Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciência

Faculdade de Engenharia

Anderson Rodrigues dos Santos

Síntese de Árvores de Padrões Fuzzy através de Programação Genética Cartesiana

Rio de Janeiro

2014

Anderson Rodrigues dos Santos

Síntese de Árvores de Padrões Fuzzy através de Programação Genética Cartesiana

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao programa de Pós-Graduação em Engenharia Eletrônica da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Orientador: Prof. Dr. Jorge Luís Machado do Amaral

Rio de Janeiro

2014

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC / B

B272 Santos, Anderson Rodrigues dos.
Síntese de árvores de padrões fuzzy através de programação genética cartesiana / Anderson Rodrigues dos santos. - 2014.
97f.

Orientador: Jorge Luís Machado do Amaral.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Aprendizado de máquinas. 2. Árvores Fuzzy de Padrões – Dissertação. 3. Programação Genética Cartesiana – Dissertação. I. Amaral, Jorge Luís Machado do. II. Universidade do Estado do Rio de Janeiro. III. Título.

CDU 621.38

_____ _____

Assinatura Data

Anderson Rodrigues dos Santos

Síntese de Árvores de Padrões Fuzzy através de Programação Genética Cartesiana

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao programa de Pós-Graduação em Engenharia Eletrônica da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Aprovado em: 30 de Julho de 2014.

Banca Examinadora:

_____

Prof. Dr. Jorge Luís Machado do Amaral. (Orientador)
Faculdade de Engenharia – UERJ

_____

Profa. Dra. Marley Maria Bernardes Rebuzzi Vellasco
Pontifícia Universidade Católica do Rio de Janeiro - PUC – Rio

_____

Prof. Dr. André Vargas Abs da Cruz
Universidade Estadual da Zona Oeste - UEZO

Rio de Janeiro

2014

DEDICATÓRIA

Para minha família, Claudia, Ana e Ademir com todo meu amor e dedicação.

AGRADECIMENTOS

Agradeço:

Ao meu orientador Professor Jorge Amaral pelo estímulo, parceria e paciência para a realização deste trabalho.

À UERJ e ao Programa de Pós-Graduação em Engenharia Eletrônica, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

À minha família, pela paciência e carinho.

Aos meus pais, pela educação, atenção e carinho de todas as horas.

Aos meus amigos por todo apoio e compreensão.

Aos professores e funcionários do Programa de Pós-Graduação em Engenharia Eletrônica e do Departamento de Engenharia Eletrônica e de Telecomunicações.

A todos os amigos que de uma forma ou de outra me estimularam ou me ajudaram.

RESUMO

SANTOS, Anderson R. *Síntese de Árvores de Padrões Fuzzy através de Programação Genética.* 2014. 97f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2014.

Esta dissertação apresenta um sistema de indução de classificadores fuzzy. Ao invés de utilizar a abordagem tradicional de sistemas fuzzy baseados em regras, foi utilizado o modelo de Árvore de Padrões Fuzzy*(APF),* que é um modelo hierárquico, com uma estrutura baseada em árvores que possuem como nós internos operadores lógicos *fuzzy* e as folhas são compostas pela associação de termos *fuzzy* com os atributos de entrada. O classificador foi obtido sintetizando uma árvore para cada classe, esta árvore será uma descrição "lógica" da classe o que permite analisar e interpretar como é feita a classificação. O método de aprendizado originalmente concebido para a APF foi substituído pela Programação Genética Cartesiana com o intuito de explorar melhor o espaço de busca. O classificador APF foi comparado com as Máquinas de Vetores de Suporte, K Vizinhos mais próximos e florestas aleatórias em diversas bases de dados do UCI *Machine Learning Repository* e observou-se que o classificador APF apresenta resultados competitivos. Ele também foi comparado com o método de aprendizado original e obteve resultados comparáveis com árvores mais compactas e com um menor número de avaliações.

Palavras-Chave: Aprendizado de máquinas, Árvores Fuzzy de Padrões, Programação Genética Cartesiana, Classificação, Interpretabilidade.

