7	0.006
Friedman p-value = 0.19		
Holm's rejects hypothesis with p-value $<= 0.005$		

Table 2.12: Average results ($x \pm \sigma$) of IQFRL and several approaches with preprocessing based on expert knowledge [80]

Alg.	Env.	Dist.(cm)	Vel.(cm/s)	Vel.ch.(cm/s)	Time(s)	# Blockades	*quality*
IQFRL	wsc8a	56.96 ± 1.00	27.45 ± 0.84	7.70 ± 0.33	233.10 ± 5.28	0.00 ± 0.00	0.11
	rooms	57.38 ± 0.34	30.97 ± 0.34	6.38 ± 0.43	261.93 ± 4.60	0.00 ± 0.00	0.10
	autolab	52.91 ± 0.20	28.75 ± 0.31	5.57 ± 0.48	499.33 ± 9.74	0.00 ± 0.00	0.17
	office	51.37 ± 0.57	24.20 ± 0.18	6.65 ± 0.25	578.27 ± 2.92	0.00 ± 0.00	0.21
	hospital	51.09 ± 0.19	26.68 ± 0.10	6.18 ± 0.35	3608.07 ± 21.72	0.00 ± 0.00	0.23
COR	wsc8a	53.20 ± 1.33	39.86 ± 0.71	5.67 ± 0.83	174.98 ± 1.79	0.00 ± 0.00	0.17
	rooms	46.80 ± 0.59	37.82 ± 0.41	6.76 ± 0.31	227.16 ± 1.03	0.00 ± 0.00	0.16
	autolab	56.88 ± 0.91	25.69 ± 0.79	10.79 ± 0.21	587.96 ± 39.72	0.00 ± 0.00	0.09
	office	55.97 ± 1.65	32.48 ± 0.90	4.06 ± 0.28	457.58 ± 15.00	0.00 ± 0.00	0.11
	hospital	54.12 ± 0.92	35.63 ± 0.77	6.95 ± 0.28	2864.92 ± 45.27	0.00 ± 0.00	0.14
WCOR	wsc8a	52.79 ± 1.36	36.98 ± 1.85	7.37 ± 0.62	187.90 ± 9.78	0.00 ± 0.00	0.17
	rooms	51.17 ± 0.77	37.19 ± 0.27	9.15 ± 0.24	234.04 ± 2.70	0.00 ± 0.00	0.23
	autolab	52.97 ± 1.10	33.47 ± 0.89	7.12 ± 0.52	455.98 ± 41.60	0.00 ± 0.00	0.16
	office	54.59 ± 1.10	33.13 ± 0.97	6.76 ± 0.53	448.16 ± 10.36	0.00 ± 0.00	0.13
	hospital	55.26 ± 1.01	33.71 ± 0.14	6.52 ± 0.12	3073.98 ± 23.63	0.00 ± 0.00	0.12
HSWLR	wsc8a	51.42 ± 0.78	30.46 ± 1.01	3.36 ± 0.13	222.34 ± 6.09	0.00 ± 0.00	0.19
	rooms	50.09 ± 0.88	28.71 ± 0.29	3.04 ± 0.20	290.70 ± 3.66	0.00 ± 0.00	0.24
	autolab	51.50 ± 0.34	23.50 ± 0.97	3.05 ± 0.14	618.40 ± 20.98	0.00 ± 0.00	0.17
	office	53.43 ± 1.22	24.69 ± 0.66	3.73 ± 0.11	594.74 ± 13.16	0.00 ± 0.00	0.13
	hospital	54.60 ± 1.65	25.07 ± 0.49	3.89 ± 0.06	4209.68 ± 166.14	0.00 ± 0.00	0.12
TSK	wsc8a	51.43 ± 1.36	37.54 ± 1.53	5.20 ± 0.50	182.54 ± 8.35	0.00 ± 0.00	0.22
	rooms	49.07 ± 1.08	37.05 ± 0.82	4.96 ± 0.21	227.58 ± 4.46	0.00 ± 0.00	0.24
	autolab	51.87 ± 2.99	33.05 ± 1.33	4.61 ± 0.11	465.56 ± 15.33	0.00 ± 0.00	0.19
	office	53.75 ± 0.97	34.26 ± 0.65	5.24 ± 0.22	432.38 ± 10.48	0.00 ± 0.00	0.14
	hospital	54.50 ± 1.49	34.31 ± 0.32	5.01 ± 0.11	3053.74 ± 123.72	0.00 ± 0.00	0.13

The IQFRL approach is able to learn knowledge bases with a much lower number of rules than MOGUL, even though it is learning both outputs at the same time. The learning of QFRs results in a low number of propositions per rule, thus demonstrating its generalization ability, in spite of the huge input space dimensionality. Moreover, the granularities of each of the input variables are, in general, also low. Therefore, the learned knowledge bases show a low complexity without losing accuracy.

IF
$d(h)$ is $A_d^{5,\,1}$ in 50 percent of $A_b^{4,\,1}$
THEN
vlin is A_{vlin}^{1} and
vang is A_{vang}^{19}

Figure 2.19: A typical rule learned by IQFRL. $A_d^{5,\,1}$ indicates a low distance and $A_b^{4,\,1}$ indicates the frontal and right sectors.

Table 2.13: Number of rules learned

Alg.	Preproc.	Output	$\#R_{straight}$	$\#R_{convex}$	$\#R_{concave}$
IQFRL	–	Both	108.00 ± 18.88	47.80 ± 16.09	40.40 ± 10.65
MOGUL	min 16	*vlin*	548.60 ± 25.60	308.20 ± 12.12	680.20 ± 24.43
		vang	547.00 ± 16.37	302.80 ± 21.57	712.40 ± 23.79
	sample 16	*vlin*	507.80 ± 29.88	268.20 ± 12.66	664.80 ± 8.52
		vang	530.20 ± 26.48	252.80 ± 8.28	709.80 ± 34.19

Table 2.14: Complexity of the rules

Alg.	Preproc.	Dataset	Output	Propositions	g_d	g_b	g_v
IQFRL	–	Straight	Both	2.74 ± 0.94	7.02 ± 10.52	5.98 ± 5.62	6.21 ± 1.53
		Convex		2.68 ± 0.69	15.37 ± 23.59	11.22 ± 8.50	6.55 ± 1.03
		Concave		2.78 ± 1.18	3.80 ± 1.79	7.07 ± 6.86	6.16 ± 1.42
MOGUL	min 16	Straight	*vlin*	17.00 ± 0.00	24.35 ± 109.80	16.00 ± 0.00	39.44 ± 137.45
			vang	17.00 ± 0.00	24.49 ± 107.66	16.00 ± 0.00	35.19 ± 117.75
		Convex	*vlin*	17.00 ± 0.00	32.34 ± 125.75	16.00 ± 0.00	51.68 ± 172.27
			vang	17.00 ± 0.00	38.99 ± 144.86	16.00 ± 0.00	45.07 ± 146.38
		Concave	*vlin*	17.00 ± 0.00	22.93 ± 100.76	16.00 ± 0.00	32.79 ± 106.77
			vang	17.00 ± 0.00	23.35 ± 103.39	16.00 ± 0.00	37.56 ± 122.75
	sample 16	Straight	*vlin*	17.00 ± 0.00	26.23 ± 117.41	16.00 ± 0.00	33.98 ± 108.16
			vang	17.00 ± 0.00	26.25 ± 116.18	16.00 ± 0.00	37.60 ± 126.86
		Convex	*vlin*	17.00 ± 0.00	25.68 ± 103.29	16.00 ± 0.00	49.61 ± 160.18
			vang	17.00 ± 0.00	31.06 ± 119.50	16.00 ± 0.00	46.56 ± 151.27
		Concave	*vlin*	17.00 ± 0.00	23.50 ± 105.79	16.00 ± 0.00	33.62 ± 112.09
			vang	17.00 ± 0.00	23.95 ± 106.27	16.00 ± 0.00	34.63 ± 121.52

2.7 Real World Applications

Two of the most used behaviors in mobile robotics are path and object tracking. In recent years several real applications of these behaviors have been described in the literature in different realms. For instance, in [63], a tour-guide robot that can either follow a predefined route or a tour-guide person was shown. With a similar goal, an intelligent hospital service robot was presented in [104]. In this case, the robot can improve the services provided in the hospital through autonomous navigation based on following a path. More recently, in [69] a team of robots that cooperate in a building developing maintenance and surveillance tasks was presented.

More dynamic environments were described in [40, 66], where the robot had to operate in buildings and populated urban areas. These environments introduce numerous challenges to autonomous mobile robots as they are highly complex. Finally, in [45] the authors presented a motion planner that was able to generate paths taking into account the uncertainty due to controls and measurements.

In these and other real applications, the robot has to deal with static and moving objects, including the presence of people surrounding the robot, etc. All these difficulties make necessary the combination of behaviors to perform tasks like path or people tracking in real environments. In order to implement these tasks in a safe way, the robot must be endowed with the ability to avoid collisions with all the objects in the environment while implementing the tasks. These behaviors are challenging tasks that allow us to show the performance of the IQFRL-based approach in realistic conditions. The following behaviors are considered in this section, in order of increasing complexity:

(a) *Path tracking with obstacles avoidance*. In this behavior, the mobile robot must follow a path with obstacles in it. A typical application of this behavior is a tour-guide robot that has to follow a predefined tour in a museum. Although in the initial path there were no obstacles in the trajectory, the modification of the environment with new exhibitors and the presence of people make it necessary that the robot modify the predefined route, avoiding the collision with the obstacles and returning to the predefined path as quickly as possible.

(b) *Object tracking with fixed obstacles avoidance*. In this case, the robot has to follow the path of a moving object while being at a reference distance to the object. For instance, a tour-guide person being followed by a robot with extended information on a screen.

If the followed object comes too close to an obstacle, the robot must avoid the collision while maintaining the tracking behavior.

(c) *Object tracking with moving obstacle avoidance*. This behavior is a modification of the previous one, and presents a more difficult problem. In addition to the fixed obstacles avoidance, the robot has to track an object while preventing collisions with moving obstacles that are crossing between the robot and the tracked object. These moving obstacles can be persons walking around or even other mobile robots doing their own behaviors.

In order to perform these behaviors, a fusion of two different controllers has been developed. On one hand, a tracking controller [82] was used in order to follow the path or the moving object. On the other hand, the wall-following controller learned with the IQFRL algorithm was used as the collision avoidance behavior. Section 2.6.3 showed that this controller is robust and operates safely while performing the task. There were no blockades during the behavior in all the tests, neither from collisions nor from other reasons. The way in which the wall-following behavior is used in order to avoid collisions is: given an obstacle that is too close to the robot, it can be surrounded following the border of this obstacle in order to avoid a collision with it. The controller described in this paper follows the wall on its right, while for this task, the obstacle can be on both sides. This can easily be solved by a simple permutation of the laser beams depending on which side the obstacle is detected.

The wall-following behavior is only executed when the robot is too close to an object —a value of 0.4 m has been used as threshold. The objective of the controller is to drive the robot to a state in which there is no danger of collision —a value of 0.5 m has been established as a safe distance. As long as the robot is in a safe state the tracking behavior is resumed. This behavior controls the linear and angular velocities of the robot in order to place it at an objective point in every control cycle. This point is defined using the desired distance between the robot and the moving object. The tracking controller uses four different input variables:

- The distance between the robot and the objective point:

$$d = \frac{\sqrt{(x_r - x_{obj})^2 + (y_r - y_{obj})^2}}{d_{ref}} \tag{2.19}$$

where (x_r, y_r) are the coordinates of the robot, (x_{obj}, y_{obj}) are the coordinates of the objective point and d_{ref} is the reference distance between the robot and the objective point.

- The deviation of the robot with respect to the objective point:

$$dev = \arctan\left(\frac{y_{obj} - y_r}{x_{obj} - x_r}\right) - \theta_r \tag{2.20}$$

  where θ_r is the angle of the robot. A negative value of the deviation indicates that the robot is moving in a direction to the left of the objective point, while a positive value means that it is moving to the right.

- The difference of velocity between the robot and the objective point:

$$\Delta v = \frac{v_r - v_m}{v_{max}} \tag{2.21}$$

  where v_r, v_m and v_{max} are the linear velocities of the robot, the moving object, and the maximum velocity attainable by the robot.

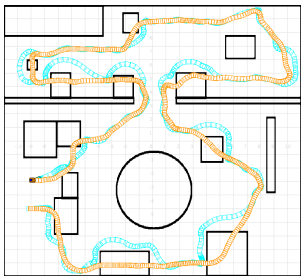
- The difference in angle between the object and the robot:

$$\Delta\theta = \theta_m - \theta_r \tag{2.22}$$

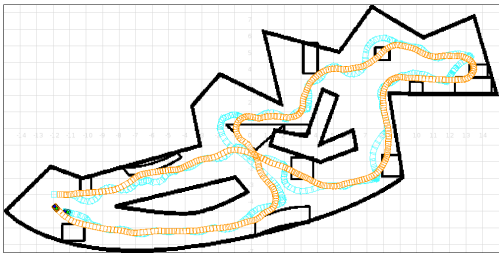
  where θ_m is the angle of the moving object.

The reference distance (d_{ref}) is different depending on the type of behavior. For the path tracking behavior, there is no moving object tracking and, therefore, the robot follows the path with $d_{ref} = 0$ in order to do a perfect path tracking. In the other two behaviors the robot follows a moving object, so it is necessary to keep a safe distance —a value of $d_{ref} = 0.5$ m was used in the experiments shown in this section.

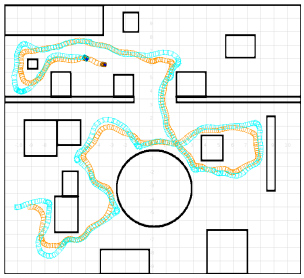
The three behaviors have been validated in two different environments (*M1* and *Domus*) which try to reproduce the plant of a museum (Fig. 2.20). Figs. 2.20(a) and 2.20(b) show the path tracking with obstacles avoidance behavior. The orange (medium grey) path represents the trajectory that has to be followed by the robot. This path also includes information of the velocity that the robot should have at each point. The higher the concentration of marks, the lower the linear velocity in that point of the path. Moreover, the path was generated without obstacles and, once the obstacles were added to the environment, the robot was placed at the beginning of the path in order to track it. The cyan (light grey) path indicates the trajectory implemented by the robot using the proposed combination of controllers (wall-following and tracking). It can be seen that the robot avoids successfully all the obstacles in its path, i.e., the wall following behavior deviates the robot from the predefined path when an obstacle



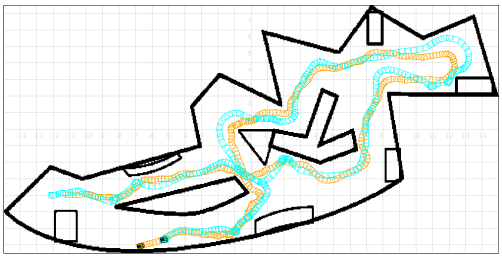
(a) *Path tracking with obstacles avoidance in M1.*



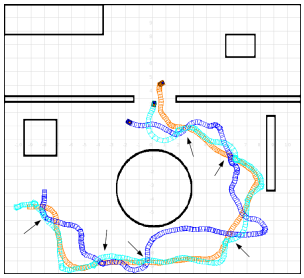
(b) *Path tracking with obstacles avoidance in Domus.*



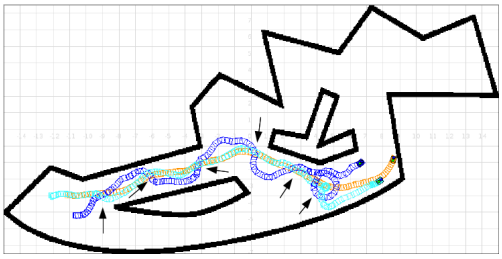
(c) *Object tracking with fixed obstacles avoidance in M1.*



(d) *Object tracking with fixed obstacles avoidance in Domus.*



(e) *Object tracking with moving obstacle avoidance in M1.*



(f) *Object tracking with moving obstacle avoidance in Domus.*

Figure 2.20: Experiments on real applications. Colors code: 1) Original path to be tracked in orange (medium grey); 2) Robot path in cyan (light grey); 3) Moving obstacle path in blue (dark grey). The arrows along the path in Figs. 20(e) and 20(f) indicate the places in which the moving obstacle interferes with the robot.

generates a possibility of collision. When the robot overcomes the obstacle, it returns to the predefined path as quickly as possible.

In the case of the moving object tracking with fixed obstacles avoidance behavior (Figs. 2.20(c) and 2.20(d)), the cyan (light grey) line represents the path of the robot due to the combination of the controllers. Also, the orange (medium grey) path shows the trajectory of the moving object tracked by the robot. In this behavior, the moving object goes too close to some obstacles in several situations, forcing the controller to execute the wall following behavior in order to avoid collisions. Moreover, the wall-following controller is also executed when the moving object turns the corners very close to the obstacles, at a distance that is unsafe for the robot.

The last and most complex behavior is moving object tracking with moving obstacle avoidance (Figs. 2.20(e) and 2.20(f)). The cyan (light grey) path shows, once again, the path followed by the robot when it tracks the moving object (orange / medium grey path) while avoiding static and moving obstacles. Also, the path followed by the moving obstacle that should be avoided by the robot is shown in blue (dark grey). The arrows along the path indicate the places in which the obstacle interferes with the robot. This behavior shows the ability of the controller learned with the IQFRL algorithm to avoid collisions, even when the moving obstacle tries to force the robot to fail: the controller can detect the situation and perform the task safely, avoiding collisions.

2.8 Conclusions

This paper describes a new algorithm which is able to learn controllers with embedded pre-processing for mobile robotics. The transformation of the low-level variables into high-level variables is done through the use of Quantified Fuzzy Propositions and Rules. Furthermore, the algorithm involves linguistic labels defined by multiple granularity without limiting the granularity levels. The algorithm was extensively tested with the wall-following behavior both in several simulated environments and on a *Pioneer 3-AT* robot in two real environments. The results were compared with some of the most well-known algorithms for learning controllers in mobile robotics. Non-parametric significance tests have been performed, showing a very good and a statistically significant performance of the IQFRL approach.

2.A IQFRL for Classification (IQFRL-C)

This section describes the modifications that are necessary to accomplish for adapting the IQFRL algorithm for classification problems.

2.A.1 Examples and Grammar

The structure of the examples used for classification is very similar to the one described in expression 2.4:

$$e^l = (d(1), \ldots, d(N_b), velocity, class) \tag{2.23}$$

where *class* represents the class of the example.

Furthermore, the consequent production (production 3) of the grammar (Fig. 2.4) must be modified to:

3. consequent $\longrightarrow F_c$

where F_c is the linguistic label of the class. The output variable (*class*) has a granularity $g_c^{\#class}$.

2.A.2 Initialization

The consequent of the rules is initialized as $F_c = A_c^{\gamma}$ where γ is the class that represents the example. Only those examples whose class is different from the default class (A_c^f) are used in the initialization of a new individual.

2.A.3 Evaluation

For each individual (rule) of the population, the following values are calculated:

- True positives (*tp*):
  - $\#tp = \left|\left\{e^l : C_l = C_j \wedge DOF_j\left(e^l\right) > 0\right\}\right|$, where C_l is the class of example e^l, C_j is the class in the consequent of the j-th rule, and $DOF_j\left(e^l\right)$ is the *DOF* of the j-th rule for the example e^l. *#tp* represents the number of examples that have been correctly classified by the rule.
  - $tpd = \sum_l DOF_j\left(e^l\right) : C_l = C_j$, i.e., the sum of the *DOF*s of the examples contributing to *#tp*.

  - $tp = \#tp + tpd/\#tp$

- False positives (fp):
  - $\#fp = \left|\left\{e^l : C_l \neq C_j \wedge DOF_j\left(e^l\right) > 0\right\}\right|$: number of patterns that have been classified by the rule but belong to a different class.
  - $fpd = \sum_l DOF_j\left(e^l\right) : C_l \neq C_j$, i.e., the sum of the DOFs of the patterns that contribute to $\#fp$.
  - $fp = \#fp + fpd/\#fp$
- False negatives (fn):
  - $\#fn = n_{ex}^{C_j} - \#tp$, where $n_{ex}^{C_j} = \left|\left\{e^l : C_l = C_j\right\}\right|$. $\#fn$ is the number of examples that have not been classified by the rule but belong to the class in the consequent of the rule.

The values of tp and fp take into account not only the number of examples that are correctly/incorrectly classified, but also the degree of fulfillment of the rule for each of the examples. In case that $tpd \approx 0$, then $tp \approx \#tp$, while if it is high ($tpd \approx \#tp$) then $tp \approx \#tp + 1$. Taking into account these definitions, the accuracy of an individual of the population can be described as:

$$confidence = \frac{1}{10^{fp}} \tag{2.24}$$

while the ability of generalization of a rule is calculated as:

$$support = \frac{tp}{tp + \#fn} \tag{2.25}$$

Finally, *fitness* is defined as the combination of both values:

$$fitness = confidence \cdot support \tag{2.26}$$

which represents the strength of an individual.

2.A.4 Mutation

For classification, the probability that an example matches the output associated to a rule (Eq. 2.7) is binary. Therefore, in order to select the example (e^{sel}) that is going to be used for mutation, the following criteria is used:

Table 2.15: Number of rules learned for dataset by IQFRL-C

$\#R_{straight}$	$\#R_{convex}$	$\#R_{concave}$
–	21.20 ± 4.35	10.00 ± 1.41

- For generalization, the probability for an example e^l to be selected is:

$$P(e^l = e^{sel}) = 1 \; - \; \frac{\sum_j DOF_j\left(e^l\right) \cdot confidence_j}{\sum_j DOF_j\left(e^l\right)} \tag{2.27}$$

  where $confidence_j$ is the *confidence* (Eq. 2.24) of the j-th individual. This probability measures the accuracy with which the individuals of the population cover the example e^l.