ABSTRACT

This work presents a system for induction of fuzzy classifiers. Instead of the traditional fuzzy based rules, it was used a model called Fuzzy Pattern Trees (FPT), which is a hierarchical tree-based model, having as internal nodes, fuzzy logical operators and the leaves are composed of a combination of fuzzy terms with the input attributes. The classifier was obtained by creating a tree for each class, this tree will be a "logic class" description which allows the interpretation of the results. The learning method originally designed for FPT was replaced by Cartesian Genetic Programming in order to provide a better exploration of the search space. The FPT classifier was compared against Support Vector Machines, K Nearest Neighbour and Random Forests on several datasets from the UCI Machine Learning Repository and it presented competitive results. It was also compared with Fuzzy Pattern trees generated by the former learning method and presented comparable results with smaller trees and a lower number of functions evaluations.

Keywords: Machine Learning, Fuzzy Pattern Trees, Cartesian Genetic Programming, Classification, interpretability.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

AG	*Algoritmos Genéticos*
APF	*Árvore de Padrões Fuzzy*
BC	*Base de Conhecimento*
BD	*Base de Dados*
BR	*Base de Regras*
FP	*Função de Pertinência*
FPGA	*Field Programmable Gate Array*
KNN	*K-Nearest Neighbors*
PADO	*Parallel Algorithm Discovery and Orchestration*
PDGP	*Parallel Distributed Genetic Programming*
PG	*Programação Genética*
PGC	*Programação Genética Cartesiana*
RAF	*Replication Accuracy Force*
RF	*Random Forests*
SF	*Sistema Fuzzy*
SFBR	*Sistema Fuzzy Baseado em Regras*
SIF	*Sistema de Inferência Fuzzy*
SVM	*Support Vector Machine*

SUMÁRIO

INTRODUÇÃO

O grande progresso na aquisição de dados digitais e a evolução das tecnologias de armazenamento vêm criando um volume imenso de dados cuja análise está acima dos limites da capacidade humana. Isto justifica um grande interesse nas áreas de descoberta de conhecimento, mineração de dados e aprendizado de máquinas. A área de descoberta de conhecimento trata do processo de identificar estruturas nos dados que sejam válidas, compreensíveis e potencialmente úteis (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996). A mineração de dados diz respeito à aplicação de métodos computacionais e algoritmos para explorar grandes quantidades de dados a procura de padrões e relacionamentos ocultos para auxiliar a tomada de decisões (CLIFTON, 2010). O aprendizado de máquinas é o ramo da inteligência artificial que trata do estudo de modelos que podem aprender a partir de um conjunto de dados(WITTEN; FRANK, 2005). A indução destes modelos pode ser feita de forma automática por meio de diversas abordagens tais como: redes neurais artificiais, métodos bayesianos, modelos gráficos, árvores de decisão, entre outros. Estas três áreas estão intimamente relacionadas e tratam da aquisição e representação do conhecimento. Quando se deseja compreender "como" o modelo induzido consegue fazer a distinção entre diferentes classes ou como representar relações existentes nos dados de forma compreensível, abordagens mais simbólicas, por exemplo, sistemas baseados em regras tornam-se mais atrativos, pois além da capacidade expressar o conhecimento da uma forma compreensível, eles também possibilitam a introdução do conhecimento do especialista.

A Teoria dos Conjuntos Fuzzy é um dos paradigmas mais importantes da Inteligência Computacional, onde há muito são explorados aspectos de inferência e de representação do conhecimento. A maior motivação para a utilização da Teoria dos Conjuntos Fuzzy foi criar uma interface entre padrões quantitativos e estruturas de conhecimento qualitativas expressas em termos de linguagem natural. Esta característica faz com que ela seja atraente do ponto de vista da representação do conhecimento, permitindo que o conhecimento adquirido em uma base de dados possa ser representado de uma forma linguística compreensível, gerando uma maior interpretabilidade do modelo(HÜLLERMEIER, 2005).

A aplicação mais frequente da Teoria dos Conjuntos Fuzzy é a indução ou adaptação de Sistemas Fuzzy Baseados em Regras (SFBR), representando um tópico de pesquisa muito importante. Os SFBR podem representar tanto funções de classificação

quanto de regressão. Existe um grande número de estratégias que foram desenvolvidas para induzir modelos fuzzy baseados em regras (CORDÓN, 2011). Tem especial importância na área de aprendizado de regras fuzzy, os métodos híbridos que combinam a teoria de sistemas fuzzy com técnicas de inteligência computacional como as redes neurais e os algoritmos evolucionários. Nos chamados métodos neuro-fuzzy (NAUCK; KLAWONN; KRUSE, 1997), (SUN; JANG, 1993), (ABRAHAM, 2005), uma estratégia possível é codificar o sistema fuzzy como uma rede neural e aplicar métodos consagrados de treinamento, como o *backpropagation* (HAYKIN, 1998). Os algoritmos evolucionários, que tem como representante mais utilizado os Algoritmos Genéticos, proporcionam um modo de codificar e evoluir funções de pertinência, operadores de agregação dos consequentes das regras, diferentes formatos de regras, diferentes operadores de agregação de regras e métodos de defuzzificação. Atualmente, eles permitem uma grande flexibilidade para projetar e otimizar SFBRs em relação as decisões de projeto, permitindo aos projetistas decidir quais os componentes devem permanecer fixos e quais devem evoluir de acordo com as medidas de desempenho (HERRERA, 2008).

A obtenção de um sistema fuzzy baseado em regras pode não ser uma tarefa simples. Se o número de variáveis for grande, o número possível de regras aumenta exponencialmente, tornando o processo de busca pelo conjunto adequado de regras mais difícil. Este efeito é conhecido como a "maldição da dimensionalidade". Além disso, dependendo da aplicação, uma grande quantidade de regras pode ser necessária para que o sistema atinja o desempenho desejado, por exemplo, em termos de acurácia[1]. Um sistema com grande número de regras apresenta um aumento do esforço computacional, uma diminuição de desempenho em tempo real e uma diminuição da interpretabilidade[2] (TORRA, 2002). Acurácia e interpretabilidade representam objetivos contraditórios. O ideal seria que os dois critérios pudessem ser satisfeitos, mas geralmente isto não é

[1] Acurácia: Capacidade de representar de modo fidedigno o sistema real. Ela deve ser tão maior quanto maior for a similaridade entre as respostas do Sistema real e do modelo fuzzy. Existem medidas bem definidas e que são amplamente aceitas para avaliar a acurácia. Por exemplo, para classificação , pode-se utilizar a porcentagem dos elementos corretamente classificados em um conjunto de dados. No caso da regressão, o erro médio quadrático pode ser usado(GACTO; ALCALÁ; HERRERA, 2011).

.

;

[2] Interpretabilidade: Capacidade de expressar o comportamento do Sistema real de forma compreensível. É uma propriedade subjetiva e normalmente está relacionada a vários fatores que estão relacionados a estrutura do modelo, tais como: número de variáveis de entrada, o número de regras, o número de termos linguístico, etc. Não existe uma medida padronizada para avaliar a interpretabilidade(GACTO; ALCALÁ; HERRERA, 2011).

possível. Portanto, os pesquisadores geralmente se concentram em obter o melhor compromisso entre interpretabilidade e acurácia(CASILLAS, 2003), dependendo dos requisitos do usuário.