- For specialization, the mutated individual uncovers the example e^{sel}. The probability to select e^l for specialization is calculated as follow:

$$P(e^l = e^{sel}) = 1 \; - \; DOF_j\left(e^l\right) \tag{2.28}$$

Finally, the consequent is mutated considering the class of the examples covered by the individual. Thus, the probability that the consequent of the individual j change to the class C_γ is defined as:

$$P\left(j \mid C_\gamma\right) = \frac{\sum_l DOF_j\left(e^l\right) \; : \; C_l = C_\gamma}{\sum_l DOF_j\left(e^l\right)} \tag{2.29}$$

2.A.5 Performance

The parameters used for IQFRL-C are the same as for regression (Sec. 2.6.2). Moreover, the default class is straight wall. Tables 2.15 and 2.16 show the number of rules learned by the classification method IQFRL-C and the complexity of the rules learned in terms of mean and standard deviation of the number of propositions and granularities for each input variable. The number of rules for each situation is very low, resulting in very interpretable knowledge bases. Furthermore, the complexity of the rules is also low, as the number of propositions and granularities learned show that the rules are very general.

Table 2.17 shows the confusion matrix for the learned classifier. The matrix was obtained as the average of a 5-fold cross-validation over the sets. Moreover, the performance of the

Table 2.16: Complexity of the rules learned by IQFRL-C

Propositions	g_d	g_b	g_v
2.67 ± 0.90	7.58 ± 12.65	8.41 ± 8.49	5.83 ± 1.65

Table 2.17: Confusion matrix for the classifier

Actual/Predicted	Straight	Convex	Concave
Straight	30.85	2.40	0.23
Convex	0.70	30.97	0.00
Concave	0.23	0.06	34.55
$Accuracy = 0.96$			
Cohen's $\kappa = 0.94$			

classifier was analyzed with the accuracy and the Cohen's κ [14]. Both measures are very close to 1, showing the high performance of the classifier obtained with IQFRL-C.

Chapter 3

FRULER: Fuzzy Rule Learning through Evolution for Regression

In this chapter the work is focused on the application of GFSs to regression problems in a general manner. In this case, no knowledge about the nature of the input variables is considered. TSK fuzzy system is used as the FRBS model in this phase of the thesis. The use of TSK fuzzy systems is widely extended in regression problems due to the precision of the obtained models. Moreover, Pittsburgh approach is followed in order to improve the ability to obtain low complex models, as it can control the number of rules. Low complex TSK models are good choices in many real problems due to the easy understanding of the relationship between the output and input variables.

This chapter presents FRULER (Fuzzy RUle Learning through Evolution for Regression), a new GFS algorithm for obtaining accurate and simple linguistic TSK-1 fuzzy rule base models to solve regression problems. The simplicity of the fuzzy system aims to improve both the generalization ability and the readability of the model. For that, FRULER generates linguistic fuzzy partitions with few labels, a low number of rules, and regularizes the consequents —which reduces the number of input variables that contribute to the output. The algorithm consists of three stages: instance selection, multi-granularity fuzzy discretization of the input variables, and the evolutionary learning of the rule base that uses the Elastic Net regularization to obtain the consequents of the rules.

FRULER was validated using 28 real-world datasets and the results were compared with three state of the art GFSs, which comprise both Mamdani and TSK linguistic and approxi-

mative fuzzy systems. Experimental results show that FRULER achieves the most accurate and simple models compared even with approximative approaches.

In this chapter, a full copy of the following publication is presented:

> I. Rodríguez-Fdez[1], M. Mucientes[1], and A. Bugarín[1]. FRULER: Fuzzy Rule Learning through Evolution for Regression. *Information Sciences*, Elsevier, No. 354, pp. 1-18, 2015.

3.1 Abstract

The use of Takagi-Sugeno-Kang (TSK) fuzzy systems in regression problems is widely extended due to the precision of the obtained models. Moreover, the use of simple linear TSK models is usually referred as a good choice in many real problems since it provides a straightforward functional relationship between the output and input variables. In this paper we present FRULER (Fuzzy RUle Learning through Evolution for Regression), a new genetic fuzzy system for automatically learning accurate and simple linguistic TSK fuzzy rule bases for regression problems. FRULER achieves a low complexity of the learned models while keeping a high accuracy, by following three stages: instance selection, multi-granularity fuzzy discretization of the input variables, and evolutionary learning of the rule base combined with Elastic Net regularization to obtain the consequents of the rules. Each of these stages was validated using 28 real-world datasets. FRULER was also experimentally compared with three state of the art genetic fuzzy systems, showing the most accurate and simple models even when compared with approximative approaches.

3.2 Introduction

Predictive models usually have two complementary requirements: accuracy and interpretability [7, 6]. On the one hand, accuracy indicates the ability of the model to predict values close to the real ones. On the other hand, interpretability refers to the easiness with which people understand the model [7]. A number of predictive models involving fuzzy rule-based systems have aimed to combine the interpretability and expressiveness of the rules with the ability

[1]Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain.

of fuzzy logic for representing uncertainty [20, 21]. In this regard, interpretability for fuzzy systems involves two main issues [7]:

- Readability: it is related to the simplicity of the structure of the fuzzy system itself, i.e., the number of variables, linguistic terms per variable, fuzzy rules, premises per rule, etc. It represents the quantitative or objective part of the model interpretability.
- Comprehensibility: it is determined by the general semantics of the fuzzy system and the fuzzy inference mechanism. It is associated with the meaning of the fuzzy partitions for the user. It represents the qualitative or subjective part of the interpretability.

In fuzzy systems, the most important aspect to be considered in terms of interpretability is the definition of the fuzzy partition for each variable, also called the data base definition. Two different approaches can be used for this definition: (i) the linguistic approach, in which all rules share the same fuzzy partition for each variable; (ii) the approximative approach, which may have a different definition of the fuzzy labels for each rule in the rule base. The former implies more interpretability through a higher simplicity and comprehensibility, while the latter usually obtains more accurate solutions. However, approximative approaches can lead to complex partitions of the input space, which makes harder to understand and capture the insights of the relationship between inputs and outputs. Moreover, strong linguistic partitions are labelled as the most interpretable choice in this regard, since they fulfil all the relevant semantic constraints such as distinguishability, coverage, normality, convexity and others [7].

In the case of fuzzy models for regression problems, two different approaches have been proposed in the literature: Mamdani fuzzy systems (where both antecedent and consequent are represented by fuzzy sets) and Takagi-Sugeno-Kang (TSK) fuzzy systems [108, 106], where the antecedents are represented by fuzzy sets, while the consequents are a weighted combination of the input variables. Although Mamdani systems are well-known because of their interpretability, the linear model in TSK rules is also a good choice since it is straightforward to understand the relationship between the output and input variables. This is of particular interest in many fields, such as robotics [93, 90, 85], medical imaging [91], industrial estimation [86] and optimization of processes [112].

One of the most widely used learning algorithms for automatic building of fuzzy rule bases are Genetic Fuzzy Systems (GFSs) [26], i.e., the combination of evolutionary algorithms and fuzzy logic. Evolutionary algorithms are appropriate for learning fuzzy rules due to their flexibility —that allows them to codify any part of the fuzzy rule base system—, and due to their capability for managing the balance between accuracy and simplicity of the model. In par-

ticular, recent developments using multi-objective evolutionary fuzzy systems can be found in [3, 39, 98, 8], where both Mamdani and TSK systems were proposed to solve large-scale regression problems. Moreover, in [76] an adaptive fuzzy inference system was proposed to cope with high-dimensional problems.

The simplicity of the models obtained by GFSs for regression has been mostly achieved in the literature through keeping the number of rules and/or the number of labels in the rule base as low as possible using multi-objective approaches [33, 54]. More recently, the use of instance selection techniques has received more attention in both classification [41, 34] and regression [92] problems. This approach faces two problems at once: decreases the complexity for large-scale problems and reduces the overfitting, as the rules can be generated with a part of the training data and the error of the rule base can be estimated with another part (or the whole) training set. Moreover, when no expert knowledge is available to determine the fuzzy labels, two different approaches can be applied: uniform discretization combined with lateral displacements [2], or non-uniform discretization [55]. Recently, [32, 42] have shown the application of non-uniform discretization techniques to classification problems.

The use of TSK fuzzy rule bases implies another complexity dimension: the polynomial in the consequent —usually with degree 1 (TSK-1) or 0 (TSK-0). The most widely used approach for learning the coefficients of the polynomial is the least squares method. However, this choice often leads to models that overfit the training data and misbehave in test. This problem can be solved by shrinking (Ridge regularization) or setting some coefficients to zero (Lasso regularization), thus obtaining simpler models. Moreover, a combination of both regularizations, called Elastic Net [119] can be used.

In this paper we present FRULER (Fuzzy RUle Learning through Evolution for Regression), a new GFS algorithm for obtaining accurate and simple linguistic TSK-1 fuzzy rule base models to solve regression problems. The simplicity of the fuzzy system aims to improve the readability of the model —and, therefore, its interpretability— by obtaining linguistic fuzzy partitions with a low number of labels, a low number of rules, and through the regularization of the consequents —which reduces the number of input variables that contribute to the output. The main contributions of this work are: i) a new instance selection method for regression, ii) a novel multi-granularity fuzzy discretization of the input variables, in order to obtain non-uniform fuzzy partitions with different degrees of granularity, iii) an evolutionary algorithm that uses a fast and scalable method with Elastic Net regularization to generate accurate and simple TSK-1 fuzzy rules.

This work is structured as follows. Section 3.3 defines the TSK model used by FRULER. Section 3.4 describes the different stages of the GFS: the instance selection method, the discretization approach, and the evolutionary algorithm. Sec. 3.5 shows the results of the approach in 28 regression problems, and the comparison with other proposals involving statistical tests. Finally, the conclusions are presented in Sec. 3.6.

3.3 Takagi-Sugeno-Kang fuzzy rule systems

Takagi, Sugeno, and Kang proposed in [106, 108] a fuzzy rule model in which the antecedents are comprised of linguistic variables, as in the case of Mamdani systems [73, 74], but the consequent is represented as a polynomial function of the input variables. These type of rules are called TSK fuzzy rules. The most common function for the consequent of a TSK rule is a linear combination of the input variables (TSK-1), and its structure is as follows:

$$\begin{aligned} &\text{If } X_1 \text{ is } A_1 \text{ and } X_2 \text{ is } A_2 \text{ and } \ldots \text{ and } X_p \text{ is } A_p \text{ then} \\ &Y = \beta_0 + X_1 \cdot \beta_1 + X_2 \cdot \beta_2 + \cdots + X_p \cdot \beta_p \end{aligned} \tag{3.1}$$

where X_j represents the j-th input variable, p the number of input variables, A_j is the linguistic fuzzy term for X_j, Y is the output variable, and β_j is the coefficient associated with X_j in the consequent part of the rule.

The matching degree h between the antecedent of the rule r_k and the current inputs to the system $(x_1, x_2, \ldots, x_p)$ is calculated as:

$$h_k = T(A_1^k(x_1), A_2^k(x_2), \ldots, A_p^k(x_p)) \tag{3.2}$$

where A_j^k is the linguistic fuzzy term for the j-th input variable in the k-th rule and T is the t-norm conjunctive operator, usually the minimum function. The final output of a TSK fuzzy rule base system composed by m TSK fuzzy rules is computed as the average of the individual rule outputs Y_k weighted by the matching degree:

$$\hat{y} = \frac{\sum_{k=1}^{m} h_k \cdot Y_k}{\sum_{k=1}^{m} h_k} \tag{3.3}$$

Linguistic TSK fuzzy rule systems represent a good trade-off between accuracy and interpretability, since:

- The use of linguistic terms in the antecedent of the rules provides a full description of the input space due to the shared definition of the fuzzy partitions in the data base of the system.

- The linear representation of the output allows the system to obtain accurate solutions using different well-studied statistical methods.
- The consequent of the rules represented by a linear combination of the input variables facilitates the understanding of the relationship between the inputs and the output.

Thus, even if the TSK fuzzy rule systems are less comprehensible in natural language terms than Mamdani approaches, the system can provide useful and understandable information, and is the preferable choice in some domains. In this article we focus on developing simple and accurate TSK fuzzy rule models based on a linguistic representation of the antecedents.

3.4 FRULER description

This section presents the three main components of FRULER: a two-stage preprocessing — formed by the instance selection and multi-granularity fuzzy discretization modules—, and a genetic algorithm, which contains an ad-hoc TSK-1 (first order) rule generation module (Fig. 3.1). Both preprocessing techniques are executed to improve the simplicity of the fuzzy rule bases obtained by the evolutionary algorithm. On the one hand, instance selection reduces the variance of the models focusing the generated rules on the representative examples. On the other hand, multi-granularity fuzzy discretization decreases the complexity of the fuzzy partitions, thus making unnecessary to establish an upper bound in the number of rules in the evolutionary stage.

The evolutionary learning process obtains a definition of the data base for each knowledge system. Then a novel ad-hoc TSK-1 rule generation module calculates the antecedents and consequents of each rule using only the representative examples. Finally, each knowledge base is evaluated using the full training dataset.

3.4.1 Instance Selection for Regression

The instance selection method for regression is an improvement of the CCISR (Class Conditional Instance Selection for Regression) algorithm [92], which is an adaptation for regression of the instance selection method for classification CCIS (Class Conditional Instance Selection) [75]. The main differences between the FRULER instance selection process and CCISR are:

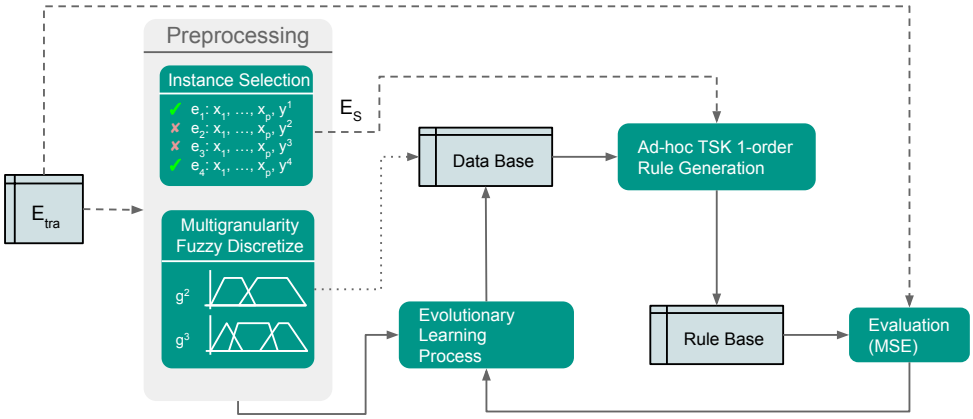


Figure 3.1: FRULER architecture showing each of the three separated stages. Dashed lines indicate flow of datasets, dotted lines multigranularity information and solid lines represent process flow.

- The error measure is based on the $1-$*nearest neighbor* ($1NN$) approach for regression, thus reducing the complexity of the calculations compared with CCISR, which uses an ad-hoc fuzzy system to evaluate the instances.
- The stopping criteria is more flexible, allowing more iterations without improvement until the termination of the process.
- The size of the initial set of selected examples is also different due to the previous improvements.

The instance selection process is based on a relation called class conditional nearest neighbor (*ccnn*) [75], defined on pairs of points from a labeled training set as follows: for a given class c, *ccnn* associates to instance a its *nearest neighbor* computed among only those instances (excluded a) in class c. This relation, therefore, describes proximity information conditioned to a class label.

In regression problems, the outputs are real values instead of labels and, therefore, they must be discretized in order to use the *ccnn* relation. Traditionally, an unsupervised discretization process needs the definition of either the number of intervals or their shape [29]. In FRULER, the shape of the intervals takes the output density into account, i.e., the intervals are selected such that they represent dense clusters. In other words, the split points between intervals are selected in the zones where the output density is locally minimum.

We use Kernel Density Estimation (KDE) with a gaussian kernel in order to estimate the probability density function of the output variable (y) in a non-parametric way. In order to select the appropriate kernel bandwidth, Scott's rule is applied. [100]. Once the probability density function is obtained, the local minimum determines the split points, and, therefore, which labels/classes are used for the *ccnn* relation. Thus, each instance is associated with one of the labels obtained by this process, and the instance selection method can follow the CCIS procedure.

Two different graphs can be constructed using this relation, as proposed in CCIS:

- Within-class directed graph (G_{wc}): consists of a graph where each instance has an edge pointing to the nearest instance of the same class.
- Between-class directed graph (G_{bc}): is a graph where each instance has an edge pointing to the nearest instance of any different class.

These graphs are used to define an instance scoring function by means of a directed information-theoretic measure (the K-divergence) applied to the in-degree distributions of these graphs. The `Score` scoring function is defined as:

$$Score(e^i) = p_w^i \cdot log\left(\frac{p_w^i}{(p_w^i + p_b^i)/2}\right) - p_b^i \cdot log\left(\frac{p_b^i}{(p_w^i + p_b^i)/2}\right) \tag{3.4}$$

where e^i is the example considered, p_w^i is the in-degree of e^i in G_{wc} divided by the total in-degree of G_{wc}, and p_b^i is the inner degree of e^i in G_{bc} divided by the total in-degree of G_{bc}. This scoring function is used to develop an effective large margin instance selection method, called Class Conditional selection (Fig. 3.2).

The instance selection algorithm starts from a set of training examples E. The method uses the *leave-one-out* mean squared error (MSE) with $1NN$ (this error is called ε) in order to estimate the information loss. Thus, although the scoring function and the graphs are based on the labels obtained by KDE, the error measure is based on the original regression problem.

First, an initial core of instances from E is selected, sorted by `Score` (Fig. 3.2, lines 1-2). The size of this initial set is:

$$k_0 = max\left(c, \left\lceil \frac{\varepsilon^E \cdot |E|}{max(y) - min(y)} \right\rceil\right) \tag{3.5}$$

where c is the number of classes obtained from KDE and ε^E is the error using the set of examples in E. This choice is motivated by (i) at least one example for each class is considered,

1: $E = \{e^1, \ldots, e^n\}$ sorted in decreasing order of `Score` (Eq. 3.4)
2: $S = \{e^1, \ldots, e^{k_0}\}$
3: $it_{wi} = 0$
4: **repeat**
5: $\quad Temp = S \cup \{e^l\}$
6: $\quad$ **if** $\varepsilon^{Temp} < \varepsilon^S$ **then**
7: $\quad\quad it_{wi} = 0$
8: $\quad\quad S_{best} = Temp$
9: $\quad$ **else**
10: $\quad\quad it_{wi} = it_{wi} + 1$
11: $\quad S = Temp$
12: **until** $E = S \vee it_{wi} > \sqrt{|E|/|S|}$
13: return S_{best}

Figure 3.2: Pseudocode of Class Conditional selection [75].

and (ii) the error in the second part can be interpreted as the miss-classification probability divided by the range of the output $max(y) - min(y)$. Thus, the second part indicates that at least the miss-classified examples must be selected in order to be correctly classified. After this, the instance selection method iteratively selects instances and adds them to the set S (lines 4-12), choosing in the first place those with the highest score. The process terminates when all the examples of E are in S or when it_{wi} —the number of consecutive iterations for which the empirical error (ε^S) increases— is greater than $\sqrt{|E|/|S|}$ (line 12). This threshold allows more iterations without improvement at the beginning of the selection process, when the error is more sensitive, and stops earlier when the number of selected instances is high.

In order to further improve the number of selected instances, CCIS uses the Thin-out post-processing (Fig. 3.3). This algorithm selects points close to the decision boundary of the 1NN rule. This is achieved by selecting instances having positive in-degree in the between-class graph set S (G^S_{bc}) and storing them in S_f. Then, an iterative process is performed as follows: points having positive in-degree in the $G^{S_1}_{bc}$ are added to S_f in case they were not "isolated" in the previous iteration, that is, if their in-degree was non-null (line 6). This iterative process terminates when the empirical error increases (line 7).

1: $S_f = \{e^l \in S$ with in-degree in $G_{bc}^S > 0\}$
2: $S_{prev} = S$
3: $S_1 = S \setminus S_f$
4: $go\_on = true$
5: **while** go_on **do**
6: $\quad S_t = \{e \in S_1$ with in-degree in $G_{bc}^{S_1} > 0$ and with in-degree in $G_{bc}^{S_{prev}} > 0\}$
7: $\quad go\_on = \varepsilon^{S_f \cup S_t} < \varepsilon^{S_f}$
8: $\quad$ **if** go_on **then**
9: $\quad\quad S_f = S_f \cup S_t$
10: $\quad\quad S_{prev} = S_1$
11: $\quad\quad S_1 = S \setminus S_f$
12: return S_f

Figure 3.3: Pseudocode of Thin-out selection [75].

3.4.2 Multi-granularity Fuzzy Discretization for Regression

The definition of the fuzzy partition of each input variable is a critical step in the design of TSK fuzzy rule bases. When no knowledge is available, the set of fuzzy labels for a variable is automatically obtained through fuzzy discretization. Moreover, if the number of labels is unknown, then a multi-granularity approach may be used. In a multi-granularity proposal, each regarded granularity has a different fuzzy partition. Specifically, a granularity g_{var}^i divides the variable *var* in i fuzzy labels, i.e., $g_{var}^i = \{A_{var}^{i,1}, \ldots, A_{var}^{i,i}\}$.