Foram propostas diversas alternativas para tratar a "maldição da dimensionalidade", dentre as quais pode-se citar: identificação de relações entre as variáveis para reduzir o número de variáveis usadas, combinação de duas ou mais variáveis para obter uma nova variável para substituir as variáveis originais, interpolação das regras e sistemas fuzzy hierárquicos (TORRA, 2002). Sistemas hierárquicos caracterizam-se por possuir diversos módulos que contribuem para a solução final. Os módulos de mais baixa ordem recebem como entradas algumas das variáveis e suas saídas são usadas pelos módulos de mais alta ordem para calcular a solução final. Esta hierarquia pode facilitar a interpretabilidade, pois os módulos de mais baixa ordem encapsulam apenas uma parte do conhecimento do sistema, sendo mais fáceis de compreender do que um único sistema monolítico de regras. Além disso, um sistema hierárquico pode auxiliar na escolha do compromisso entre interpretabilidade e acurácia, pois permitem a definição de módulos de mais baixa ordem de acordo com os critérios dos projetistas. A literatura apresenta diversos sistemas hierárquicos (WANG; ZENG; KEANE, 2006), (SOUZA; VELLASCO; PACHECO, 2002), (GONCALVES et al., 2006), sendo que em 2008 Huang, Gedeon and Nikravesh propuseram um novo algoritmo para indução de classificadores fuzzy, chamado de Árvores de Padrões Fuzzy (APF) (HUANG; GEDEON; NIKRAVESH, 2008). Estes classificadores são modelos hierárquicos com uma estrutura de árvore, no qual os nós internos são operadores generalizados utilizados nos sistemas fuzzy e as folhas dessas árvores são compostas por termos fuzzy associados a um atributo de entrada. Eles implementam uma função que mapeia uma combinação dos atributos de entrada em um número no intervalo [0,1] apresentado na raiz da árvore. O classificador APF é interessante por diversas razões. Além do fato dele possuir propriedades interessantes do ponto de vista do aprendizado, como o mecanismo de seleção de atributos embutidos, APFs são atrativas do ponto de vista de interpretabilidade (SENGE; HÜLLERMEIER, 2011). De um modo geral, elas podem ser vistas como uma descrição lógica generalizada de uma classe. As APFs podem ser consideradas como uma alternativa viável ao modelo clássico de regras fuzzy. Quando comparado a estes modelos, a estrutura hierárquica da árvore pode permitir uma representação mais

compacta além de permitir a determinação do compromisso de acurácia com interpretabilidade de forma mais natural.

O algoritmo original para o aprendizado destas árvores (HUANG; GEDEON; NIKRAVESH, 2008) apresenta problemas de convergência prematura, podendo ficar preso em ótimos locais. Um algoritmo alternativo mais eficiente foi desenvolvido por Senge e Hüllermeier (2011). Este algoritmo segue uma estratégia de busca chamada de *beam-search* (ZHANG, 1998), que explora um grafo de soluções expandindo o nó mais promissor em um conjunto limitado de opções. Este algoritmo possui uma característica gulosa, a busca no espaço de soluções é feita de forma restrita limitada pelo número de opções (largura do feixe de busca). Isto sugere a possibilidade de melhoria das soluções obtidas se for utilizado um mecanismo de busca global.

A Programação genética cartesiana (PGC)(MILLER, 2011) é uma forma de programação genética no qual os programas são representados por grafos. Uma das motivações da utilização de grafos ao invés da árvore utilizada na programação genética convencional é o fato de que grafos são mais gerais, flexíveis e compactos e podem ser aplicados em diversos domínios(DHARWADKER; PIRZADA, 2011). Dentre as vantagens da PGC estão as características de neutralidade, a redundância e a ausência de um problema chamado *Bloat* comum em outros métodos de programação genética(BANZHAF, 1994),(MILLER; SMITH, 2006),(MILLER, 2001).

Este trabalho apresenta um método para indução de modelos de Árvores de Padrões Fuzzy (APF) de forma automática, utilizado na tarefa de classificação. O método de aprendizado do APF foi substituído e em seu lugar foi utilizada a Programação Genética Cartesiana (PGC). A PGC é um método de busca global capaz de explorar espaços de busca bastante grandes de forma eficiente e a representação dos programas na forma de grafos pode ser facilmente utilizada para representar APFs.

Foram realizados diversos estudos de casos para obter uma melhor compreensão do funcionamento deste método, desenvolver estratégias para obter uma melhor generalização e avaliar o desempenho dos classificadores fuzzy gerados.

O restante desta dissertação está divida da seguinte forma: O capítulo 1 apresenta conceitos básicos dos Sistemas Fuzzy baseados em Regras; o capítulo 2 introduz as Árvores de Padrões Fuzzy. O capítulo 3 discorre sobre a Programação Genética Cartesiana; o Capítulo 4 apresenta o modelo proposto, o Capítulo 5 discute sobre os resultados obtidos e o Capítulo 6 apresenta a conclusão e a sugestão de trabalhos futuros.

1 SÍNTESE DE SISTEMAS FUZZY

1.1 Conceitos Básicos sobre Teoria dos Conjuntos Fuzzy

A Teoria de Conjuntos Fuzzy foi introduzida por L.A. Zadeh com o objetivo de fornecer um ferramental matemático para o tratamento de informações de caráter impreciso ou vago(ZADEH, 1965).

Formalmente, um conjunto nebuloso A do universo de discurso U é definido por uma função de pertinência μ_A: $U \rightarrow [0,1]$. Esta função associa cada elemento x de U o grau $\mu_A(x)$, com o qual x pertence a A (ROSS, 2010). A função de pertinência $\mu_A(x)$ indica o grau de compatibilidade entre x e o conceito expresso por A.

Um conjunto A da teoria dos conjuntos clássica pode ser visto como um caso particular de um conjunto fuzzy, para o qual μ_A: $U \rightarrow \{0,1\}$, ou seja, a pertinência é do tipo "tudo ou nada", "V ou F", e não gradual como para os conjuntos nebulosos (ROSS, 2010).

É comum associar-se um rótulo linguístico a um conjunto fuzzy. Uma função de pertinência (FP) possível para o conjunto fuzzy *Alta* é mostrada na Figura 1. Observe que, neste caso, a função de pertinência para Altura maior ou igual a 1,80 m tem valor próximo de 1, indicando que este valor pertence, definitivamente, ao conjunto *Alta*, enquanto que, para o valor menor ou igual a 1,30 m esta indicação não se verifica. Para o caso de uma altura próxima de 1,60 m, a função de pertinência fornece um valor em torno de 0,5. Do ponto de vista da informação, este valor representa uma completa falta de conhecimento sobre o assunto em questão: "*Alta*".

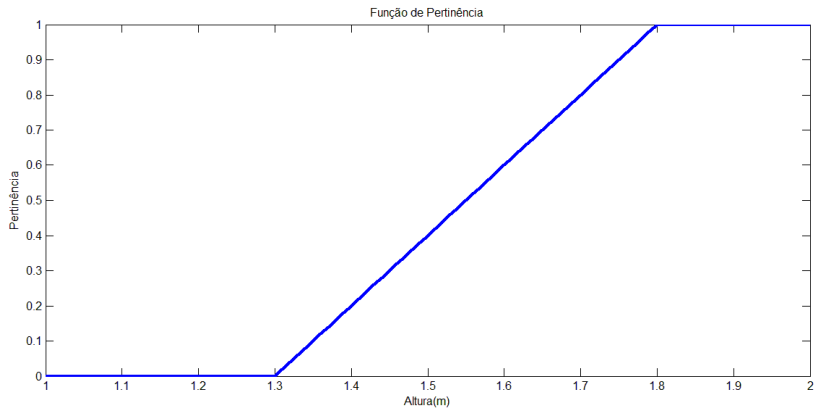

Figura 1: Função de Pertinência do conjunto fuzzy "Alta".

Na lógica fuzzy, uma variável linguística é uma variável cujos valores são nomes de conjuntos fuzzy. Ela pode ser definida por uma quádrupla (X, U, $T(X)$, M)(SANDRI; CORREA, 1999), onde X é o nome da variável, U é o universo de discurso de X, $T(X)$ é um conjunto de nomes para valores de X e M é uma função que associa uma função de pertinência a cada elemento de $T(X)$. Chamamos aqui de termos linguísticos, indistintamente, tanto os elementos de $T(X)$, que são conjuntos fuzzy, quanto suas funções.

A principal função das variáveis linguísticas é fornecer uma maneira sistemática para uma caracterização aproximada de fenômenos através da descrição linguística, que é normalmente empregada por seres humanos. Isto permite um melhor entendimento do problema que está sendo analisado. A utilização de variáveis linguísticas ao invés de variáveis quantificadas permite o tratamento de sistemas que são muito complexos para serem analisados através de mecanismos matemáticos convencionais. (GOMIDE; GUDWIN; TANSCHEIT, 1995).