The generation of the fuzzy linguistic labels can be divided into two stages. First, the variable must be discretized to obtain a set of split points C^g for each granularity g. Then, given the split points, the fuzzy labels can be defined for each granularity. In a top-down approach, the split points are searched iteratively, i.e., only a new split point is added at each step, obtaining two new intervals. Therefore, the approach proposed in this work aims to preserve interpretability between contiguous granularities: adding a new label to the previous granularity and modifying the flanks of the adjacent labels (Fig. 3.4). In regression problems (TSK-1 in our case), the discretization process must search for the split point that minimizes the error when a linear model is applied to each of the resulting intervals.

In order to select the maximum number of split points for a variable, we used the well-

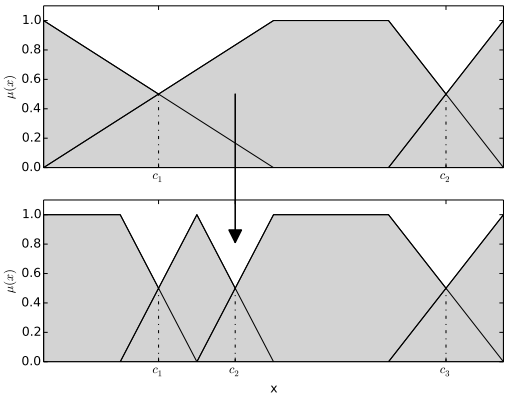


Figure 3.4: Top-down approach for the multi-granularity discretization. Only one label is divided into two new labels in order to obtain the next granularity.

known Bayesian Information Criterion (BIC). This criterion takes two issues into account: the error produced when applying the model to the data and its complexity. In our case, the error is obtained from the summation of the mean squared error (*MSE*) of a least squares model fitted for each interval of the discretization. The complexity of the model is determined by the number of parameters, in this case the number of inner splits and the parameters fitted by each regression applied in each interval.

The pseudocode of the discretization method for a variable is shown in Fig. 3.5. First, the split points for granularity 1 are initialized using the domain limits (line 2). The BIC measure for this first granularity is calculated (line 3) using *MSE*, a function that gets a set of examples X, learns a linear regression model using least squares and, finally, calculates the mean squared error of the model. In this case, the number of parameters is two, corresponding to the coefficients of the linear model. After that, an iterative process is executed: at each step, the split points of a new granularity are defined adding a new split point to the previous granularity (lines 5-16).

In order to obtain the split point for the new granularity, first, the best split point (c_i) for each interval between the split points of the previous granularity ($[C_i^g, G_{i+1}^g]$) is obtained (line 6). The best split point is defined as the point that obtains the global minimum of the function `LinearError` (Fig. 3.6) in an interval (Fig. 3.5, line 7). `LinearError` gets a set of

1: $g = 1$
2: $C^g = \{min(X), max(X)\}$
3: $BIC^g = |X| \cdot log(MSE(X)) + 2 \cdot log(|X|)$
4: $it_{wi} = 0$
5: **repeat**
6: $C = \{c_i \mid c_i = \text{argmin}_c$ LINEARERROR$(\{x \in X : C_i^g < x < C_{i+1}^g\}, c), \forall i = 0, ..., g, \forall c \in [C_i^g, C_{i+1}^g]\}$
7: $i_{min} = \text{argmin}_i$ LINEARERROR$(\{x \in X : C_i^g < x < C_{i+1}^g\}, c_i), \forall c_i \in C$
8: $C^{g+1} = C^g \cup \{c_{i_{min}}\}$
9: $g = g + 1$
10: $BIC^g = |X| \cdot log(\sum_{i=0}^{g} MSE(\{x \in X : C_i^g < x < C_{i+1}^g\}) + (|C^g| - 2) \cdot 2 \cdot log(|X|)$
11: **if** $BIC^g < BIC^{min}$ **then**
12: $it_{wi} = 0$
13: $min = g$
14: **else**
15: $it_{wi} = it_{wi} + 1$
16: **until** $it_{wi} > \sqrt{\frac{|X|}{30}/min}$
17: **return** $C^1, \ldots, C^{min}$

Figure 3.5: Pseudocode of the discretization method.

examples X and a split point c and calculates the total squared error (SE) of X, considering the corresponding linear regression models at each side of the split point. Only split points that obtain intervals with size of at least 30 are taken into account to assure that the obtained linear regressions are statistically valid.

The selected split point is added to the new granularity split points (lines 8-9), and the BIC measure is calculated (line 10). The number of parameters used for the BIC measure is 2 (coefficients of the linear regression for a single variable) for each interval. The number of intervals is calculated as $|C^g| - 2$, where 2 is subtracted to disregard the split points at the end of the domain of variable X.

Finally, when the number of consecutive iterations without improvement in the BIC value (it_{wi}) is greater than $\sqrt{\frac{|X|}{30}/min}$, the algorithm stops (lines 11-16). This criterion ensures that at

1: **function** LINEARERROR(X, c)
2: $X_l = \{x \in X : x < c\}$
3: $X_r = \{x \in X : x > c\}$
4: **return** $SE(X_l) \cdot \frac{|X_l|}{|X|} + SE(X_r) \cdot \frac{|X_r|}{|X|}$

Figure 3.6: Pseudocode of the function to be minimized by the discretization method.

the beginning of the discretization process —the granularity is low—, the BIC may worsen for more iterations, while with larger granularities, the algorithm becomes stricter in the stopping criterion. The number of data points is divided by 30 in order to get the maximum number of intervals.

After obtaining the discretization of the variable for each granularity, the method proposed in [55] is applied for each C^g —set of split points for the granularity g— in order to get the multi-granularity fuzzy partitions. This method uses a fuzziness parameter: fuzziness 0 indicates crisp intervals, while fuzziness 1 indicates the selection of a fuzzy set with the smallest kernel —set of points with membership equal to 1.

3.4.3 Evolutionary Algorithm

The evolutionary algorithm learns a linguistic TSK model. The integration of the evolutionary algorithm with the preprocessing stage is as follows (Fig. 3.1):

- First, the instance selection process is executed over the training examples E_{tra} in order to obtain a subset of representative examples E_S.
- Then, the multi-granularity fuzzy discretization process obtains the fuzzy partitions for each input variable.
- Finally, the evolutionary algorithm searches for the best data base configuration using the obtained fuzzy partitions, generates the entire linguistic TSK rule base using E_S and evaluates the different rule bases using E_{tra}.

Figure 3.7 shows the evolutionary learning process and how it uses the fuzzy partitions and the training examples. In what follows, we describe in detail the different components of the evolutionary algorithm.

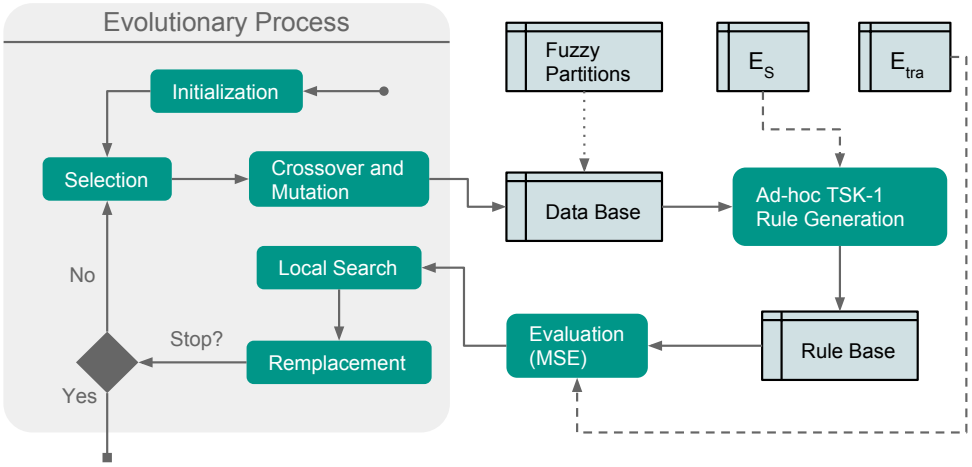


Figure 3.7: The Evolutionary learning process used in FRULER. Dashed lines indicate flow of datasets, dotted lines are for multigranularity information and solid lines represent process flow.

Chromosome Codification

The chromosome codification represents the parameters needed to create the data base and the rule base. Each individual has to codify a single fuzzy partition for each input variable from the fuzzy partitions obtained in the multi-granularity fuzzy discretization (Sec. 3.4.2). Moreover, the individuals also use the 2-tuple representation of the labels [52]. This approach applies a displacement of a linguistic term within an interval that expresses the movement of a label between its two adjacent labels. In our case, a different displacement is going to be applied to each of the split points.

Thus, the chromosome is codified with a double coding scheme ($C = C_1 + C_2$):

- C_1 represents the granularity of each input variable. It is codified with a vector of p integers:

$$C_1 = (g_1, g_2, \ldots, g_p) \tag{3.6}$$

  where g_i is the granularity of input variable i. When the granularity of a variable is equal to 1, then it is not used in the antecedent part. However, this variable can still be part of the consequent, since it may be relevant for calculating the output.

- C_2 represents the lateral displacements of the split points of the input variables fuzzy

partitions. Thus, the length of C_2 depends on the granularity of each input variable: $|C_2| = \sum_{j=1}^{p} (|g_j| - 1), \forall g_j \in C_1$:

$$C_2 = (\alpha_1^1, \ldots, \alpha_1^{g_1-1}, \ldots, \alpha_p^1, \ldots, \alpha_p^{g_p-1}) \tag{3.7}$$

where α_i^j is the lateral displacement of the j split point of variable i. Each lateral displacement can vary in the $(0.5, 0.5)$ interval, which represents half of the distance between each split point (Fig. 3.8). An example of a lateral displacement can be seen in Figure 3.9. The fuzzy partitions are always strong —the sum of the degree of fulfillment for each point of the domain is always equal to 1— and, therefore, interpretability is maintained.

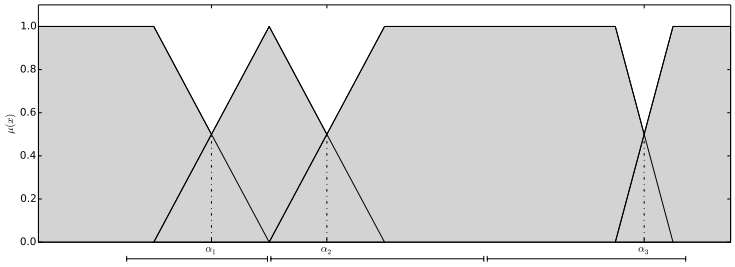


Figure 3.8: An example of lateral displacement intervals for limits equal to $(0.5, 0.5)$. The split points can move a maximum of half of the distance to the next split point.

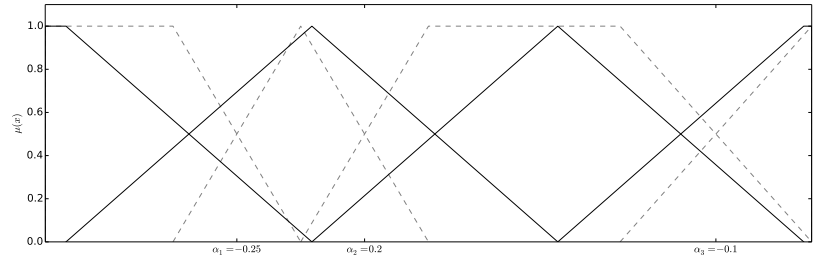


Figure 3.9: A lateral displacement example. The dashed lines indicate the original fuzzy partition, while the solid lines indicate the obtained partition after the displacement has been applied.

Initialization

The initial pool of individuals is generated by a combination of two initialization procedures. A half of the individuals are generated with the same random granularity for each variable, while the other half is created with a different random granularity for each variable. The lateral displacements are initialized to 0 in all cases.

After that, when the product of the granularities indicated in C_1 (i.e., the maximum number of rules that can be obtained) is greater than the number of input variables times the highest maximum granularity of the variables, then a variable is randomly selected to be removed from the antecedent part —its granularity is set to 1— until the previous condition is satisfied. This is done in order to avoid too complex solutions in the initialization stage —during the evolutionary learning this upper bound to the number of rules does not apply.

TSK Rule Base Generation

An ad-hoc method is used to construct the rule base from the data base codified in the chromosome, i.e. the fuzzy partitions indicated in C_1 after applying the displacement in C_2. The Wang & Mendel algorithm [113] is used to create the antecedent part of the rule base for each individual. The method is quick and simple, and obtains a representative rule base given the definition of the data base and a set of examples.

The consequent part of the rules is learned using the Elastic Net method [119] in order to obtain the coefficients of the degree 1 polynomial for each rule. Elastic Net linearly combines the ℓ_1 (Lasso regularization) and ℓ_2 (Ridge regularization) penalties of the Lasso and Ridge methods to overcome some of their limitations. This combination allows to obtain sparse models —forces variables with low or null correlation with the output to have coefficients equal to 0— while learning a smooth linear regression —the coefficients are shrunk towards 0.

Elastic Net obtains the coefficients of a linear regression minimizing the following equation:

$$\hat{\beta} = \arg\min_{\beta} ||Y - X \cdot \beta||_2^2 + \lambda \cdot \alpha \cdot \|\beta\|_2^2 + \lambda \cdot (1-\alpha) \cdot \|\beta\|_1 \tag{3.8}$$

where β is the coefficients vector $(\beta_0, \beta_1, \ldots, \beta_p)$, Y is the outputs vector $(y^1, \ldots, y^n)$, X is the inputs matrix with size $n \times p$ —rows represent examples while columns are the input variables—, λ is the regularization parameter and α represents the trade-off between ℓ_1 and ℓ_2 penalization.

In order to use Elastic Net for learning the consequents, the coefficients for each rule cannot be calculated separately due to the aggregation function used to obtain the output of the system (Eq. 3.3). Therefore, all the coefficients must be optimized at the same time, taking into account the degree of fulfillment of each rule (Eq. 3.2) for each input vector x^i. Thus, the matrix X is modified as follows:

- The normalized degree of fulfillment for each rule r_k and each example e^i is calculated as:

$$z_k^i = \frac{h_k(x^i)}{\sum_{u=1}^{m} h_u(x^i)} \tag{3.9}$$

  where the denominator is the normalization term for each input vector x^i, i.e., the summation of the degree of fulfillment of all rules.

- Then, the matrix X is defined as:

$$X = \begin{pmatrix} z_1^1, x_1^1 \cdot z_1^1, \ldots, x_p^1 \cdot z_1^1, \ldots, z_m^1, x_1^1 \cdot z_m^1, \ldots, x_p^1 \cdot z_m^1 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ z_1^n, x_1^n \cdot z_1^n, \ldots, x_p^n \cdot z_1^n, \ldots, z_m^n, x_1^n \cdot z_m^n, \ldots, x_p^n \cdot z_m^n \end{pmatrix} \tag{3.10}$$

  where each row replicates the input vector $x^i = (1, x_1^i, x_2^i, \ldots, x_p^i)$ —a 1 was added to take into account the independent term— as many times as the number of rules (m), weighting each rule r_k by z_k^i.

- Finally, the coefficient vector is the concatenation of the coefficients of all rules:

$$\beta = \left(\beta_0^1, \beta_1^1, \ldots, \beta_p^1, \ldots, \beta_0^m, \beta_1^m, \ldots, \beta_p^m\right) \tag{3.11}$$

In order to solve the minimization problem of Elastic Net (Eq. 3.8), the Stochastic Gradient Descent (SGD) optimization technique was used [17, 110]. This gradient descent method is characterized by updating each coefficient separately using only one example at a time. This is particularly suited for sparse datasets, which is a common case when X is constructed using Eq. 3.10 —z_k^i is 0 when a rule does not cover an example.

The pseudocode of the method is shown in Figure 3.10. SGD needs three parameters to solve the Elastic Net approach: the regularization (λ), the trade-off between Lasso and Ridge (α) and the initial learning rate (η^0). On the one hand, α usually takes a low value in order to behave like ℓ_1 but with the shrinkage of ℓ_2 in the features with coefficient not equal to 0. On the other hand, λ and η^0 can be obtained using a grid search —testing a set of possible

values in a predefined interval— using only a small subset of examples since the convergence properties are maintained [17].

The algorithm is composed of three different loops: i) lines 3-27, which represent an iteration over the whole dataset, ii) lines 6-17, which iterate over each example and iii) lines 11-17, which iterate over the coefficients. Note that, in this case, the number of coefficients is the number of columns in X ($p \cdot m$), i.e., the number of input variables of the problem (p) times the number of rules (m). First, the examples are shuffled (line 5) each time the whole dataset is used. Then, for each example e^i, t is incremented by 1 and the learning rate (η^t) and the shrinkage portion for both ℓ_1 and ℓ_2 (s and u respectively) are updated (lines 6-10). After that, for each coefficient w_j, line 12 applies Ridge regularization [17], while lines 13-17 apply the Lasso approach [110]. The Lasso approach uses thresholds in order to decide if the variable is going to be selected (weight different from 0) and updates the threshold for each input variable for the next iterations (q_j in line 17). Finally, the coefficient of determination R^2 is calculated (line 18) and compared with the best obtained so far. If it is better, then the estimated coefficients $\hat{\beta}$ are updated and, if it is not, the number of iterations without improvement (it_{wi}) is incremented by 1. When it_{wi} exceeds the threshold defined in line 27, the algorithm stops. This threshold is directly proportional to the number of examples, and decreases with the number of iterations.

Only those examples in E_s are used to obtain the rule base from the codified chromosome. In this manner, those examples that are not representative are not taken into account for the rule generation. Thus, the method avoids the generation of too specific rules, and reduces the time needed to create the rule base.

Evaluation

The fitness function is based on the estimation of the error of the generated rule base:

$$fitness = MSE(E_{tra}) = \frac{1}{2 \cdot |E|} \sum_{i=1}^{|E|} (F(x^i) - y^i)^2, \tag{3.12}$$

where E_{tra} is the full training dataset and $F(x^i)$ is the output obtained by the knowledge base for input x^i. Using all the examples for evaluation can be seen, in some way, as a validation process, as the rule base was constructed with a subset of them (E_S).

1: **function** SGD-ELASTICNET(X, Y, λ, α, η^0)
2: $it = 0,\ t = 0,\ s = 1,\ u = 0,\ w^0 = 0_{1\times p},\ q = 0_{1\times p}$
3: **repeat**
4: SHUFFLE_ROWS(X) ▷ SGD needs to reorder the rows of X
5: **for** $i = 1,\ldots,n$ **do**
6: $t = t + 1$ ▷ t counts how many updates of the weights have been applied
7: $\eta^t = \eta^0 \cdot (\lambda \cdot t)^{-1}$ ▷ Updates the learning rate to be more conservative
8: $\hat{y}^i = x^i \cdot w^t \cdot s$ ▷ Obtains the estimated output
9: $s = s \cdot (1 - \alpha \cdot \eta^t \cdot \lambda)$ ▷ Updates how much of ℓ_2 was applied
10: $u = u + (1-\alpha) \cdot \eta^t \cdot \lambda$ ▷ Updates how much of ℓ_1 was applied
11: **for** $j = 1,\ldots,p$ **do**
12: $w_j^{t+\frac{1}{2}} = w_j^t - \eta^t \cdot (\hat{y}^i - y^i) \cdot x_j^i / s$ ▷ Applies the ℓ_2 regularization
13: **if** $s \cdot w_j^{t+\frac{1}{2}} > 0$ **then** ▷ Applies ℓ_1 regularization and thresholds
14: $w_j^{t+1} = \max(0, w_j^{t+\frac{1}{2}} - (u + q_j)/s)$ ▷ Positive threshold
15: **else if** $s \cdot w_j^{t+\frac{1}{2}} < 0$ **then**
16: $w_j^{t+1} = \min(0, w_j^{t+\frac{1}{2}} + (u - q_j)/s)$ ▷ Negative threshold
17: $q_j = q_j + s \cdot (w_j^{t+1} - w_j^{t+\frac{1}{2}})$ ▷ Updates thresholds
18: $R_i^2 t = 1 - \frac{1}{n}\sum_{i=0}^{n}(x^i \cdot w^{t+1} \cdot s - y^i)^2$ ▷ Calculates R^2
19: **if** $R_i^2 t > R_{best}^2$ **then** ▷ Updates bests values and iter. without improvement
20: $\hat{\beta} = w^{t+1} \cdot s$
21: $R_{best}^2 = R_t^2$
22: $it_{wi} = 0$
23: **else**
24: $it_{wi} = it_{wi} + 1$
25: $it = it + 1$ ▷ it counts how many times the full dataset was used
26: **until** $it_{wi} > sqrt(|X|/it)$
27: **return** $\hat{\beta}$

Figure 3.10: Pseudocode of SGD for Elastic-Net.

Selection and Replacement

The selection is performed by a binary tournament. On the other hand, the replacement method joins the previous and current populations, and selects the N best individuals as the

new population.

Crossover and Mutation

Two crossover operations are defined: one-point crossover for exchanging the C_1 parts (it also exchanges the corresponding C_2 genes) and, when the C_1 parts are equal, the parent-centric BLX (PCBLX) [51] is used to crossover the C_2 part. In order to prevent the crossover of too similar individuals, an incest prevention was implemented. When the euclidean distance of the lateral displacements is less than a particular threshold L, the individuals are not crossed.

The mutation (with probability p_{mut}) applies two possible operations with equal probability to a randomly selected gene of the C_1 part: i) decreasing the granularity by 1 or ii) increasing the granularity to a more specific granularity —all the granularities have the same chance. In order to calculate the new lateral displacements in the corresponding C_2 part, the displacements of the previous granularity are taken into account. The displacement associated with a particular split point is calculated adding the displacements of the two nearest split points of the previous granularity (before mutation) weighted by the distance between the split points.