Da mesma forma que na teoria dos conjuntos clássica, também é possível realizar operações de interseção, união e negação com conjuntos fuzzy. Os operadores de intersecção e união podem ser definidos, através das normas e conormas triangulares (t-normas e s-normas, respectivamente). Seguem-se as respectivas definições(PEDRYCZ, 1989). Uma t-norma é uma função binária t: [0,1] x [0,1] $\rightarrow$ [0,1], que satisfaz às seguintes condições, $\forall$ x, y, z, w $\in$ [0,1]:

Monotonia:	$t(x, w) \leq t(y, z)$, para $x \leq y$ e $w \leq z$	(1)
Comutatividade:	$t(x, y) = t(y, x)$	(2)
Associatividade:	$t(t(x, y), z) = t(x, t(y, x))$	(3)
Condições limites:	$t(x, 0) = 0$ e $t(x, 1) = x$	(4)

As t-normas mais empregadas para definir a interseção de conjuntos fuzzy são, o *min*, o *produto e a interseção bold* (ROSS, 2010).

Uma s-norma é uma função binária s: [0,1] x [0,1] $\rightarrow$ [0,1], que satisfaz as propriedades de monotonia, associatividade e comutatividade como a t-norma e apresenta, nas condições limites, o seguinte comportamento:

Condições limites: $s(x, 0) = x$ e $s(x, 1) = 1$ (5)

As s-normas mais empregadas para definir a união de conjuntos fuzzy são, além do *max*, a soma probabilística e a soma truncada.(ROSS, 2010).

A partir das informações apresentadas sobre conjuntos e lógica fuzzy pode-se conceituar um sistema de inferência fuzzy baseado em regras ou simplesmente, sistema fuzzy baseado em regras (SFBR).

1.2 Sistemas Fuzzy Baseados em Regras

Os Sistemas Fuzzy representam uma das áreas de aplicação mais importantes da Teoria dos Conjuntos Fuzzy, sendo que o modelo mais utilizado é aquele baseado em regras. Nestes sistemas, a lógica fuzzy é utilizada para representar diferentes formas de conhecimento sobre um problema e modelar as interações entre as suas variáveis. Eles são uma extensão dos sistemas clássicos baseados em regras, porque utilizam regras do tipo "SE-ENTÃO", onde os antecedentes e consequentes são proposições fuzzy ao invés das clássicas (HERRERA, 2008). SFBR demonstraram sua eficácia nas áreas de controle (BERNARD, 1988), modelagem(PONGRACZ; BOGARDI; DUCKSTEIN, 1999), aprendizado de máquinas (CORDÓN; DEL JESUS; HERRERA, 1999) e mineração de dados (FREITAS, 2003) em um grande número de aplicações.

A estrutura geral de um SFBR pode ser vista na figura 2. Ele é composto por uma interface de fuzzificação, uma unidade de inferência, uma interface de defuzzificação e por uma base de conhecimento (BC), que por sua vez divide-se em Base de dados (BD) e a Base de Regras (BR).