Local Search

After the replacement, all the new individuals (their C_1 part of the chromosome was not generated before) go to a local search process. This stage generates n_{ls} new C_1 parts with equal or less granularity —with equal probability— for each variable. Then, the C_2 part is generated randomly with a uniform distribution in the $(-0.5, 0.5)$ interval. The new chromosomes are decoded and evaluated and, if there is a solution that obtains better fitness, then it replaces the original individual.

Restart and Stopping Criteria

The restart mechanism uses the incest prevention threshold L as a trigger. First, L is initialized as the maximum length of the C_2 part, i.e. the product of the number of input variables times the largest maximum granularity of the variables, divided by 4. This implies that the incest prevention allows crossovers between individuals that have a distance higher than a quarter of the maximum euclidean distance. Then, at each iteration, L is decreased in different ways in order to accelerate convergence:

- L is decreased by 0.4 in all the iterations, in order to increase convergence.
- If there are no new individuals in the population, then L is decreased by 0.2.
- If the best individual does not change, L is also decreased by 0.2.

Finally, when L reaches 0, the population is restarted, and L is reinitialized. Only the best individual so far is kept, and the local search process is executed with the best individual in order to generate new individuals until the population is complete. When the restart criterion is fulfilled twice, the algorithm stops, i.e., one single restart is executed. Moreover, if the number of evaluations reaches a threshold, then the algorithm is also stopped. When the evolutionary algorithm stops, the best rule base consequents are optimized applying the SGD algorithm (Sec. 3.4.3) using all the training examples.

3.5 Results

In order to analyze the performance of FRULER, we have used 28 real-world regression problems from the KEEL project repository [4]. Table 3.1 shows the characteristics of the datasets, with the number of instances ranging from 337 to 40,768 examples, and the number of input variables from 2 to 40. The most complex problems —large scale— due to both the number of examples and variables are the ones in the last 8 rows (Table 3.1).

Table 3.1: The 28 datasets of the experimental study stating their number of input variables (#Variables) and examples (#Cases).

Problem	Abbr.	#Variables	#Cases
Electrical Length	ELE1	2	495
Plastic Strength	PLA	2	1,650
Quake	QUA	3	2,178
Electrical Maintenance	ELE2	4	1,056
Friedman	FRIE	5	1,200
Auto MPG6	MPG6	5	398
Delta Ailerons	DELAIL	5	7,129
Daily Electricity Energy	DEE	6	365
Delta Elevators	DELELV	6	9,517
Analcat	ANA	7	4,052
Auto MPG8	MPG8	7	398

Abalone	ABA	8	4,177
Concrete Compressive Strength	CON	8	1,030
Stock prices	STP	9	950
Weather Ankara	WAN	9	1,609
Weather Izmir	WIZ	9	1,461
Forest Fires	FOR	12	517
Mortgage	MOR	15	1,049
Treasury	TRE	15	1,049
Baseball	BAS	16	337
California Housing	CAL	8	20,640
MV Artificial Domain	MV	10	40,768
House-16H	HOU	16	22,784
Elevators	ELV	18	16,559
Computer Activity	CA	21	8,192
Pole Telecommunications	POLE	26	14,998
Pumadyn	PUM	32	8,192
Ailerons	AIL	40	13,750

In the following subsections we show the results obtained by the different parts of the algorithm. Moreover, the results obtained by FRULER are compared with other state of the art approaches.

3.5.1 Experimental Setup

FRULER was designed to keep the number of parameters as low as possible. For the instance selection technique, no parameters are needed. In the multi-granularity fuzzy discretization stage, the fuzziness parameter for the generation of the fuzzy intervals was set to 1, i.e., the highest fuzziness value. For the evolutionary algorithm, the values of the parameters were: population size $= 61$, maximum number of evaluations $= 100,000$, $p_{cross} = 1.0$, $p_{mut} = 0.2$, and $n_{ls} = 5$. For the generation of the TSK fuzzy rule bases, the weight of the tradeoff between ℓ_1 and ℓ_2 regularizations of Elastic Net was $\alpha = 0.95$, and the regularization parameter λ was obtained from a grid search in the interval $[1, 1E-10]$. η^0 was obtained halving the initial value (0.1) until the result worsens.

A 5-fold cross validation was used in all the experiments. Moreover, 6 trials (with different seeds for the random number generation) of FRULER were executed for each 5-fold cross validation. Thus, a total of 30 runs were obtained for each dataset. The results shown in the next section are the mean values over all the runs. The run times were measured using a single thread in an Intel Xeon Processor E5-2650L (20M Cache, 1.80 GHz, 8.00 GT/s Intel QPI).

3.5.2 Performance of the Instance Selection Process

We considered two different measures to evaluate the instance selection process:

- Reduction: is the percentage of reduction in the number of examples, defined as:

$$Reduction = \left(1 - \frac{|E_s|}{|E_{tra}|}\right) \cdot 100 \tag{3.13}$$

  where $|E_s|$ is the number of examples in the subset of selected examples and $|E_{tra}|$ is the original number of examples in the training set.

- Increase in error: is the increment in the error after applying the instance selection process, defined as:

$$Increase = \frac{\varepsilon_{E_s}}{\varepsilon_{E_{tra}}} \tag{3.14}$$

  where ε_E is the mean squared error of a `leave-one-out 1NN` regression.

Table 3.2: Average (5-fold cross validation) results of reduction (percentage of reduction in the number of examples), error increase (increment in the error after applying the instance selection process) and runtime obtained by the instance selection method for each dataset.

Datasets	Reduction (%)	Increase in error	Time (m:s)
ELE1	83.4	0.85	00:54
PLA	91.8	0.76	01:54
QUA	70.9	1.07	03:50
ELE2	84.9	4.08	02:09
FRIE	79.5	1.53	01:33
MPG6	81.6	1.24	00:45
DELAIL	96.9	1.04	10:59
DEE	77.7	1.14	00:42
DELELV	94.0	0.97	15:44

ANA	98.8	16.61	05:09
MPG8	81.4	0.92	00:44
ABA	91.7	0.96	06:13
CON	88.0	1.33	02:04
STP	73.6	2.35	02:07
WAN	85.4	1.58	02:03
WIZ	64.2	1.36	02:37
FOR	93.3	0.54	00:58
MOR	83.4	4.28	02:12
TRE	81.6	5.35	02:20
BAS	83.5	1.61	00:38
CAL	91.6	1.27	39:44
MV	98.7	3.87	40:33
HOU	95.4	1.12	46:08
ELE	96.0	1.33	30:49
CA	98.9	7.40	11:57
POLE	98.7	18.36	27:15
PUM	80.3	1.01	14:48
AIL	95.4	1.12	24:31

Table 3.2 shows the average values of instances reduction and error increase for each dataset. The percentage of reduction achieved is over 80% in most of the datasets. In four of the datasets (QUA, FRIE, DEE, STP) the reduction is in the range 70-80%, and only one dataset (WIZ) has a reduction under 70% (64.2%). The reduction rate does not depend neither on the size of the dataset, nor on the number of variables, but on the complexity of the data. On the other hand, the increase in 1NN error is very low, as it is greater than 2 for only eight datasets (ELE2, ANA, STP, MOR, TRE, MV, CA, POLE). The run time of the instance selection process is generally low, and only the large scale problems take more than 15 minutes.

3.5.3 Performance of the Multi-Granularity Fuzzy Discretization Process

We evaluated the discretization with three different measures:

- Average maximum granularity (over all the variables) for each dataset. This measure summarizes the complexity of the fuzzy partitions generated by the discretization.
- Maximum granularity among the variables for each dataset. This represents the upper bound of the fuzzy partitions obtained for each dataset. It is expected that the smaller this value, the simpler the models obtained by FRULER.
- The number of variables that have not been discretized at all, i.e., their maximum granularity is equal to 1.

Table 3.3: Results of the multi-granularity fuzzy discretization process for each dataset —5-fold cross validation. The table shows the average granularity of all the input variables (Average), the maximum granularity (Max), the number of input variables with granularity 1 (#Not Used), and the runtime (Time).

Problem	Average	Max	#Not used	Time (s:ms)
ELE1	2.30	2.40	0.00	00:11
PLA	3.50	4.40	0.00	00:17
QUA	3.93	7.80	0.00	00:26
ELE2	5.10	7.60	0.00	00:21
FRIE	2.00	2.00	0.00	00:18
MPG6	3.56	5.80	0.00	00:13
DELAIL	9.76	14.00	0.00	00:65
DEE	2.17	3.00	0.00	00:11
DELELV	7.80	15.40	0.80	00:63
ANA	1.97	6.60	5.00	00:13
MPG8	2.86	5.60	1.00	00:09
ABA	3.95	7.80	1.00	00:35
CON	5.95	14.00	1.00	00:29
STP	4.27	9.40	0.00	00:24
WAN	5.04	14.80	0.00	00:28
WIZ	4.27	9.00	0.00	00:25
FOR	2.43	6.00	4.00	00:15
MOR	3.95	8.00	0.00	00:27
TRE	3.71	6.20	0.00	00:35
BAS	2.58	5.20	4.00	00:13
CAL	5.13	13.80	0.00	01:40

MV	3.38	18.60	3.00	02:91
HOU	3.68	12.20	5.00	02:20
ELE	8.03	17.20	2.00	01:57
CA	4.14	14.40	8.00	00:73
POLE	4.52	16.00	5.00	01:05
PUM	2.04	3.20	0.00	01:41
AIL	6.69	19.00	6.20	02:01

Table 3.3 summarizes the results for each dataset. The average maximum granularity is below 9 in all the cases except for DELAIL dataset. Moreover, the maximum granularity is always below 20 and only in 11 cases (DELAIL, DELELV, CON, WAN, CAL, MV, HOU, ELE, CA, POLE, AIL) it is above granularity 10. Even in the datasets with high granularities, the maximum number of fuzzy sets does not generate a huge search space for the evolutionary algorithm. Finally, the number of variables without discretization is 0 in most of the cases. In terms of run time, the discretization module has almost no cost, as the most expensive discretization process is less than 3 seconds.

3.5.4 Statistical Analysis

In this section we compare FRULER with three of the most accurate genetic fuzzy systems for regression in the literature:

- $FS_{MOGFS}{}^{e}$+TUN^{e} [3]: a multi-objective evolutionary algorithm that learns Mamdani fuzzy rule bases. This algorithm learns the granularities from uniform multi-granularity fuzzy partitions (up to granularity 7) and the lateral displacement of the labels. It includes a post-processing algorithm for tuning the parameters of the membership functions and for rule selection.
- L-METSK-HD^{e} [39]: a multi-objective evolutionary algorithm that learns linguistic TSK-0 fuzzy rule bases. The algorithm learns the granularities from uniform multi-granularity fuzzy partitions (up to granularity 7).
- A-METSK-HD^{e} [39]: a multi-objective evolutionary algorithm that learns approximative TSK-1 fuzzy rule bases. The algorithm starts with the solution obtained on the first stage and applies a tuning of the membership functions, rule selection and a Kalman-based calculation of the consequents of the rules.

Table 3.4: Average number of rules (#Rules) and test MSE (Test Error) for the compared algorithms. The test errors in this table should be multiplied by $10^5, 10^{-8}, 10^{-6}, 10^9, 10^8, 10^{-6}, 10^{-4}, 10^{-8}$ in the case of ELE1, DELAIL, DELELV, CAL, HOU, ELV, PUM, AIL respectively.

algorithms	FRULER		FS_{MOGFS}^{e}+TUN^{e}		L-METSK-HD^{e}		A-METSK-HD^{e}	
	#Rules	Test Error	#Rules	Test Error	#Rules	Test Error	#Rules	Test Error
ELE1	**4.1**	2.012	8.1	**1.954**	15	1.925	11.4	2.022
PLA	**1.4**	1.219	18.6	1.194	23	1.218	19.2	**1.136**
QUA	7.8	0.0181	**3.2**	**0.0178**	35.9	0.019	18.3	0.0181
ELE2	**4.3**	6,729	8	10,548	59	20,095	36.9	**3,192**
FRIE	**8.0**	**0.731**	22	3.138	95.1	3.084	66	1.888
MPG6	**13.7**	**3.727**	20	4.562	99.6	4.469	53.6	4.478
DELAIL	**2.5**	1.458	6.2	1.528	98.3	1.621	36.8	**1.402**
DEE	**7.9**	**0.080**	18.3	0.093	96.4	0.095	50.6	0.103
DELELV	**5.8**	1.045	7.9	1.086	91	1.119	39.1	1.031
ANA	**3.9**	0.008	10	**0.003**	48.9	0.006	33.3	0.004
MPG8	**12.7**	**4.084**	23	4.747	98.7	5.61	64.2	5.391
ABA	**4.5**	2.393	8	2.509	42.4	2.581	23.1	**2.392**
CON	**8.9**	**20.598**	15.4	32.977	96.5	38.394	53.7	23.885
STP	42.4	**0.353**	**23**	0.912	100	0.78	66.4	0.387
WAN	**5.6**	**0.888**	8	1.635	91.1	1.773	48	1.189
WIZ	**8.9**	**0.663**	10	1.011	55.4	1.296	29.1	0.944
FOR	**5.6**	**2,214**	10	2,628	93.7	4,633	40.6	5,587
MOR	7.9	**0.007**	7	0.019	40.9	0.028	27.2	0.013
TRE	**4.5**	**0.027**	9	0.044	42.8	0.052	28.1	0.038
BAS	**6.2**	305,777	17	**261,322**	95.7	320,133	59.8	368,820
CAL	15.4	2.110	**8.4**	2.95	99.8	2.638	55.8	**1.71**
MV	**6.0**	0.083	14	0.158	76.4	0.244	56.5	**0.061**
HOU	12.1	**8.005**	**11.7**	9.4	68.9	10.368	30.5	8.64
ELE	**5.4**	**2.934**	8	9	76.4	8.9	34.9	7.02
CA	**7.1**	**4.634**	14	5.216	71.3	5.88	32.9	4.949
POLE	40.8	110.898	**13.1**	102.816	100	150.673	46.3	**61.018**
PUM	**7.8**	0.367	17.6	0.292	87.5	0.594	63.3	**0.287**
AIL	**8.5**	**1.404**	15	2	99.1	1.822	48.4	1.51

Table 3.4 shows the average results of FRULER and the three algorithms selected for comparison. Two different results are shown for each algorithm and dataset: the number of rules of the obtained rule base, and the test error measured using equation 3.12 over the test data. These indicators allow to compare both the simplicity and the accuracy of the learned models. The values with the best accuracy —lowest error— and best number of rules in table 3.4 are marked in bold.

It can be seen that the number of rules of FRULER is the lowest in the majority of the datasets. It should be noted that the number of rules in the large scale problems (the last 8 problems) is also low despite the high number of examples. Only in 5 problems the $FS_{MOGFS}{}^{e}$+TUN^{e} Mamdani proposal produces the lowest number of rules. In the case of accuracy, in 15 of the 28 problems FRULER achieves the best results. In the other 13 datasets, the best results are for $FS_{MOGFS}{}^{e}$+TUN^{e} (best in 4 problems) and A-METSK-HD^{e} (best in 9 problems). From the results, we did not find any influence in the performance of FRULER by neither the training dataset size nor the dimensionality of the problem.

In order to analyze the statistical significance of these results, we used the STAC platform [89] to apply the statistical tests. A Friedman test was used for both the number of rules and the test error in order to get a ranking of the algorithms and check whether the differences between them were statistically significant.

Table 3.5: Friedman test ranking results and test p-value for the test error in table 3.4.

Algorithm	Ranking
FRULER	1.714
A-METSK-HD^{e}	2.036
$FS_{MOGFS}{}^{e}$+TUN^{e}	2.786
L-METSK-HD^{e}	3.464
p-value	$< 1E-5$

Table 3.6: Wilcoxon comparison for the two most accurate algorithms in table 3.5.

Comparison	p-value
FRULER vs A-METSK-HD^{e}	0.079

Table 3.5 shows the ranking for the test error, with the p-value of the test. Our proposal —generates linguistic TSK-1 rules— gets the top ranking, i.e., it has the best results in accuracy among all the algorithms. Then, the next algorithm in the ranking is the approximative approach, due to its fine tuning of the rules, followed by the linguistic approaches. In order to compare whether the difference between FRULER and the second ranked algorithm (A-METSK-HD^{e}[39]) was significant, we performed a Wilcoxon test (Table 3.6). The p-value indicates that the difference is statistically significant when using a significance level of 0.1. Thus, even with linguistic rules, FRULER obtains a great accuracy compared to approximative approaches, while getting simpler models.

Table 3.7: Friedman test ranking and test p-value results for the number of rules in table 3.4.

Algorithm	Ranking
FRULER	1.214
$FS_{MOGFS}{}^{e}$+TUN^{e}	1.786
A-METSK-HD^{e}	3
L-METSK-HD^{e}	4
p-value	$< 1E-5$

Table 3.8: Wilcoxon comparison of the number of rules for the two most accurate approaches in table 3.7.

Comparison	p-value
FRULER vs A-METSK-HD^{e}	$< 1E-4$

To analyze the complexity of the models obtained for each algorithm, the same Friedman test was performed to the number of rules in table 3.4 (Table 3.7). Once again, FRULER has the lowest ranking. The next algorithm in the ranking is $FS_{MOGFS}{}^{e}$+TUN^{e} Mamdani approach, followed by the METSK-HD^{e} approaches with a big difference in the ranking. We applied a Wilcoxon test (Table 3.8) in order to assess whether the difference in complexity among the most accurate proposals (Table 3.5) was significant. The difference is statistically significant (p-value equal to $1E-4$) in the number of rules. This shows that FRULER obtains accurate solutions with simpler models.

Table 3.9: Average runtime and number of evaluations per run of FRULER.

Datasets	ELE1	PLA	QUA	ELE2	FRIE	MPG6	DELAIL
Time (h:m:s)	0:00:51	0:01:41	0:09:48	0:03:05	0:05:46	0:02:12	0:09:58
Evaluations	8,885	7,345	13,020	16,798	17,283	21,556	19,236
Datasets	DEE	DELELV	ANA	MPG8	ABA	CON	STP
Time (h:m:s)	0:02:26	0:25:01	0:05:05	0:03:29	0:17:45	0:05:55	0:27:41
Evaluations	24,131	24,386	27,107	29,355	31,537	32,318	38,468
Datasets	WAN	WIZ	FOR	MOR	TRE	BAS	CAL
Time (h:m:s)	0:10:27	0:26:03	0:02:28	0:27:50	0:23:30	0:04:41	1:57:03
Evaluations	35,812	36,168	45,367	60,101	57,569	59,362	33,951
Datasets	MV	HOU	ELE	CA	POLE	PUM	AIL
Time (h:m:s)	1:17:02	4:15:17	3:01:30	0:38:12	1:53:15	31:14:27	12:50:38
Evaluations	35,001	61,709	68,055	78,036	99,827	96,543	100,000

Table 3.9 shows the average run time of FRULER in each dataset. We also display the number of evaluations until the stopping condition was met. Although each of the stages of FRULER increases the computational complexity, they contribute to focus the search on the simplest models. Our method obtains solutions in the range between 1-23 minutes for datasets 1-20 (the most simple ones) and solutions in the range from 1-30 hours for datasets 21-28 (the most complex ones). Moreover, the number of evaluations is below the limit $(100,000)$, except for the largest problem (AIL). The run time of FRULER is in the same order of magnitude as A-METSK-HD, being only worse in six datasets (QUA, WIZ, MOR, TRE, PUM and AIL) [2].

In order to demonstrate the simplicity of the models generated by FRULER, Figure 3.11 shows an example of one of the rule bases generated for the WAN dataset. There are two columns for each rule: the fuzzy sets in the antecedent and the weight of the variables in the consequent. For the sake of simplicity and understandability, the consequents are represented

[2] A quantitative comparison with the computational times of [39] has not been performed, since the processors are different

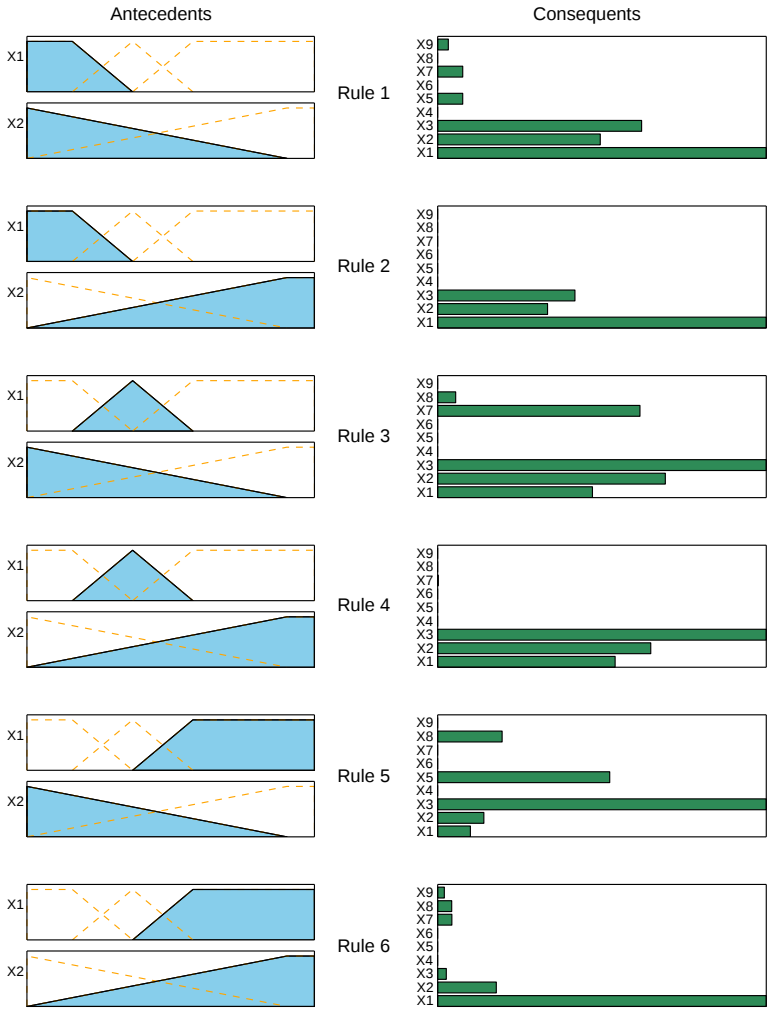


Figure 3.11: An example of TSK fuzzy rule base for the WAN dataset. The system uses only 2 variables for the antecedent part and has 6 different rules. For the sake of simplicity and understandability, the consequents are represented with their absolute value and have been scaled to have the maximum weight equal to 1. The test error obtained by this example is 0.885.

with their absolute value and have been scaled to a maximum weight of 1. The antecedent only uses two variables with granularity 3 and 2 respectively, thus 6 rules are needed to cover all the combinations. On the other hand, the consequent column shows the importance of each

input variable for each rule, providing a qualitative understanding of the model. In this case, the first three variables (X1, X2 and X3) have the greatest importance in the consequent. Note that, even though this is one of the simplest models obtained by FRULER, the test error is very low (0.885).