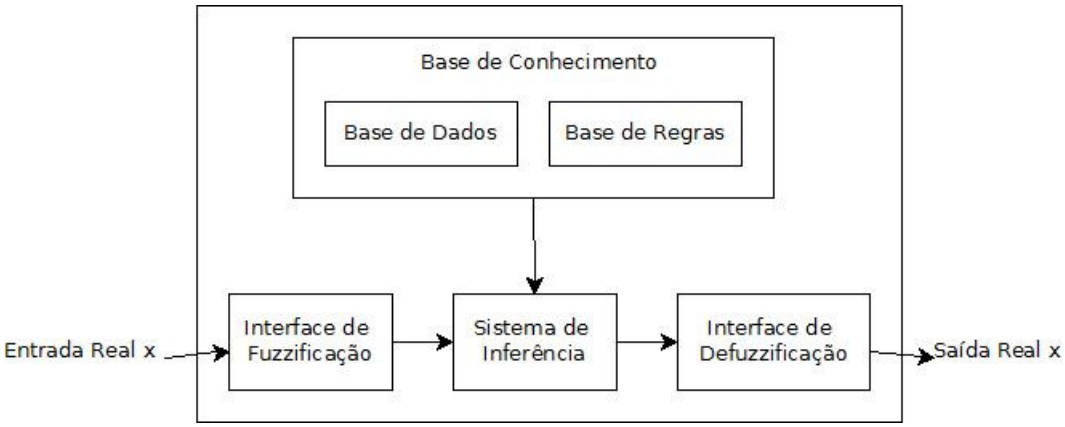

Figura 2: Modelo Geral de um Sistema de Inferência Fuzzy

A base de dados contem os conjuntos de termos linguísticos utilizados nas regras, as funções de pertinência que definem a semântica dos rótulos linguísticos e as funções de escala. Cada variável linguística é associada a uma partição fuzzy do seu domínio representando o conjunto fuzzy associado a cada termo linguístico. Uma partição fuzzy pode ser vista na figura 3. As funções de escala são utilizadas na transformação entre o universo de discurso onde os conjuntos fuzzy são definidos e o domínio das variáveis de entrada e saída. (HERRERA, 2008).

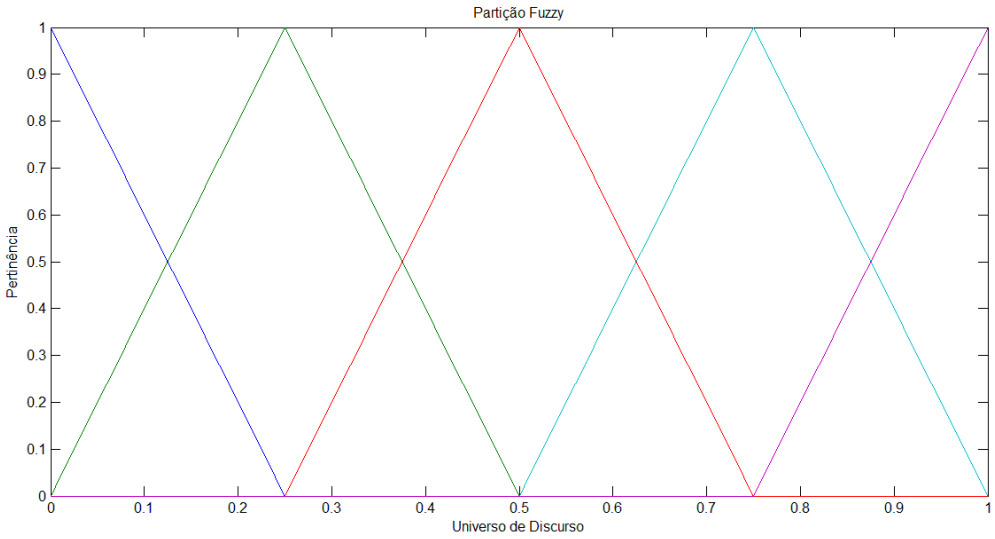

Figura 3: Exemplo de uma partição fuzzy

A base de regras é um conjunto de regras linguísticas do tipo SE-ENTÃO, que possuem a seguinte forma:

SE *um conjunto de condições é satisfeito*

ENTÃO *um conjunto de consequentes pode ser inferido*

Os SFBR podem ser categorizados em diferentes tipos: Mamdani e Takagi-Sugeno. O primeiro utiliza modelos linguísticos baseados na coleção de regras SE-ENTÃO da forma:

R_i: SE X_{i1} é A_{i1} E ... E X_{in} é A_{in} ENTÃO Y é B_i

Ou

R_i: SE X_{i1} é A_{i1} E ... E X_{in} é A_{in} ENTÃO C_k com w_{ik}

Com i = 1 até n (número de regras), e com X_{i1} , ..., X_{in} e Y sendo, respectivamente, as variáveis de entrada e saída, para o caso de um problema de regressão, e C_k a classe de saída para o caso de um problema de classificação. A_{i1}, ..., A_{in} e B_i são, respectivamente, os rótulos dos antecedentes e consequentes. O grau de confiança associado a classe é determinado por w_{ik} (HERRERA, 2008).

O SFBR Mamdani cria um aparato que permite a tradução de forma natural do conhecimento de um especialista em regras linguísticas. Podendo ser agregado a este conhecimento, regras geradas automaticamente a partir de um conjunto de dados que relaciona a entrada com a saída. O SFBR Mamdani é apropriado para aplicações que enfatizam a interpretabilidade (CORDÓN, 2011).