3.6 Conclusions

In this paper, a novel genetic fuzzy system called FRULER was presented. FRULER learns simple and linguistic TSK-1 knowledge bases for regression problems following a new approach than involves two general-purpose preprocessing stages: a new instance selection for regression and a novel non-uniform multi-granularity fuzzy discretization. Furthermore, FRULER's evolutionary learning algorithm incorporates an automatic generation of the TSK fuzzy rule bases from fuzzy partitions, and uses Elastic Net in order to obtain consequents with low overfitting.

FRULER was compared with three state of the art algorithms that learn different types of fuzzy rules: linguistic Mamdani, linguistic TSK-0 and approximative TSK-1. The results were analyzed using statistical tests of significance, which show that FRULER obtains high accuracy, but with a lower number of rules and with a linguistic data base. This is of particular interest in problems where both high accuracy and interpretability are demanded, in order to provide simple and understandable regresssion models to the users.

Chapter 4

S-FRULER: Scalable Fuzzy Rule Learning through Evolution for Regression

This chapter focuses on the scalability of FRULER, the proposed GFS that learns simple and linguistic TSK-1 knowledge bases for regression problems (in chapter 3). Although the runtime of FRULER is acceptable (between 1-23 minutes) for the simplest datasets, it does not scale properly when solving large scale problems (from 1 to 30 hours). Moreover, with larger problems it may not converge to a good solution. The goal is to obtain models with similar characteristics than those generated by FRULER —accurate and simple—, but reducing the runtime of the algorithm to converge.

In GFSs the size of the problem has a huge influence in the performance of the obtained models, since i) the learned fuzzy rule bases suffer from exponential rule explosion when the number of variables increases, and ii) the convergence time increases with the number of examples. This chapter presents S-FRULER, a scalable distributed version of FRULER, which uses the Spark software. S-FRULER focuses on splitting the problem into smaller partitions and incorporates a feature selection process for reducing the number of variables used in each partition. Each partition is then solved independently using the FRULER algorithm. Then, an aggregation function is used to obtain linguistic TSK fuzzy rule bases from the rule bases generated for each dataset partition.

S-FRULER has been validated in terms of scalability, precision and complexity using 10

large-scale datasets and has been compared with three state of the art GFSs. Experimental results show that S-FRULER scales well and achieves simple linguistic models with a precision comparable with approximative models. Moreover, S-FRULER has been applied to a bioinformatics problem that was proposed as a benchmark for scalability of regression problems, obtaining good results in both accuracy and complexity.

In this chapter, a full copy of the following publication is presented:

> I. Rodríguez-Fdez[1], M. Mucientes[1], and A. Bugarín[1]. S-FRULER: Scalable Fuzzy Rule Learning through Evolution for Regression. *Knowledge-Based Systems*, Elsevier, Available online 26 July 2016, DOI:10.1016/j.knosys.2016.07.034

4.1 Abstract

In genetic fuzzy systems (GFS) the size of the problem has a huge influence in the performance of the obtained models, since i) the fuzzy rule bases learned suffer from exponential rule explosion when the number of variables increases, and ii) the convergence time increments with the number of examples. In this paper we present S-FRULER, a scalable distributed version of FRULER which is a GFS that learns simple and linguistic TSK-1 knowledge bases for regression problems. S-FRULER obtains models with high accuracy and low complexity, whilst reducing the algorithm runtime. S-FRULER focuses on splitting the problem into smaller partitions and incorporates a feature selection process for reducing the number of variables used in each partition. Each partition is then solved independently using the FRULER algorithm. Afterwards, an aggregation function obtains the final linguistic TSK fuzzy rule base from the information generated in each partition. S-FRULER has been validated in terms of scalability, precision and complexity using 10 large-scale datasets and has been compared with three state of the art GFSs. Experimental results show that S-FRULER scales well while achieving simple models with a linguistic approach and a precision comparable with approximative models. Moreover, S-FRULER has been applied to a bioinformatics problem that was proposed as a benchmark for scalability of regression problems, obtaining good results in both accuracy and complexity.

[1]Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain.

4.2 Introduction

The vast majority of techniques in statistical learning and data mining were traditionally designed to work with a limited amount of data [47]. The adaptation of these approaches to large scale datasets has been a huge challenge addressed in the last years [43]. Usually, all the problems that emerge with the use of big amounts of data are labelled under the term Big Data, which covers distributed processing and storing of data. Statistical learning algorithms in a Big Data context suffer from the problem of scalability, that is, the capability of the algorithm to maintain competitive performance as the size of the problem increases. Performance of the algorithms refers in this context to the computational cost, the precision of the obtained models, and their complexity. Also, the size of the problems can grow in two different dimensions: the number of examples available in training and the number of variables considered.

Recently, the use of Fuzzy Systems in the Big Data paradigm has attracted attention [35]. The first approach in this field was to scale the fuzzy c-means algorithm [48]. More recently, fuzzy modeling was applied to medical big data problems, using a neuro-fuzzy classifier for dimensionality reduction [9]. In [70], the authors propose an algorithm for Big Data binary imbalanced classification problems using the MapReduce scheme. Finally, in [30] it is introduced the first approach of a fuzzy rule based associative classifier based on the MapReduce approach.

Particularly, in a genetic fuzzy system (GFS) approach, the size of the problem has a huge influence in the performance of the obtained models [26, 49]. The fuzzy rule bases learned suffer from exponential rule explosion when the number of variables increases. Thus, with huge search spaces, the convergence time towards precise and simple models rises. Moreover, evolutionary algorithms are computationally expensive by themselves due to the large number of evaluations needed to reach convergence. Furthermore, in many cases, the evaluation process to obtain the fitness may take a long time. There are different techniques to improve the scalability of GFS that can be classified into three categories [36]: i) algorithm oriented that adapts the structure of the evolutionary algorithm, ii) data oriented that modifies the training data to reduce the computational cost of the learning process, and iii) distributed approaches that take advantage of the availability of several machines to reduce the runtime.

To scale the learning process of a GFS in an algorithm-oriented manner, several papers in the literature focused in the control of the search space, by reducing the number of rules and/or the number of labels used in the rule base through a multi-objective approach [2, 3, 33, 8]. Specifically, in linguistic approaches where the partition of each variable into fuzzy labels is

defined equally for all rules, rule explosion can be controlled by limiting the number of labels considered in the learning process [2, 94, 95]. Moreover, in recent years, the data oriented approach has received increasing attention. The use of instance selection techniques decreases the complexity of large scale problems and reduces overfitting. In [2] was introduced an estimation error mechanism which selects randomly a subset of examples to estimate the real error, and the complete training dataset was only used for the most promising individuals. Also, in [92], a new instance selection method for regression was applied to generate the rules with the selected examples and the error of the rule base was estimated with the whole training dataset.

From a Big Data point of view, the distributed computing approach is the most appropriate for scaling GFS. Among the most frequently used frameworks in Big Data analytics [88], the most popular ones are: i) MPI (Message Passing Interface) which efficiently exploits multi-core clusters architectures, and ii) Apache Spark [118], a recently developed platform that can be executed in traditional clusters such as Hadoop [115]. Spark was designed to perform distributed processing and other workloads like streaming, interactive queries, and machine learning focused algorithms. While MPI provides a solution mostly oriented to high performance computing, Spark also deals with failures and straggler nodes effectively but with an impact on speed. From the perspective of GFS, only a few works use Big Data frameworks to solve the scaling problem [36].

The extended use of Spark is closely linked to the success of Hadoop, which processes vast amounts of data in parallel on large clusters, usually implemented using the Hadoop Distributed File System. Hadoop popularized the approach of MapReduce, a distributed methodology based on the definition of two different functions: Map and Reduce [28]. On one hand, a Map function distributes a block of data to several Worker Nodes, and executes the same process — called Task — in each data partition. On the other hand, the Reduce function aggregates the results of the Map functions by means of a Key-Value representation of the results and performs some operations to obtain the final result. Spark adds to this framework the capability to use other data-flows with an improvement of in-memory computing and an easy-of-programming high-level functions that facilitate to build parallel applications.

Solving large scale regression problems with GFSs can be found in some of the most recent works in the field [2, 3, 94, 95]. However, the number of variables and/or number of examples in the datasets used in these works are still not high enough to be properly considered labelled as Big Data. Among the different approaches, FRULER [95] obtains Takagi-Sugeno-

Kang 1-order (TSK-1) fuzzy rule bases with high accuracy and the lowest number of rules. Although the runtime of this approach is acceptable (between 1-23 minutes) for the most simple datasets, it does not scale properly when solving large scale problems (from 1 to 30 hours). Moreover, with larger problems it may not converge to a good solution in reasonable time.

In this paper we propose a scalable version of FRULER, called S-FRULER, which allows to obtain models with similar characteristics than those obtained by FRULER —accurate and simple—, but reducing the runtime of the algorithm to converge. The main contributions of this work are: i) a novel distributed GFS approach, ii) a random feature selection process to reduce the number of variables used in each dataset partition, and iii) an aggregation function to obtain linguistic TSK fuzzy rule bases from the rule bases obtained in each dataset partition.

This paper is structured as follows: Section 2 defines the TSK model used in this work and its implications for a Big Data approach. Section 3 describes the different stages of S-FRULER. Section 4 shows the results of the approach in 10 regressions problems and the application of S-FRULER to a large bioinformatics problem. Finally, Section 5 presents the conclusions.

4.3 TSK Fuzzy Systems and Big Data

In this section the factors to be taken into account when learning TSK fuzzy rule bases in a Big Data environment are described. Takagi, Sugeno, and Kang proposed in [108, 106] a fuzzy rule model in which the antecedents are comprised of linguistic variables, as in the case of Mamdani [73, 74], but the consequent is represented as a polynomial function of the input variables. This type of rules is called TSK fuzzy rules. The most common function for the consequent of a TSK rule is a linear combination of the input variables (TSK-1), and its structure is as follows:

$$\begin{aligned} &\text{If } X_1 \text{ is } A_1 \text{ and } X_2 \text{ is } A_2 \text{ and } \ldots \text{ and } X_p \text{ is } A_p \text{ then} \\ &Y = \beta_0 + X_1 \cdot \beta_1 + X_2 \cdot \beta_2 + \cdots + X_p \cdot \beta_p \end{aligned} \tag{4.1}$$

where X_j represents the j-th input variable, p the number of input variables, A_j is the linguistic fuzzy term for X_j, Y is the output variable, and β_j is the coefficient associated with X_j in the consequent part of the rule.

The matching degree h between the antecedent of the rule r_k and the current inputs to the

system $(x_1, x_2, \ldots, x_p)$ is calculated as:

$$h_k = T(A_1^k(x_1), A_2^k(x_2), \ldots, A_p^k(x_p)) \tag{4.2}$$

where A_j^k is the linguistic fuzzy term for the j-th input variable in the k-th rule and T is the t-norm conjunctive operator, usually the minimum function. The final output of a TSK fuzzy rule base system composed by m TSK fuzzy rules is computed as the average of the individual rule outputs Y_k weighted by the matching degree:

$$\hat{y} = \frac{\sum_{k=1}^{m} h_k \cdot Y_k}{\sum_{k=1}^{m} h_k} \tag{4.3}$$

In a regression analysis, the β coefficients are the relation that transforms the inputs into the desired output. The most fitted coefficients to the data can be found minimizing the least squares equation:

$$\hat{\beta} = \arg\min_{\beta} ||Y - X \cdot \beta||_2^2 \tag{4.4}$$

where β is the coefficients vector $(\beta_0, \beta_1, \ldots, \beta_p)$, Y is the outputs vector $(y^1, \ldots, y^n)$, X is the inputs matrix with size $n \times p$ —rows represent examples while columns are the input variables. The coefficients associated with each rule consequent cannot be learned separately using Eq. 4.4 because the function that needs to be approximated is the aggregation of all rules (Eq. 4.3). Therefore, all the coefficients must be optimized at the same time, taking into account the degree of fulfillment of each rule (Eq. 4.2) for each input vector. Thus, the X matrix is modified as follows:

- The normalized degree of fulfillment for each rule r_k for each example e^i is calculated as:

$$z_k^i = \frac{h_k(x^i)}{\sum_{u=1}^{m} h_u(x^i)} \tag{4.5}$$

  where the denominator is the normalization term for each input vector x^i, i.e., the summation of the degree of fulfillment of all rules.

- Then, the X matrix is defined as:

$$X = \begin{pmatrix} z_1^1, x_1^1 \cdot z_1^1, \ldots, x_p^1 \cdot z_1^1, \ldots, z_m^1, x_1^1 \cdot z_m^1, \ldots, x_p^1 \cdot z_m^1 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ z_1^n, x_1^n \cdot z_1^n, \ldots, x_p^n \cdot z_1^n, \ldots, z_m^n, x_1^n \cdot z_m^n, \ldots, x_p^n \cdot z_m^n \end{pmatrix} \tag{4.6}$$

where each row replicates the input vector $x^i = (1, x_1^i, x_2^i, \ldots, x_p^i)$ —where a 1 was added to take into account the independent term— as many times as the number of rules (m), weighting each rule r_k by z_k^i.

- Finally, the coefficient vector is the concatenation of the coefficients of all rules:

$$\beta = \left(\beta_0^1, \beta_1^1, \ldots, \beta_p^1, \ldots, \beta_0^m, \beta_1^m, \ldots, \beta_p^m\right) \tag{4.7}$$

The computational cost for automatic learning of the coefficients of a TSK fuzzy rule base depends on the size of the X matrix (Eq. 4.6). The size of X not only depends on the number of examples (n) and the number of variables (p), but also on the number of rules (m). The number of rules increases exponentially with the number of variables, number of fuzzy labels per variable and the number of examples to be covered due to rule explosion. Thus, this process is the largest computational cost in a GFS that learns TSK Fuzzy rule bases and the most critical point to take into account for scalability. This can be partly solved through limiting the number of available labels — controlling the rule explosion and reducing the size of X —, using instance selection — reducing the number of examples n — or by feature selection — reducing the number of variables p.

Traditionally, an iterative Kalman filter was used to solve least squares to approximate the most fitted solution to the data [108]. However this approach has the drawbacks of overfitting and the high computational cost of the iterative Kalman filter. On one hand, the overfitting problem can be solved by shrinking (Ridge regularization) [91] or setting some coefficients to zero (Lasso regularization) to obtain simpler models. Moreover, a combination of both regularizations, called Elastic Net [119], can be used [95].

On the other hand, new scalable approaches for solving regression problems, regularized regression problems in particular, have been proposed in the recent years, such as coordinate descent [38] and stochastic gradient descent (SGD) [110, 17]. SGD is characterized by updating each coefficient separately using only one example at a time. This is particularly suited for sparse datasets, which is a common case when X is constructed using Eq. 4.6 —z_k^i is 0 when a rule does not cover an example. For example, in [95], a SGD approach was developed to solve Elastic-Net regularization to obtain the consequents of TSK-1 Fuzzy rule bases.

4.4 S-FRULER

This section presents S-FRULER (Scalable Fuzzy Rule Learning through Evolution for Regression), a distributed approach for applying FRULER [95] with scalability properties that allow its application to large scale problems. FRULER is a GFS that learns simple and linguistic TSK-1 knowledge bases for regression problems. It is composed of a two-stage preprocessing — formed by an instance selection and a multi-granularity fuzzy discretization—, and a genetic algorithm, which contains an ad-hoc TSK-1 rule generation module. Although the runtime of this approach is acceptable for medium size datasets, it does not scale properly when solving large scale problems as it may not converge. To solve that, S-FRULER divides the problem into a set of smaller problems that are more tractable using a distributed approach. Each of the divisions is then solved independently in the Map phase using FRULER. Then, the solutions obtained in each Map are combined in the Aggregation phase in order to obtain a final solution for the original data.

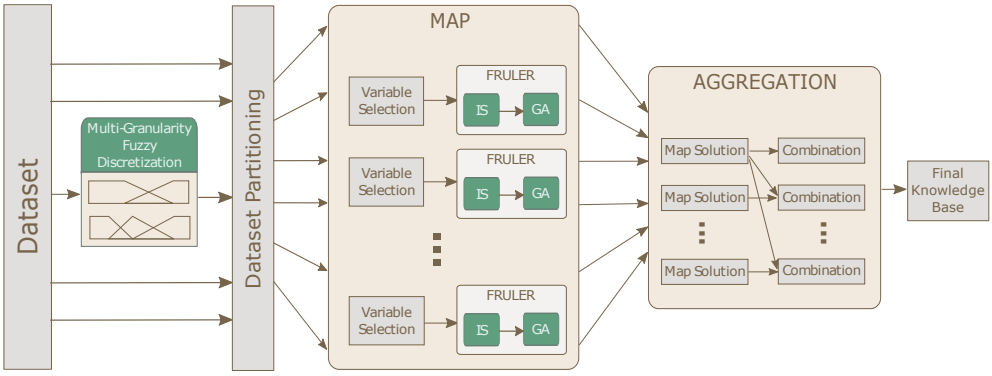


Figure 4.1: S-FRULER architecture showing the preprocessing, Map and Aggregation phases.

The algorithm structure is shown in figure 4.1. First, the multi-granularity fuzzy discretization process is performed using the whole training dataset. Then, the training dataset is splitted into n_{map} partitions, which correspond to the tasks to be distributed into the Working Nodes. Also, for each dataset partition, only a subset of randomly selected variables is taken into account. Each task is solved using FRULER —without the discretization phase—, as if it was an independent problem. Thus, only the instance selection (IS) and the genetic algorithm (GA) are executed. After obtaining the solutions for each task, these are combined completing

the variables not used in one dataset partition with information of the others. The following subsections describe each of these phases in more detail.

4.4.1 Preprocessing before mapping

This stage comprises the multi-granularity fuzzy discretization of the input variables and the partition of the training dataset.

Multi-granularity Fuzzy Discretization

The first step of S-FRULER consists in the application of the Multi-granularity Fuzzy Discretization method used in FRULER [95] to discretize the input variables into fuzzy labels. This method obtains non-uniform fuzzy partitions with different degrees of granularity. A granularity g^i_{var} divides the variable *var* into *i* fuzzy labels, i.e., $g^i_{var} = \{A^{i,1}_{var}, \ldots, A^{i,i}_{var}\}$.

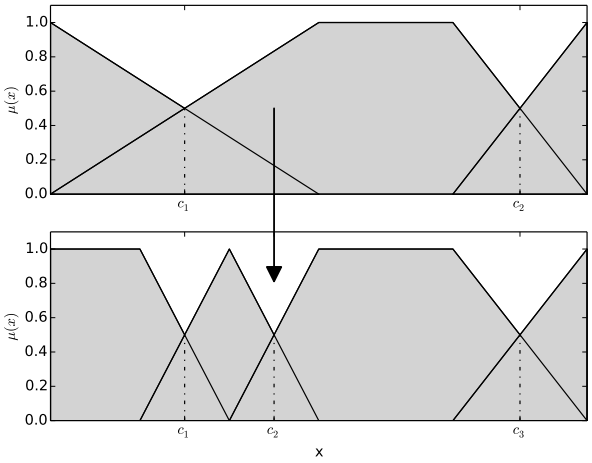


Figure 4.2: Top-down approach for the multi-granularity discretization. Only one label is divided into two new labels in order to obtain the next granularity.

The algorithm works as follows[2]:

[2]For the sake of completeness, the stages of the algorithm are sketched in this section. A detailed description can be found in [95]

- First, each variable is discretized to obtain a set of split points C^g for each granularity g.
  - The split points are searched iteratively, i.e., only a new split point is added at each granularity, starting from the most general granularity. Therefore, the approach proposed in this work aims to preserve interpretability between contiguous granularities: adding a new label to the previous granularity and modifying the adjacent labels (Fig. 4.2).
  - The split points that minimize the error when a linear model is applied to each of the resulting intervals are selected.
  - The method keeps adding split points until a bayesian information criterion (BIC) worsens. To calculate the BIC, the error is measured as the summation of the mean squared error of a least squares fitted model for each interval of the discretization, while the complexity is determined by the number of inner splits and the parameters fitted by each regression applied in each interval.
- Then, the method proposed in [55] is applied to each C^g —set of split points for the granularity g— in order to get the multi-granularity fuzzy partitions. This method uses a parameter that assesses the fuzziness of the linguistic labels. Fuzziness 0 indicates crisp intervals, while fuzziness 1 indicates the selection of a fuzzy set with the smallest kernel —set of points with membership equal to 1.

Dataset Partitioning

After the discretization of the input variables, the dataset is partitioned and distributed along the Workers Nodes of the computation cluster. The training set is divided randomly into n_{map} partitions with equal size $\frac{n}{n_{map}}$, where n is the total number of examples in the training data. In scalability terms, the higher the number of partitions n_{map} that can be used, the better. However, when the number of partitions surpasses a certain problem-dependent threshold, the obtained results worsen. This means that the problem has been divided too much, and each Map has not enough information to get a valid result. Thus, the automatic definition of the most adequate number of partitions for each problem can be useful when no information about the underlying characteristics of the problem is known. In S-FRULER, the number of

partitions n_{map} is defined heuristically as:

$$n_{map} = log_2(p^2 * l * n) \tag{4.8}$$

where p is the number of input variables, n is the number of examples and l is the maximum granularity over all the input variables. Thus, n_{map} depends on the maximum size of X (Eq. 4.6) allowed in the initialization of FRULER, which is the critical part in scaling the algorithm (See Sec. 4.3).

4.4.2 Map function

The Map function is applied independently and distributively to each training dataset partition. For each training dataset partition, only a subset of selected variables is used. Thus, all the process done in each task uses the corresponding subset of examples and subset of variables, which is going to be referred as the dataset partition. For each task, FRULER is applied without the discretization step, since it was already performed before the partition of the data. It is composed by an instance selection of the most representative examples and a genetic algorithm, which contains an ad-hoc TSK-1 rule generation module (Fig. 4.3). The evolutionary learning process obtains a definition of the data base. Then an ad-hoc TSK-1 rule generation module obtains the antecedents and consequents of each possible rule using only the representative examples.

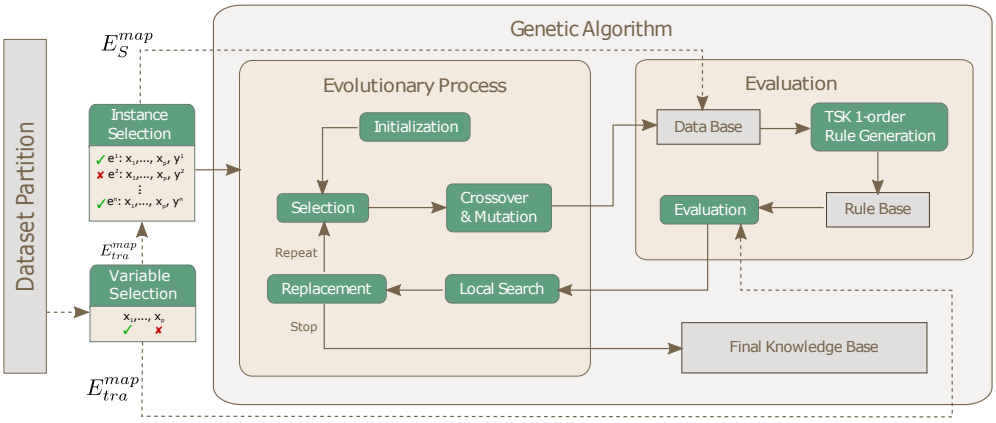


Figure 4.3: FRULER [95] architecture showing each of the two stages. Dashed lines indicate flow of data sets and solid lines represent process flow.

Variable Selection

To simplify the learning process in each task, only a subset of randomly selected variables is used for each dataset partition. The probability of selecting a particular input variable X_j in a dataset partition is:

$$P(X_j \in X_s^i) = \frac{p_m}{p}, \tag{4.9}$$

where X_s^i is the selected subset of input variables in the dataset partition i, p_m is the subset size of selected input variables and p is the total number of input variables. Thus, the probability that a particular input variable is not selected for all the dataset partitions is:

$$P(X_j \notin X_s^i, \forall i = 1, \ldots, n) = \left(\frac{p - p_m}{p}\right)^{n_{map}}. \tag{4.10}$$

where n_{map} is the total number of dataset partitions. Therefore, to define a subset size of selected input variables p_m that assures that a particular feature X_j is selected in at least one dataset partition with probability α_{p_m}, the following equation can be used:

$$p_m \geq -p \cdot ((1 - \alpha_{p_m})^{1/n_{map}} - 1) \tag{4.11}$$

Therefore, each task uses a dataset formed by $\frac{n}{n_{map}}$ examples from the original training dataset and p_m randomly selected input variables.

Instance Selection for Regression

The instance selection method for regression used in FRULER [95] is an improvement of the CCISR (Class Conditional Instance Selection for Regression) algorithm [92]. The method is based on a relation called class conditional nearest neighbor which is used to obtain the nearest example with the same class and the nearest example of a different class for each instance in the dataset partition. With this information, two graphs can be built: one where each example points to the nearest example with the same class and another where each example points to the nearest example with a different class. Using these graphs, the method obtains the in-degree for same-class and different-class for each example, and then an information measure based on the K-divergence is calculated. This measure is then used to sort the examples by its ability to represent its own class and to differentiate contiguous classes.

Since the class conditional relation uses classes, firstly it is necessary to discretize the output variable. For that, the Kernel Density Estimation (KDE) with a Gaussian Kernel is

used to estimate the probability density function of the output. Then, the local minima of this function are used as the split points between classes, and, therefore, to determine which class corresponds to each example.

Then the instance selection method obtains the subset of selected examples for a *map* (E_S^{map}) from the full training dataset partition (E_{tra}^{map}). It performs the following three stages[3]:

- First, the k_0 first examples —sorted by the K-divergence score— are picked as the initial set of selected examples E_S^{map}, where k_0 is defined as:

$$k_0 = max\left(c, \left\lceil \frac{\varepsilon^{E_{tra}^{map}} \cdot |E_{tra}^{map}|}{max(y) - min(y)} \right\rceil\right) \tag{4.12}$$

  where c is the number of classes obtained from KDE, E_{tra}^{map} is the complete training dataset partition and $\varepsilon^{E_{tra}^{map}}$ is the 1-nearest neighbor error for regression using E_{tra}^{map}.

- Then, the second stage adds examples in order, until the error worsens for more than $\sqrt{|E_{tra}^{map}|/|E_S^{map}|}$ iterations.
- Finally, the last stage removes points that are not close to the decision boundary of the 1-nearest neighbor rule, that is, examples with zero in-degree in the different-class graph — there is no other instance that points to the example.

Genetic Algorithm

The evolutionary algorithm is visually described in the Evolutionary Process box in Fig. 4.3. Its objective is to obtain the best data base configuration using the obtained fuzzy partitions (Fig. 4.1). To evaluate each individual, the algorithm generates the entire linguistic TSK-1 fuzzy rule base using the selected fuzzy partitions and the selected examples E_S^{map} (sec. 4.4.2), and then calculates the Mean Squared Error (MSE) using the whole training dataset partition E_{tra}^{map}. In the next sections each step of the evolutionary algorithm are described.

Codification

The chromosome codification represents the parameters needed to create the data base using a double coding scheme ($C = C_1 + C_2$):

[3]For the sake of completeness, the stages of the algorithm are sketched in this section. A detailed description can be found in [95]

- C_1 represents the granularity used in each input variable. It is coded as a vector of p_m integers:

$$C_1 = (g_1, g_2, \ldots, g_{p_m}) \tag{4.13}$$

where g_i represents the granularity for input variable i. When the granularity of a variable is equal to 1, then it is not used in the antecedent part. However, this variable can still be used in the consequent, since it could be relevant for calculating the output.

- C_2 represents the lateral displacements of the split points of the input variables fuzzy partitions. Thus, the length of C_2 depends on the granularity for each input variable: $|C_2| = \sum_{j=1}^{p} (|g_j| - 1), \forall g_j \in C_1$:

$$C_2 = (\alpha_1^1, \ldots, \alpha_1^{g_1-1}, \ldots, \alpha_p^1, \ldots, \alpha_p^{g_p-1}) \tag{4.14}$$

where α_i^j represents the lateral displacement of the j-th split point of variable i. Each lateral displacement are allowed to vary in the $(0.5, 0.5)$ interval which represents half of the distance between each split point (Fig. 4.4).

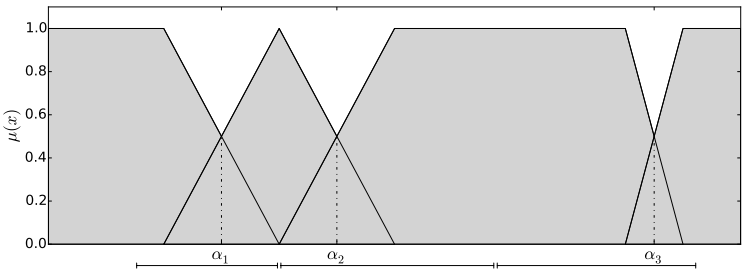


Figure 4.4: An example of lateral displacement intervals for limits equal to $(0.5, 0.5)$. The split points are allowed to move a maximum of half of the distance to the next split point.

Initialization

The initial pool of individuals is generated by a combination of two initialization procedures. A half of the individuals are generated with the same random granularity for each variable, while the other half is created with a different random granularity for each variable. The lateral displacements are initialized to 0 in all cases. After that, when the product of the granularities indicated in C_1 (i.e., the maximum number of rules that can

be obtained) is greater than the number of input variables times the highest maximum granularity of the variables, then a variable is randomly selected to be removed from the antecedent part —its granularity is set to 1— until the previous condition is satisfied. This is done in order to avoid too complex solutions in the initialization stage —during the evolutionary learning this upper bound to the number of rules does not apply.

Crossover

The iterative part of the evolutionary algorithm starts with a binary tournament selection process. Then, two crossover operations can be applied: one-point crossover for exchanging the C_1 parts (it also exchanges the corresponding C_2 genes) and, when the C_1 parts are equal, the parent-centric BLX (PCBLX) [51] is used to crossover the C_2 part. In order to prevent the crossover of too similar individuals, an incest prevention was implemented. When the Euclidean distance of the lateral displacements is less than a particular threshold L, the individuals are not crossed.

Mutation

After the crossover, the mutation is applied to each offspring with probability p_{mut}. It can apply two possible operations with equal probability to a randomly selected gene of the C_1 part: i) decreasing the granularity by 1 or ii) increasing the granularity to a more specific granularity —all the granularities having the same chance. In order to calculate the new lateral displacements in the corresponding C_2 part, the algorithm uses the displacements of the two nearest split points of the previous granularity (before mutation) weighted by the distance between the split points.

Local Search

The offsprings are evaluated using the mechanism described in Sec. 4.4.2. Then, a local search is performed for each offspring, generating n_{ls} new C_1 parts with equal or less granularity —with equal probability— for each variable and the C_2 is generated randomly. The new chromosomes are decoded and evaluated and, if there is a solution that obtains better fitness, then it replaces the original individual. After that, the previous and current populations are merged, and the N best individuals are selected as the new population.

Incest Prevention and Restart Mechanism

The algorithm incorporates a restart mechanism that uses the incest prevention threshold L as a trigger. First, L is initialized as the maximum length of the C_2 part, i.e. the

product of the number of input variables times the largest maximum granularity of the variables, divided by 4. This implies that the incest prevention allows crossovers between individuals that have a distance higher than a quarter of the maximum euclidean distance. Then, for each iteration, L is decreased always by 0.4, and by 0.2 if there are no new individuals or the best individual does not change, in order to accelerate convergence. When L reaches 0, the population is restarted, and L is reinitialized. Only the best individual so far is kept, and the local search process is executed with the best individual in order to generate new individuals until the population is complete. When the restart criterion is fulfilled twice, the algorithm stops, i.e., one single restart is executed.

Evaluation

The evaluation process is described in the Evaluation box in Fig. 4.3. First, the real data base is obtained applying the displacements in C_2 to the fuzzy partitions selected in C_1. Then, the Wang & Mendel algorithm [113] is used to create the antecedent part of the rule base for each individual. The consequent part of the rules is learned using the Elastic Net method [119] in order to obtain the coefficients of the degree 1 polynomial for each rule. In order to solve the minimization problem of Elastic Net, an Stochastic Gradient Descent optimization technique was used [17, 110]. Only those examples in E_S^{map} are used to obtain the rule base from the codified chromosome. In this manner, those examples that are not representative are discarded for the rule generation.

Then, the resulting TSK-1 fuzzy rule base is evaluated using the following equation:

$$fitness = MSE(E_{tra}^{map}) = \frac{1}{2 \cdot |E_{tra}^{map}|} \sum_{i=1}^{|E_{tra}^{map}|} (F(x^i) - y^i)^2, \tag{4.15}$$

where E_{tra}^{map} is the full training dataset partition and $F(x^i)$ is the output obtained by the knowledge base for the input x^i. Using all the examples for evaluation can be seen, in some way, as a validation process, as the rule base was constructed with a subset of them (E_S^{map}).

4.4.3 Aggregation function

After the execution of FRULER for each dataset partition the reduction phase is applied in order to get the final solution. Each of the FRULER executions obtains a TSK-1 Knowledge Base, composed by a linguistic partition of the input variables — data base — and the TSK fuzzy rule set — rule base. These Knowledge Bases may be combined using an ensemble

technique, that takes into account the degree of fulfillment of the input data to each knowledge base, to calculate a weighted average of the outputs. However, this approach increases remarkably the complexity of the model, as the number of rules of the final solution increases with the number of partitions of the data.

Thus, to combine the solutions generated in the Map phase without increasing significantly the complexity, the properties of each of the obtained Knowledge Bases must be taken into account. On one hand, the rule base strongly depends on the label partitions of the data base. A change of granularity in one input variable partition can lead to a substantial change on the rule consequents. On the other hand, the granularity for each input variable can be easily combined, in a similar way to a crossover operation for integer valued individuals. This combination of granularities can take into account the variables not used in each dataset partition. After combining the granularities, the rule base can be generated using the ad-hoc TSK rule base Generation process of FRULER.

1: **function** AGGREGATION($S = \{s_1, s_2, \ldots, s_{n_{map}}\}, E_S = E_S^1 \cup E_S^2 \cup \cdots \cup E_S^{n_{map}}$)

2: **for** $i = 1, \ldots, n_{map}$ **do**

3: **for** $k = 1, \ldots, n_{map}$ **do**

4: $$r_{i,k} = \bigcup_{j=1,\ldots,p} \left\{ a_{i,k,j} \ : \ a_{i,k,j} = \begin{cases} s_{i,j} \text{ if } s_{i,j} \neq \text{ NULL} \\ s_{k,j} \text{if } s_{i,j} = \text{ NULL and } s_{k,j} \neq \text{ NULL} \\ 1 \quad \text{otherwise} \end{cases} \right\}$$

5: $R = \cup\, r_{i,k},\ i = 1, \ldots, n_{map},\ k = 1, \ldots, n_{map}$

6: $error_{min} = \infty$

7: **for each** $r \in R$ **do**

8: $rb =$ Generate rule base from r using E_S

9: **if** $MSE(rb, E_{tra}) < error_{min}$ **then**

10: $error_{min} = MSE(rb, E_{tra})$

11: $Best = rb$

12: **return** *Best*

Figure 4.5: Pseudocode of the Aggregation function.

Fig. 4.5 shows the pseudocode of the Aggregation function. The Aggregation function uses two parameters: S contains the solutions generated by FRULER in each dataset partition

as a list of granularities, and E_S contains the union of all the selected instances in each dataset partition. Thus, E_S can be seen as the selected instances of the entire training dataset if a stratification approach is used to perform the instance selection method taking each dataset partition as a stratum. The solution obtained in each dataset partition contains the displacements in addition to the granularity for each input variable [4].

The process is as follows: for each partition solution (line 2) the Aggregation function completes the input variables not previously selected in the dataset partition with information of the other solutions (line 3). Thus the Aggregation function generates a maximum of n_{map}^2 final solutions. This combination is done, for each input variable j, as follows (line 4):

- The granularity $s_{i,j}$ (and the displacements associated with it) is selected if it exists.
- If the value is not defined in s_i ($s_{i,j}$ = NULL) but it is in s_k, then $s_{k,j}$ is used.
- Finally, if the value is not defined neither s_i nor s_k, this variable is not considered (granularity 1).

Note that the inner loop (line 4) takes also into account s_i, thus the original solution is also kept. Then, R (line 7) is the set of the final solutions that combine the information of the different results obtained in each dataset partition.

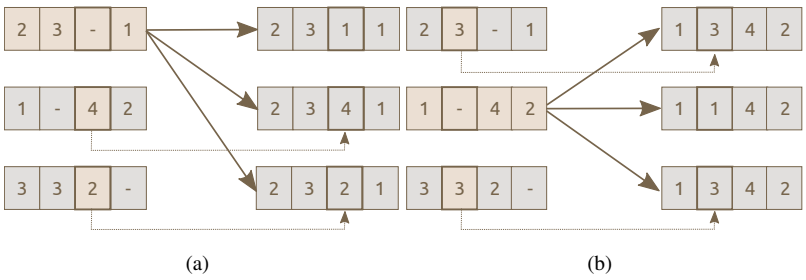


Figure 4.6: Visual illustration of the Aggregation function for three different solutions obtained in the Map phase.

Fig. 4.6 illustrates how the solutions obtained by three partitions are combined with this Aggregation function. In the (a) case, the third input variable was removed in the feature selection process and this information is completed with two choices: i) adding a granularity 1 to the variable (the variable is not used) and ii) the granularity indicated in the other solutions. In the (b) scenario, we show the particular case where the other solutions indicate the same

[4] This information was omitted in the pseudocode for the sake of clarity

granularity for the incomplete information of the selected solution. However, two solutions are created because the lateral displacement information may not be the same.

Once all the possible combinations have been obtained, the Aggregation function generates the associated rule bases (line 10) using the ad-hoc TSK rule base generation process of FRULER. This process uses the combination of selected instances for all dataset partitions E_S. Then, the error is measured using the whole training dataset E_{tra} (line 11), so a validation is performed due to the use of examples not seen in the TSK rule base generation process. Finally, the solution with the lowest error (lines 8-15) is selected as the best solution and returned by the Aggregation function (line 16).

4.5 Results

In order to analyze the performance of S-FRULER, two types of validation have been done: a) a comparison with other GFSs using 10 regression problems (Sec. 4.5.2) from the KEEL project repository [4]; and b) an application to a bioinformatics problem (Sec. 4.5.3), which contains more than 250,000 examples and the number of input variables range from 60 to 180 — available in the Interdisciplinary Computing and Complex BioSystems (ICOS) research group webpage [12]. Table 4.1 shows the characteristics of the regression datasets, with the number of instances ranging from 7,129 to 40,768 examples, and the number of input variables from 5 to 40. The datasets are sorted in incremental order of the number of variables.

Table 4.1: The 10 datasets of the experimental study.

Problem	Abbr.	# Variables	# Cases
Delta Ailerons	DELAIL	5	7,129
Delta Elevators	DELELV	6	9,517
California Housing	CAL	8	20,640
MV Artificial Domain	MV	10	40,768
House-16H	HOU	16	22,784
Elevators	ELV	18	16,559
Computer Activity	CA	21	8,192
Pole Telecommunications	POLE	26	14,998
Pumadyn	PUM	32	8,192
Ailerons	AIL	40	13,750

4.5.1 Experimental Setup

In terms of parameters, S-FRULER only adds to FRULER the parameter of the probability that a particular feature is selected in at least one dataset partition (α_{p_m}). For the experiments performed in this section $\alpha_{p_m} = 0.9$. Moreover, FRULER [95], which is embedded in S-FRULER, was designed to keep the number of parameters as low as possible. In the multi-granularity fuzzy discretization, the fuzziness parameter used for the generation of the fuzzy intervals was 1, i.e., the highest fuzziness value. For the instance selection technique, no parameters are needed. For the evolutionary algorithm, the values of the parameters were: population size $= 61$, maximum number of evaluations $= 100,000$, $p_{cross} = 1.0$, $p_{mut} = 0.2$, and the number of neighbours generated in the local search was $n_{ls} = 5$. For the generation of the TSK fuzzy rule bases, the weight of the tradeoff between ℓ_1 and ℓ_2 regularizations on the Elastic Net was $\alpha = 0.95$, and the regularization parameter λ was obtained from a grid search in the interval $[1, 1E-10]$. η^0 was obtained halving the initial value (0.1) until the result worsens.

S-FRULER[5] was developed entirely using Java version 8. The data partitioning (Sec. 4.4.1), Map Function (Sec. 4.4.2) and Aggregation function (Sec. 4.4.3) were implemented using Spark with standard functions.

For each dataset, we performed 10 trials (with different seeds for the random number generation) of S-FRULER. For each trial, the dataset was divided randomly into training (80%) and test (20%). The results shown in the next section are the mean values over all the runs. The runtimes have been obtained in two different ways:

- Using the Spark Standalone mode, where the workers are simulated as threads, executed in an HP Proliant composed by four processors AMD Opteron 6262 HE with a total of 64 cores and 128 GB of memory.
- Using the Spark Cluster mode executed in an Amazon Elastic MapReduce (EMR) 4.0.0 that deploys an Apache Hadoop NextGen MapReduce (YARN) system and uses m3.xlarge machines (Intel Xeon E5-2670 v2 with 4 cores and 15 GB of memory). For each dataset, a different configuration on the number of workers was used depending on the number of dataset partitions, assuring that there is at least one core for each dataset partition.

[5]The software is available at `http://tec.citius.usc.es/fruler/`

4.5.2 Statistical Analysis

In order to evaluate the models learned by S-FRULER, we compared them with three of the most accurate genetic GFSs for regression in the literature[6]:

- FS_{MOGFS}^{e}+TUN^{e} [3]: a multi-objective evolutionary algorithm that learns Mamdani fuzzy rule bases. This algorithm learns the granularities from uniform multi-granularity fuzzy partitions (up to granularity 7) and the lateral displacement of the labels. It includes a post-processing algorithm for tuning the parameters of the membership functions and for rule selection.
- L-METSK-HD^{e} [39]: a multi-objective evolutionary algorithm that learns linguistic TSK-0 fuzzy rule bases. The algorithm learns the granularities from uniform multi-granularity fuzzy partitions (up to granularity 7).
- A-METSK-HD^{e} [39]: a multi-objective evolutionary algorithm that learns approximative TSK-1 fuzzy rule bases. The algorithm starts with the solution obtained on the first stage and applies a tuning of the membership functions, rule selection and a Kalman-based calculation of the consequents of the rules.

Tables 4.2 and 4.3 show the results obtained by S-FRULER and the other three GFSs for the datasets in Table 4.1 for precision — the mean test error over the executions — and complexity — the number of rules (# Rules) of the final solution. Moreover, in order to analyze statistically the difference between the different approaches, a Friedman ranking test followed by a Holm post-hoc test were performed for both measures. The Friedman ranking test calculates a rank for each approach based on its performance, where the lower the rank, the better. On the other hand, the Holm post-hoc method tests if the difference between two rankings is significant. We applied the Holm post-hoc test using the S-FRULER approach as a control method, thus calculating the p-value for each comparison of S-FRULER with each of the other approaches.

In the case of precision (Table 4.2), both S-FRULER and A-METSK-HD^{e} achieve the best result in 5 of the datasets. A-METSK-HD^{e} obtains the lowest rank (1.5) followed by S-FRULER (1.8), however the difference between them is rather low, since the Holm p-value is high (0.603). FS_{MOGFS}^{e}+TUN^{e} and L-METSK-HD^{e} obtain worse errors than S-FRULER (ranks 3 and 3.7 respectively) and the differences are statistically significant (Holm p-values

[6] Although FRULER [95] has a better accuracy than these GFSs, we excluded it from the comparison, since S-FRULER is based on FRULER.

Table 4.2: Average test errors for the different algorithms. The errors in this table should be multiplied by $10^{-8}, 10^{-6}, 10^{9}, 10^{8}, 10^{-6}, 10^{-4}, 10^{-8}$ in the case of DELAIL, DELELV, CAL, HOU, ELV, PUM, AIL respectively. The best results are marked in bold face. It also shows the Friedman Ranking for each approach and the Holm adjusted p-value comparing S-FRULER with the other algorithms.

Algorithms	S-FRULER	A-METSK-HD^{e}	FS$_{MOGFS}$^{e}+TUN^{e}	L-METSK-HD^{e}
DELAIL	1.44	**1.40**	1.53	1.62
DELELV	1.12	**1.03**	1.09	1.12
CAL	2.18	**1.71**	2.95	2.64
MV	**0.05**	0.06	0.16	0.25
HOU	**8.2**	8.5	9.4	10.4
ELV	**3.2**	7.0	9.0	8.9
CA	**4.6**	5.0	5.2	5.9
POLE	124	**61**	103	151
PUM	0.349	**0.287**	0.292	0.594
AIL	**1.4**	1.5	2.0	1.8
Friedman Ranking	1.8	1.5	3	3.7
Holm p-value		0.603	0.075	0.003

below 0.1).

In terms of complexity (Table 4.3), S-FRULER obtains the best result in 9 of 10 datasets, while FS$_{MOGFS}$^{e}+TUN^{e} obtains the best result in the remaining dataset (POLE). The Friedman ranking test shows these results, where S-FRULER obtains the lowest rank (1.1) followed by FS$_{MOGFS}$^{e}+TUN^{e} (1.9), while A-METSK-HD^{e} and L-METSK-HD^{e} obtain the worst ranking (3 and 4 respectively). Comparing S-FRULER with FS$_{MOGFS}$^{e}+TUN^{e}, the Holm p-value is rather low (0.166) showing a noticeable difference, while the difference with A-METSK-HD^{e} and L-METSK-HD^{e} is remarkable (p-value below 0.005).

Finally, we compare S-FRULER with FRULER [95] in terms of runtime of the full learning process in order to highlight the obtained speedup. Table 4.4 shows the runtimes of FRULER and S-FRULER and also shows the speedup obtained in each execution mode of S-FRULER (Standalone using threads or in a Cluster) when compared with FRULER and the number of dataset partitions n_{map} obtained for each dataset. We also show the runtime with A-METSK-HD^{e}, the single GFS which is not statistically different from S-FRULER (Table

Table 4.3: Average number of rules for the different algorithms. The best results are marked in bold face. It also shows the Friedman Ranking for each approach and the Holm adjusted p-value comparing S-FRULER with the other algorithms.

Algorithms	S-FRULER	A-METSK-HD^e	$FS_{MOGFS}{}^e$+TUN^e	L-METSK-HD^e
DELAIL	**3**	37	6	98
DELELV	**2**	39.1	7.9	91
CAL	**7**	56	8	100
MV	**4**	56	14	76
HOU	**11**	30	12	69
ELV	**5**	35	8	76
CA	**11**	32	14	71
POLE	20	46	**13**	100
PUM	**4**	63	18	88
AIL	**11**	48	15	99
Friedman Ranking	1.1	3	1.9	4
Holm p-value		0.002	0.166	$<$ 1E-3

4.2).

It can be seen that the number of dataset partitions depends on the problem, but, in general, as the number of variables increases so does the number of dataset partitions used. S-FRULER, in both Standalone (threads) and Cluster, obtains lower runtimes than FRULER, as expected. The speedup depends on the problem, as datasets with a similar number of dataset partitions (e.g., DELAIL, DELELV, CAL and MV with 21-23 partitions) have a very different speedup (8, 15, 27 and 8 respectively in Standalone or 12, 21, 35 and 13 in Cluster mode). It is worth emphasizing that for the problems with the worst runtime in FRULER (HOU, ELV, PUM and AIL), S-FRULER achieves speedups far above the parallelization level given by the number of dataset partitions. This is because the scalability obtained by S-FRULER is not only given by the distributed approach, but also due to the faster convergence of FRULER in each dataset partition as the partitioned problem is more simple.

These results show that S-FRULER not only reduces FRULER runtimes, but also is able to learn the simplest models in terms of number of rules, using a linguistic approach, while maintaining a high precision comparable to an approximative approach.

Table 4.4: Runtime comparison between FRULER, S-FRULER and A-METSK-HD^e (the single GFS which is not statistically different from S-FRULER —Table 4.2). All times are formatted in hours:minutes:seconds. Runtimes of A-METSK-HD^e were obtained using a different computer (Intel Core 2 Quad Q9550 2.83GHz, 8GB RAM).

Algorithms	FRULER	S-FRULER	Standalone		Cluster		A-METSK-HD^e
	Time	n_{map}	Time	Speedup	Time	Speedup	Time
DELAIL	0:09:58	21	0:01:18	8	0:00:48	12	2:30:39
DELELV	0:25:01	22	0:01:38	15	0:01:13	21	1:29:30
CAL	1:57:03	23	0:04:20	27	0:03:22	35	5:13:28
MV	1:17:02	23	0:09:27	8	0:05:49	13	3:17:54
HOU	4:15:17	25	0:04:06	62	0:03:09	81	5:07:58
ELV	3:01:30	26	0:03:10	57	0:03:14	56	3:06:58
CA	0:38:12	25	0:03:46	10	0:01:48	21	3:37:49
POLE	1:53:15	27	0:10:20	11	0:05:14	22	4:40:22
PUM	31:14:27	24	0:01:58	956	0:01:39	1,139	2:22:25
AIL	12:50:38	28	0:07:13	107	0:03:32	218	5:26:30

4.5.3 Application to Bioinformatics

As stated in the introduction, the large scale regression problems solved with GFSs in the literature are still far from those considered Big Data problems in terms of volume and velocity. A high computational cost when the X matrix (Eq. 4.6) is huge, due to number of examples and/or number of variables, can be solved with parallelization approaches, e.g. solving several X matrix at the same time. However, if the storage demand of various X cannot be stored in memory of a single processor, it is necessary the use of Big Data approaches, such as distributed computing. In order to demonstrate the capability of S-FRULER in a challenging domain, we have applied our approach to a real bioinformatics problem with much higher computational requirements than the datasets of the previous section.

The problem is focused on one of the main open problems in computational biology which is the prediction of the 3D structure of protein chains [13]. One approach to this problem is to predict some attributes of a protein, such as the secondary structure, the solvent accessibility or the coordination number (CN), and then integrate this knowledge into a full 3D structure predictor. Particularly, the CN problem is defined as the prediction, for a given residue, of the number of residues of the same protein that are in contact with it in the native state.

Two residues are said to be in contact when the distance between them is below a certain threshold. Figure 4.7 shows a graphical representation of the CN — in this case the CN is 4— for a particular residue, discarding trivial contacts.

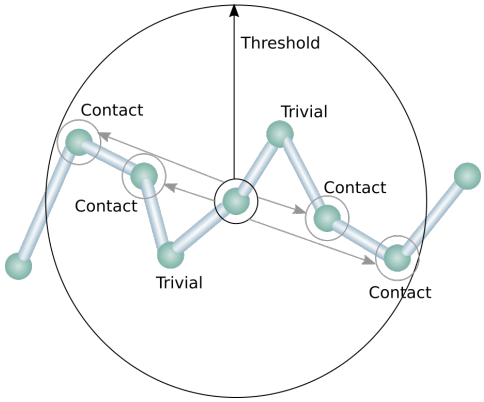


Figure 4.7: Graphical representation of the CN of a residue [13].

In [64] was proposed a real-valued definition of contact, and linear regression is used to predict the CN based on the amino acid type of the protein primary sequence and global information about the protein. The amino acid information can be extracted from the primary sequence, where one amino acid is defined as one nominal variable with 20 possible values. However, a Position-Specific Scoring Matrices (PSSM) representation derived from the primary sequence can be used to represent the amino acid information. The PSSM representation is an statistical profile of the primary sequence that takes into account how this sequence may have evolved. Thus, each amino acid is defined as 20 continuous variables.

The amino acids used to predict the CN can be extracted from a local context (a windows of amino acids) of the target in the chain. Therefore, the number of variables used to solve the problem can increase in blocks of 40 features as the window expands. For example, with the first neighbors two amino acids are added (before and after the target), each with 20 new features. Given these characteristics, in [105] this problem is proposed as a benchmark for testing the scalability of regression methods. In this manner, different datasets can be constructed adjusting the number of examples and/or variables used. The available datasets can be downloaded from the PSP Benchmark project web site [12].

S-FRULER was used to solve four versions of this dataset with different number of vari-

ables, depending on the size of the window of amino acids considered (Table 4.5). We have tested FRULER on these datasets, but it fails to converge —in less than three days— in the three most complex problems. As the runtime of FRULER is usually lower than the GFSs used for the comparisons in the previous section [39], it can be assumed that these GFSs would also fail on these datasets. We compared S-FRULER with the regression methods implemented in the Spark scalable machine learning library MLlib [78]. The methods available in this resource include Ridge Regression (ℓ_2 regularization) and Lasso Regression (ℓ_1 regularization), both solved using distributed mini-batch SGD. The regularization parameter λ was obtained in the same way as S-FRULER (see [95] for more details). The MLlib results presented in Table 4.5 correspond to 1,500 iterations. Nevertheless these methods were run for $3,000$ iterations with no further improvement.

Table 4.5: Datasets characteristics for the four CN problems, Mean Squared Error (MSE), and runtime (hours:minutes:seconds) for S-FRULER, Ridge and Lasso MLlib implementations.

			S-FRULER			Ridge SGD		Lasso SGD	
Dataset	# Cases	# Vars	n_{map}	Test Error	Time	Test Error	Time	Test Error	Time
w1	257,560	60	33	**12.45**	04:42:08	15.09	**00:42:56**	19.01	00:45:05
w2	257,560	100	34	**12.15**	05:32:33	14.20	**01:00:44**	18.91	01:01:25
w3	257,560	140	35	**12.23**	05:48:42	14.18	**01:04:36**	18.89	01:04:43
w4	257,560	180	36	**12.30**	12:17:42	13.62	01:06:42	18.93	**01:05:46**

Table 4.5 shows the characteristics of these four datasets, with the mean squared test error obtained by S-FRULER, Ridge and Lasso, and their runtime using an HP Proliant composed of four processors AMD Opteron 6262 HE with a total of 64 cores and 128 GB of memory. The test error in S-FRULER is lower than Ridge and Lasso approaches for all the datasets. Furthermore, the precision obtained with the less informative dataset (w1) is better than the result obtained by Ridge and Lasso with the dataset with more information (w4). Since the error of S-FRULER is always better than the other approaches —11-21% better than Ridge and 53-56% better than Lasso—, a statistical test is not necessary to confirm that S-FRULER performs significantly the best. The highest test error using S-FRULER occurs for w1, as it is the dataset with less information. On the other hand, when the number of input variables is really high (w4), the test error worsens very slightly when compared to w3. In the case of Ridge, the error improves even with the largest dataset (w4), since it uses all the variables in

some degree. However, this has the drawback of increasing the complexity of the obtained model, while a low complexity model based on rules can be used by the expert. Regarding the runtimes, S-FRULER always takes longer times to get its best model. We have executed Ridge and Lasso for longer runtimes (comparable to those of S-FRULER) but they did not further improve.

We have also compared S-FRULER with four approaches presented in [18], where the regression problem was transformed into a classification problem with two classes using an uniform-frequency discretization. The algorithms in the comparison [18] are:

- GAssist[10]: is a Pittsburgh-style learning classifier system (LCS). It uses a standard genetic algorithm to evolve a population of individuals, each of them being a complete and variable-length rule set.
- BioHEL[11]: is an evolutionary learning system that follows the iterative rule learning approach and is designed to handle large-scale bioinformatic datasets.
- C4.5[87]: builds decision trees from a set of training data using the concept of information entropy. At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets that maximize the presence of one class.
- PART[37]: iteratively builds a partial C4.5 decision tree in each iteration and transforms the most promising leaf into a rule.

Table 4.6: Comparison of error (in percentage) for the four CN problems discretized into 2 classes. The Friedman ranking and the p-value were obtained by comparing each approach with S-FRULER.

Algorithm	S-FRULER	BioHEL	GAssist	PART	C4.5
w1	**23.1**	24.2	25.2	29.1	31.4
w2	**22.7**	24.0	25.3	29.1	31.4
w3	**22.3**	23.6	25.4	30.1	31.9
w4	**22.6**	23.5	25.2	24	31.8
Friedman Ranking	1	2	3.25	3.75	5
Holm Adj. p-value		0.371	0.088	0.042	0.001

In order to compare the regression models generated by S-FRULER with the classification models [18], the output of the regression model is discretized using the same uniform-frequency discretization that transformed the regression problem into a classification one.

Table 4.6 shows the error (in percentage) obtained by S-FRULER compared to the other four approaches. It also displays the Friedman ranking test followed by a Holm post-hoc test comparing S-FRULER with the other approaches. In this particular case, the statistical tests are less reliable due to the low ratio between the number of datasets and the number of algorithms. In spite of this, we show on Tables 4.6 and 4.7 the p-values for the sake of completeness. S-FRULER obtains the best result in all the datasets and gets the best ranking in the Friedman test, followed by the other genetic approaches BioHEL and GAssisst.

Table 4.7: Comparison of complexity (number of rules/leaf nodes) for the four CN problems discretized into 2 classes. The Friedman ranking and the p-value were obtained by comparing each approach with S-FRULER.

Algorithm	S-FRULER	BioHEL	GAssist	PART	C4.5
w1	**31.6**	110.0	40.2	6,271.9	22,662.1
w2	**16.2**	110.9	36.5	7,889.5	22,144.3
w3	**7.6**	113.0	35.4	7,450.8	21,342.7
w4	52.7	113.6	**36.5**	7,006.0	20,671.1
Friedman Ranking	1.25	3	1.75	4	5
Holm Adj. p-value		0.235	0.655	0.042	0.003

In order to compare the complexity of the models, the number of rules of the resulting models are shown in table 4.7. In the case of C4.5, the complexity is measured as the number of leaf nodes. S-FRULER obtains the best results in three datasets (w1, w2 and w3) and the lowest ranking (1.25) while GAssist obtains the best result for w4 and the second lowest ranking (1.75).

The results presented in this section show that S-FRULER can cope with large datasets in both number of examples and number of attributes. It was also demonstrated that the models learned by S-FRULER obtain the best results in both accuracy and complexity when compared with other state-of-the-art techniques

4.6 Conclusions

In this paper, we have presented an scalable version of FRULER [95] —a GFS that learns simple and linguistic TSK-1 knowledge bases for regression problems—, called S-FRULER.

S-FRULER obtains simple models with high precision but reducing the runtime of FRULER. The approach is based on a distributed computing methodology and was implemented using the Spark software, thus providing both standalone and cluster modes. S-FRULER partitions the dataset into small sets, which are more tractable, and executes FRULER for each of them. Also, for each dataset partition, a random feature selection process to reduce the number of variables is used. After the Map phase, the method implements an aggregation function to obtain linguistic TSK-1 fuzzy rule bases from the rule bases generated in each dataset partition. The use of Spark facilitates the building of parallel applications that can run on different cluster solutions, e.g. Hadoop YARN, providing some Big Data desired properties like fault tolerance. Nevertheless, further improvements may be applied to S-FRULER in order to take advantage of this type of architecture, such as adopting a scheme more similar to the MapReduce approach.

S-FRULER has speedups usually larger than the number of dataset partitions used, showing an scalability higher than linear in both standalone and cluster mode. It was compared with three GFSs in terms of complexity of the models (number of rules) and precision (mean squared error), showing good results with less complex models. S-FRULER was also applied to a large-scale problem in computational biology: the prediction of the coordinate number of a residue in the context of prediction of the 3D structure of protein chains. Results demonstrate the capability of S-FRULER to obtain precise and simple models in large scale problems.

CHAPTER 5

CONCLUSIONS

In this PhD dissertation the problem of automatic learning of FRBSs through the use of GFSs to solve regression problems has been addressed. Particularly, the work has focused on designing GFSs that obtain models with low complexity while maintaining high precision without using expert-knowledge about the problem to be solved.

In the field of mobile robotics, a new algorithm was proposed, called Iterative Quantified Fuzzy Rule Learning (IQFRL), whose main contributions are:

- It is based on the Iterative Rule Learning approach and uses Genetic Programming to define the valid structures of the fuzzy rules.
- IQFRL is able to learn controllers with embedded preprocessing without expert knowledge.
- The transformation of the low-level variables into high-level variables is done through the use of Quantified Fuzzy Propositions and Rules.
- The algorithm involves linguistic labels defined by multiple granularity without limiting the granularity levels.

Furthermore, the algorithm was extensively tested with the wall-following behavior both in several simulated environments and on a *Pioneer 3-AT* robot in two real environments. The results were compared with some of the most well-known algorithms for learning controllers in mobile robotics. Non-parametric significance tests have been performed, showing very good results and a statistically significant performance of the IQFRL approach.

A novel genetic fuzzy system called FRULER was also presented. The major contribu-

tions of this approach are the following:

- FRULER learns simple and linguistic TSK-1 knowledge bases for regression problems using a Pittsburgh GFS approach.
- This new approach has two general-purpose preprocessing stages for regression problems:
  - A new instance selection algorithm for regression that helps the genetic algorithm to focus the search on models with lower complexity and, therefore, with better generalization.
  - A novel non-uniform multi-granularity fuzzy discretization, which allows to generate linguistic data bases without bounding the maximum number of labels.
- The evolutionary algorithm incorporates an automatic generation of the TSK fuzzy rule bases through Elastic Net, solved using Stochastic Gradient Descent, in order to obtain consequents with low overfitting.

FRULER was compared with three state of the art algorithms that learn different types of fuzzy rules: linguistic Mamdani, linguistic TSK-0 and approximative TSK-1. The results were analyzed using statistical tests, which show that FRULER obtains high accuracy, but with a lower number of rules and with a linguistic data base. This is of particular interest in problems where both high accuracy and interpretability are demanded, in order to provide qualitative understanding of the model to the users.

Finally, S-FRULER was presented in order to improve the scalability of FRULER. S-FRULER obtains low-complex models with high precision but reducing the runtime of FRULER. The main contributions of S-FRULER are as follows:

- S-FRULER is based on the MapReduce methodology and was implemented using the Spark software, thus providing both standalone and cluster modes.
- S-FRULER partitions the dataset into small sets, which are more tractable, and executes FRULER for each of them.
- For each dataset partition, a random feature selection process to reduce the number of variables is used.
- After the Map phase, the method implements an aggregation function to obtain linguistic TSK-1 fuzzy rule bases from the rule bases generated in each dataset partition.

S-FRULER has speedups usually larger than the number of dataset partitions used, show-

ing an scalability higher than linear in both standalone and cluster mode. It was compared with three GFSs in terms of complexity of the models (number of rules) and precision (mean squared error), showing good results with less complex models. S-FRULER was also applied to a large-scale problem in computational biology: the prediction of the coordinate number of a residue in the context of the prediction of the 3D structure of protein chains. Results demonstrate the capability of S-FRULER to obtain precise and simple models in large scale problems.

The research accomplished in this thesis leads to a number of interesting new developments that could be taken into consideration to continue as a future work:

- The approaches proposed in this thesis might be adapted to work with nominal attributes in order to be used in other fields where categorical data is provided.
- A trending topic in Big Data and predictive analytics is the shift from batch processing to streaming data. GFS are computationally expensive and are more suited for batch processing. However, the generated FRBs can be updated and corrected when new data is badly predicted. This approach can lead to a more operational solution for long-term predictive systems.

Bibliography

[1] Alejandro Agostini and Enric Celaya. Reinforcement learning for robot control using probability density estimations. In *Proceedings of the 7th International Conference on Informatics in Control (ICINCO)*, pages 160–168, 2010.

[2] Rafael Alcalá, Jesús Alcalá-Fdez, Francisco Herrera, and José Otero. Genetic learning of accurate and compact fuzzy rule based systems based on the 2-tuples linguistic representation. *International Journal of Approximate Reasoning*, 44(1):45–64, 2007.

[3] Rafael Alcalá, María José Gacto, and Francisco Herrera. A fast and scalable multiobjective genetic fuzzy system for linguistic fuzzy modeling in high-dimensional regression problems. *IEEE Transactions on Fuzzy Systems*, 19(4):666–681, 2011.

[4] Jesus Alcala-Fdez, Luciano Sanchez, Salvador Garcia, Maria Jose del Jesus, Sebastian Ventura, J.M. Garrell, Jose Otero, Cristobal Romero, Jaume Bacardit, and Victor M. Rivas. KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2009.

[5] Jesus Alcala-Fdez, Luciano Sanchez, Salvador Garcia, Maria Jose del Jesus, Sebastian Ventura, JM Garrell, Jose Otero, Cristobal Romero, Jaume Bacardit, Victor M Rivas, et al. Keel: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3):307–318, 2009.

[6] Jose M. Alonso, Ciro Castiello, and Corrado Mencar. Interpretability of fuzzy systems: Current research trends and prospects. In *Springer Handbook of Computational Intelligence*, pages 219–237. Springer, 2015.

[7] Jose M. Alonso and Luis Magdalena. Special issue on interpretable fuzzy systems. *Information Sciences*, 181(20):4331–4339, 2011.

[8] Michela Antonelli, Pietro Ducange, and Francesco Marcelloni. An efficient multi-objective evolutionary fuzzy system for regression problems. *International Journal of Approximate Reasoning*, 54(9):1434–1451, 2013.

[9] Ahmad Taher Azar and Aboul Ella Hassanien. Dimensionality reduction of medical big data using neural-fuzzy classifier. *Soft computing*, 19(4):1115–1127, 2015.

[10] Jaume Bacardit. *Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time*. PhD thesis, Ramon Llull University, Barcelona, Spain, 2004.

[11] Jaume Bacardit, Edmund K. Burke, and Natalio Krasnogor. Improving the scalability of rule-based evolutionary learning. *Memetic Computing*, 1(1):55–67, March 2009.

[12] Jaume Bacardit and Natalio Krasnogor. The ICOS PSP benchmarks repository, 2008. "`http://ico2s.org/datasets/psp_benchmark.html`, last visit: 26/08/2016".

[13] Jaume Bacardit, Michael Stout, Natalio Krasnogor, Jonathan D. Hirst, and Jacek Blazewicz. Coordination number prediction using learning classifier systems. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06*, page 247, Seattle, Washington, USA, 2006.

[14] Arie Ben-David. A lot of randomness is hiding in accuracy. *Engineering Applications of Artificial Intelligence*, 20(7):875–885, 2007.

[15] Christopher M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[16] Bert Bonte and Bart Wyns. Automatically designing robot controllers and sensor morphology with genetic programming. In *Proceedings of the 6th IFIP Artificial Intelligence Applications and Innovations (AIAI)*, pages 86–93, 2010.

[17] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the 19th International Conference on Computational Statistics*, pages 177–186. Springer, 2010.

[18] Dan Andrei Calian and Jaume Bacardit. Integrating memetic search into the BioHEL evolutionary learning system for large-scale datasets. *Memetic Computing*, 5(2):95–130, 2013.

[19] Adrián Canosa, Ismael Rodríguez-Fdez, Manuel Mucientes, and Alberto Bugarín. STAC: Statistical Tests for Algorithms Comparison, 2015. "`https://citius.usc.es/transferencia/demostradores-tecnoloxicos/stac`, last visit: 24/08/2016".

[20] Jorge Casillas, Oscar Cordón, Francisco Herrera Triguero, and Luis Magdalena. *Accuracy improvements in linguistic fuzzy modeling*, volume 129. Springer, 2013.

[21] Jorge Casillas, Oscar Cordón, Francisco Herrera Triguero, and Luis Magdalena. *Interpretability issues in fuzzy modeling*, volume 128. Springer, 2013.

[22] David Chaves, Ismael Rodríguez-Fdez, Manuel Mucientes, and Alberto Bugarín. TSK-View: TSK fuzzy rule bases viewer, 2016. "`https://citius.usc.es/transferencia/demostradores-tecnoloxicos/fruler`, last visit: 24/08/2016".

[23] Oscar Cordon, Fernando Gomide, Francisco Herrera, F. Hoffmann, and Luis Magdalena. Ten years of genetic fuzzy systems: current framework and new trends. *Fuzzy sets and systems*, 141(1):5–31, 2004.

[24] Oscar Cordón and Francisco Herrera. A three-stage evolutionary process for learning descriptive and approximate fuzzy-logic-controller knowledge bases from examples. *International Journal of Approximate Reasoning*, 17(4):369–407, 1997.

[25] Oscar Cordón and Francisco Herrera. Hybridizing genetic algorithms with sharing scheme and evolution strategies for designing approximate fuzzy rule-based systems. *Fuzzy sets and systems*, 118(2):235–255, 2001.

[26] Oscar Cordón, Francisco Herrera, Frank Hoffmann, and Luis Magdalena. *Genetic fuzzy systems*. World Scientific Publishing Company Singapore, 2001.

[27] Oscar Cordón, Francisco Herrera, Frank Hoffmann, and Luis Magdalena. *Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases*, volume 19. World Scientific, 2001.

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

[29] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the twlefth international conef-erence on Machine learning*, volume 12, pages 194–202, 1995.

[30] Pietro Ducange, Francesco Marcelloni, and Armando Segatori. A mapreduce-based fuzzy associative classifier for big data. In *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8. IEEE, 2015.

[31] Agoston E. Eiben and James E. Smith. *Introduction to evolutionary computing*. Springer-Verlag, 2003.

[32] Michela Fazzolari, Rafael Alcalá, and Francisco Herrera. A multi-objective evolutionary method for learning granularities based on fuzzy discretization to improve the accuracy-complexity trade-off of fuzzy rule-based classification systems: D-MOFARC algorithm. *Applied Soft Computing*, 24:470–481, 2014.

[33] Michela Fazzolari, Rafael Alcalá, Yusuke Nojima, Hisao Ishibuchi, and Francisco Herrera. A review of the application of multiobjective evolutionary fuzzy systems: Current status and further directions. *IEEE Transactions on Fuzzy Systems*, 21(1):45–65, 2013.

[34] Michela Fazzolari, Bruno Giglio, Rafael Alcalá, Francesco Marcelloni, and Francisco Herrera. A study on the application of instance selection techniques in genetic fuzzy rule-based classification systems: Accuracy-complexity trade-off. *Knowledge-Based Systems*, 54:32–41, 2013.

[35] Alberto Fernández, Cristobal José Carmona, María José del Jesus, and Francisco Herrera. A view on fuzzy systems for big data: Progress and opportunities. *International Journal of Computational Intelligence Systems*, 9(sup1):69–80, 2016.

[36] Alberto Fernández, Victoria López, María José del Jesus, and Francisco Herrera. Revisiting Evolutionary Fuzzy Systems: Taxonomy, Applications, New Trends and Challenges. *Knowledge-Based Systems*, 2015.

[37] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In J. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann, 1998.

[38] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[39] María José Gacto, Marta Galende, Rafael Alcalá, and F Herrera. METSK-HDe: A multiobjective evolutionary algorithm to learn accurate TSK-fuzzy systems in high-dimensional and large-scale regression problems. *Information Sciences*, 276:63–79, 2014.

[40] Cristina Gamallo, Carlos V. Regueiro, Pablo Quintía, and Manuel Mucientes. Omnivision-based kld-Monte Carlo localization. *Robotics and Autonomous Systems*, 58(3):295–305, 2010.

[41] Salvador Garcia, Joaquín Derrac, José Ramón Cano, and Francisco Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435, 2012.

[42] Salvador Garcia, Julián Luengo, José Antonio Sáez, Victoria López, and Francisco Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750, 2013.

[43] Nicolás García-Pedrajas and Aida de Haro-García. Scaling up data mining algorithms: review and taxonomy. *Progress in Artificial Intelligence*, 1(1):71–87, 2012.

[44] Brian Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*, pages 317–323, 2003.

[45] Adrián González-Sieira, Manuel Mucientes, and Alberto Bugarín. A State Lattice Approach for Motion Planning under Control and Sensor Uncertainty. In *Proceedings of the First Iberian Robotics Conference (ROBOT)*, pages 247–260, Madrid (Spain), 2013.

[46] David Perry Greene and Stephen F. Smith. Competition-based induction of decision models from examples. *Machine Learning*, 13(2):229–257, 1993.

[47] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and Hastie. *The elements of statistical learning*. Springer, 2009.

[48] Timothy C. Havens, James C. Bezdek, Christopher Leckie, Lawrence O. Hall, and Marimuthu Palaniswami. Fuzzy c-means algorithms for very large data. *IEEE Transactions on Fuzzy Systems*, 20(6):1130–1146, 2012.

[49] Francisco Herrera. Genetic fuzzy systems: taxonomy, current research trends and prospects. *Evolutionary Intelligence*, 1(1):27–46, 2008.

[50] Francisco Herrera. Genetic fuzzy systems: taxonomy, current research trends and prospects. *Evolutionary Intelligence*, 1(1):27–46, 2008.

[51] Francisco Herrera, Manuel Lozano, and Ana M. Sánchez. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems*, 18(3):309–338, 2003.

[52] Francisco Herrera and Luis Martínez. A 2-tuple fuzzy linguistic representation model for computing with words. *IEEE Transactions on Fuzzy Systems*, 8(6):746–752, 2000.

[53] Chia-Hung Hsu and Chia-Feng Juang. Evolutionary robot wall-following control using type-2 fuzzy controller with species-DE-activated continuous ACO. *IEEE Transactions on Fuzzy Systems*, 21(1):100–112, 2013.

[54] Hisao Ishibuchi and Yusuke Nojima. Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning. *International Journal of Approximate Reasoning*, 44(1):4–31, 2007.

[55] Hisao Ishibuchi and Takashi Yamamoto. Performance evaluation of fuzzy partitions with different fuzzification grades. In *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE).*, volume 2, pages 1198–1203. IEEE, 2002.

[56] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Springer, 2013.

[57] Jyh-Shing Roger Jang. Anfis: adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man and Cybernetics*, 23(3):665–685, 1993.

[58] Chia-Feng Juang and Yu-Cheng Chang. Evolutionary-Group-Based Particle-Swarm-Optimized Fuzzy Controller With Application to Mobile-Robot Navigation in Unknown Environments. *IEEE Transactions on Fuzzy Systems*, 19(2):379–392, 2011.

[59] Chia-Feng Juang and Chia-Hung Hsu. Reinforcement ant optimized fuzzy controller for mobile-robot wall-following control. *IEEE Transactions on Industrial Electronics*, 56(10):3931–3940, 2009.

[60] Alexandros Karatzoglou, Alex Smola, Kurt Hornik, and Achim Zeileis. kernlab - An S4 Package for Kernel Methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

[61] Chuck Karr. Genetic algorithms for fuzzy controllers. *Ai Expert*, 6(2):26–33, 1991.

[62] M. Yousefi Azar Khanian, Ahmad Fakharian, M. Godarzvand Chegini, and Bahram Jozi. An intelligent fuzzy controller based on genetic algorithms. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 486–491, 2009.

[63] Gunhee Kim, Woojin Chung, Kyung-Rock Kim, Munsang Kim, Sangmok Han, and Richard H. Shinn. The autonomous tour-guide robot jinny. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*, volume 4, pages 3450–3455. IEEE, 2004.

[64] Akira R Kinjo, Katsuhisa Horimoto, and Ken Nishikawa. Predicting absolute contact numbers of native protein structure from amino acid sequence. *Proteins: Structure, Function, and Bioinformatics*, 58(1):158–165, 2005.

[65] Toshiyuki Kondo. Evolutionary design and behavior analysis of neuromodulatory neural networks for mobile robots control. *Applied Soft Computing*, 7(1):189–202, 2007.

[66] Rainer Kümmerle, Michael Ruhnke, Bastian Steder, Cyrill Stachniss, and Wolfram Burgard. A Navigation System for Robots Operating in Crowded Urban Environments. In *Proccedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2013.

[67] Jong-Yih Kuo and Yuan Cheng Ou. An Evolutionary Fuzzy Behaviour Controller Using Genetic Algorithm in RoboCup Soccer Game. In *Proceedings of the Ninth International Conference on Hybrid Intelligent Systems (HIS)*, volume 1, pages 281–286, 2009.

[68] Chi Wen Lo, Kun Lin Wu, Yue Chen Lin, and Jing Sin Liu. An intelligent control system for mobile robot navigation tasks in surveillance. In *Robot Intelligence Technology and Applications 2*, pages 449–462. Springer, 2014.

[69] Joaquín López, Diego Pérez, Enrique Paz, and Alejandro Santana. WatchBot: A building maintenance and surveillance system based on autonomous robots. *Robotics and Autonomous Systems*, 61(12):1559–1571, 2013.

[70] Victoria López, Sara del Río, José Manuel Benítez, and Francisco Herrera. Cost-sensitive linguistic fuzzy rule based classification systems under the mapreduce framework for imbalanced big data. *Fuzzy Sets and Systems*, 258:5–38, 2015.

[71] H.R. Lourenço, O.C. Martin, and T. Stützle. *Handbook of Metaheuristics*, chapter Iterated Local Search, pages 321–353. Kluwer Academic Publishers, 2003.

[72] Shingo Mabu, Andre Tjahjadi, Siti Sendari, and Kotaro Hirasawa. Evaluation on the robustness of Genetic Network Programming with reinforcement learning. In *Proceedings of the IEEE International Conference on Systems Man and Cybernetics (SMC)*, pages 1659–1664, 2010.

[73] Ebrahim H. Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. In *Proceedings of the Institution of Electrical Engineers*, volume 121, pages 1585–1588. IET, 1974.

[74] Ebrahim H. Mamdani and Sedrak Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of man-machine studies*, 7(1):1–13, 1975.

[75] Elena Marchiori. Class conditional nearest neighbor for large margin instance selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(2):364–370, 2010.

[76] Antonio A Márquez, Francisco A Márquez, Ana M Roldán, and Antonio Peregrín. An efficient adaptive fuzzy inference system for complex and high dimensional regression problems in linguistic fuzzy modelling. *Knowledge-Based Systems*, 54:42–52, 2013.

[77] Ricardo Martínez-Soto, Oscar Castillo, and Juan R. Castro. Genetic algorithm optimization for type-2 non-singleton fuzzy logic controllers. *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*, pages 3–18, 2014.

[78] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. MLlib: Machine learning in apache spark. *arXiv preprint arXiv:1505.06807*, 2015.

[79] Manuel Mucientes, Rafael Alcalá, Jesús Alcalá-Fdez, and Jorge Casillas. Learning weighted linguistic rules to control an autonomous robot. *International Journal of Intelligent Systems*, 24(3):226–251, 2009.

[80] Manuel Mucientes, Jesú. Alcalá-Fdez, Rafael Alcalá, and Jorge Casillas. A case study for learning behaviors in mobile robotics by evolutionary fuzzy systems. *Expert Systems With Applications*, 37:1471–1493, 2010.

[81] Manuel Mucientes and Alberto Bugarín. People detection through quantified fuzzy temporal rules. *Pattern Recognition*, 43:1441–1453, 2010.

[82] Manuel Mucientes and Jorge Casillas. Quick Design of Fuzzy Controllers with Good Interpretability in Mobile Robotics. *IEEE Transactions on Fuzzy Systems*, 15(4):636–651, 2007.

[83] Manuel Mucientes, David L. Moreno, Alberto Bugarín, and S. Barro. Design of a fuzzy controller in mobile robotics using genetic algorithms. *Applied Soft Computing*, 7(2):540–546, 2007.

[84] Manuel Mucientes, David L. Moreno, Alberto Bugarín, and Senén Barro. Evolutionary learning of a fuzzy controller for wall-following behavior in mobile robotics. *Soft Computing*, 10(10):881–889, 2006.

[85] Manuel Mucientes, Ismael Rodríguez-Fdez, and Alberto Bugarín. Evolutionary learning of Quantified Fuzzy Rules for hierarchical grouping of laser sensor data in intelligent control. In *Proceedings of the IFSA-EUSFLAT 2009 conference*, pages 1559–1564, Lisbon (Portugal), 2009.

[86] Manuel Mucientes, Juan C. Vidal, Alberto Bugarín, and Manuel Lama. Processing time estimations by variable structure TSK rules learned through genetic programming. *Soft Computing*, 13(5):497–509, 2009.

[87] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[88] Jorge L. Reyes-Ortiz, Luca Oneto, and Davide Anguita. Big data analytics in the cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf. *Procedia Computer Science*, 53:121–130, 2015.

[89] Ismael Rodríguez-Fdez, Adrián Canosa, Manuel Mucientes, and Alberto Bugarín. STAC: a web platform for the comparison of algorithms using statistical tests. In *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Accepted*, 2015.

[90] Ismael Rodríguez-Fdez, Manuel Mucientes, and Alberto Bugarín. Iterative Rule Learning of Quantified Fuzzy Rules for Control in Mobile Robotics. In *Proceedings of IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 111–118, Paris (France), 2011.

[91] Ismael Rodríguez-Fdez, Manuel Mucientes, and Alberto Bugarín. Photons detection in Positron Emission Tomography through Iterative Rule Learning of TSK rules. In *Actas del VIII Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, pages 251–258, Albacete (Spain), 2012.

[92] Ismael Rodríguez-Fdez, Manuel Mucientes, and Alberto Bugarín. An Instance Selection Algorithm for Regression and its Application in Variance Reduction. In *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2013.

[93] Ismael Rodríguez-Fdez, Manuel Mucientes, and Alberto Bugarín. Learning fuzzy controllers in mobile robotics with embedded preprocessing. *Applied Soft Computing*, 26:123–142, 2015.

[94] Ismael Rodríguez-Fdez, Manuel Mucientes, and Alberto Bugarín. Reducing the Complexity in Genetic Learning of Accurate Regression TSK Rule-Based Systems. In *Proceedings of the 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Istanbul (Turkey), 2015.

[95] Ismael Rodríguez-Fdez, Manuel Mucientes, and Alberto Bugarín. FRULER: Fuzzy Rule Learning through Evolution for Regression. *Information Sciences*, 354:1–18, 2016.

[96] Enrique H. Ruspini. A new approach to clustering. *Information and control*, 15(1):22–32, 1969.

[97] Khairulmizam Samsudin, Faisul Arif Ahmad, and Syamsiah Mashohor. A highly interpretable fuzzy rule base using ordinal structure for obstacle avoidance of mobile robot. *Applied Soft Computing*, 11(2):1631–1637, 2011.

[98] Luciano Sánchez, José Otero, and Inés Couso. Obtaining linguistic fuzzy rule-based regression models from imprecise data with multiobjective genetic algorithms. *Soft Computing*, 13(5):467–479, 2009.

[99] Bernhard Schölkopf, Alex J. Smola, Robert C. Williamson, and Peter L. Bartlett. New support vector algorithms. *Neural Computation*, 12(5):1207–1245, 2000.

[100] David W. Scott. *Multivariate density estimation: theory, practice, and visualization*, volume 383. John Wiley & Sons, 2009.

[101] Romano Scozzafava and Barbara Vantaggi. Fuzzy inclusion and similarity through coherent conditional probability. *Fuzzy Sets and Systems*, 160(3):292–305, 2009.

[102] K.S. Senthilkumar and K.K. Bharadwaj. Hybrid Genetic-Fuzzy Approach to Autonomous Mobile Robot. In *Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 29–34, 2009.

[103] Rudy Setiono and Lucas Chi Kwong Hui. Use of a quasi-Newton method in a feedforward neural network construction algorithm. *IEEE Transactions on Neural Networks*, 6(1):273–277, 1995.

[104] Ming-Yuan Shieh, J.C. Hsieh, and C.P. Cheng. Design of an intelligent hospital service robot and its applications. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 5, pages 4377–4382. IEEE, 2004.

[105] Michael Stout, Jaume Bacardit, Jonathan D. Hirst, and Natalio Krasnogor. Prediction of recursive convex hull class assignments for protein residues. *Bioinformatics*, 24(7):916–923, April 2008.

[106] Michio Sugeno and G.T. Kang. Structure identification of fuzzy model. *Fuzzy sets and systems*, 28(1):15–33, 1988.

[107] Muhammad Umar Suleman and Mian M. Awais. Learning from demonstration in robots: Experimental comparison of neural architectures. *Robotics and Computer-Integrated Manufacturing*, 27(4):794–801, 2011.

[108] Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man and Cybernetics*, (1):116–132, 1985.

[109] Philip R. Thrift. Fuzzy logic synthesis with genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 509–513, 1991.

[110] Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. In *Proceedings of the Joint conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pages 477–485. Association for Computational Linguistics, 2009.

[111] William N. Venables and Brian D. Ripley. *Modern applied statistics with S*. Springer-Verlag, 2002.

[112] Juan C. Vidal, Manuel Mucientes, Alberto Bugarín, and Manuel Lama. Machine scheduling in custom furniture industry through neuro-evolutionary hybridization. *Applied Soft Computing*, 11(2):1600–1613, 2011.

[113] L-X Wang and Jerry M. Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man and Cybernetics*, 22(6):1414–1427, 1992.

[114] Li-Xin Wang. Fuzzy systems are universal approximators. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, pages 1163–1170. IEEE, 1992.

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

[116] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[117] Lotfi A. Zadeh. A computational approach to fuzzy quantifiers in natural languages. *Computers & Mathematics with Applications*, 9(1):149–184, 1983.

[118] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.

[119] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society*, 67(2):301–320, 2005.

List of Figures

List of Tables