Entre os principais problemas do SFBR Mamdani, está na falta de acurácia em problemas complexos, ocorrendo devido à estrutura das regras linguísticas (CORDÓN, 2011).

O segundo tipo é baseado em uma estrutura de regras que tem o antecedente fuzzy e o consequente é uma função polinomial, que pode ser representado da seguinte forma:

R_i: SE X_{i1} é A_{i1} E ... E X_{in} é A_{in} ENTÃO $Y = p\ (X_{i1},...,X_{in})$

Onde $p(.)$ é normalmente uma função linear das entradas, isto é, $Y = p_0 + p_1\ X_{i1} + ... + p_n\ X_{in}$. Este sistema foi apresentado por Takagi e Sugeno, e sua principal característica é expressar a dinâmica local de cada regra com um modelo de sistema linear. Praticamente todos os sistemas dinâmicos não-lineares podem ser representados por modelos fuzzy Takagi-Sugeno com um alto grau de precisão(MEHRAN, 2008). Em (FANTUZZI; ROVATTI, 1996) e (BUCKLEY, 1992) é provado que modelos *fuzzy* Takagi-Sugeno são aproximadores universais de qualquer sistema não-linear(TAKAGI; SUGENO, 1985).

Em (CORDÓN, 2002) foi indicado que o SFBR Takagi-Sugeno divide o espaço de entrada do problema em diversos subespaços lineares e define uma relação entrada-saída linear em cada subespaço. No processo de inferência, estas relações parciais são agregadas com o intuito de criar uma relação entrada-saída global.

A maior vantagem deste SFBR é a presença de um sistema compacto de equações que permite que os coeficientes da função de saída sejam estimados por modelos clássicos, facilitando o procedimento de projeto. Em contrapartida, o SFBR Takagi-Sugeno é mais difícil de ser interpretado que o Mamdani. Isto ocorre pois a estrutura dos consequentes das regras é difícil de ser entendida por especialistas e porque sua saída depende da ativação da regra antecedente e da função no consequente das regras. Esta por sua vez, depende das entradas ao invés de ser constante.

A interface de fuzzificação tem a função de realizar o mapeamento de dados precisos para os conjuntos fuzzy de entrada. As funções de pertinência de cada conjunto fuzzy definido para cada variável de entrada recebem os valores reais das entradas para determinar o grau de pertinência de cada premissa da regra.

O módulo de inferência de um SFBR atua de modo diferente, dependendo do tipo de problema (classificação ou regressão) e do tipo de regras (linguísticas ou do tipo Takagi-Sugeno) Entretanto, pode-se descrever o comportamento do SFBR, de uma forma geral, do seguinte modo: Quando um SFBR é aplicado em uma situação particular (um determinado conjunto de entradas), todas as regras são disparadas em paralelo e para cada regra é computada sua conclusão. Este cálculo leva em conta o grau que cada antecedente é satisfeito (normalmente obtido por t-normas), de tal modo que se o antecedente não for satisfeito, a conclusão será um conjunto vazio. Finalmente a saída final será calculada através de uma combinação das conclusões todas as regras. Esta combinação normalmente consiste na união de todas as conclusões de todas as regras (realizada por uma t-conorma) e um passo final de defuzzificação. A defuzzificação transforma a união das conclusões em um valor preciso no caso de problemas de regressão ou fornecem a classe associada ao padrão de entrada de acordo como o modelo de inferência. Maiores detalhes do processo de inferência podem ser vistos em(ROSS, 2010).

A maior dificuldade na síntese de sistemas fuzzy encontra-se na definição dos termos linguísticos e das regras. Uma das maneiras de sanar este problema consiste em utilizar os chamados métodos neuro-fuzzy(NAUCK; KLAWONN; KRUSE, 1997),(SUN; JANG, 1993),(ABRAHAM, 2005). Nestes métodos, uma estratégia

possível é codificar o sistema fuzzy como uma rede neural e aplicar métodos consagrados de treinamento, como o *backpropagation* (HAYKIN, 1998).

Outra abordagem possível consiste na utilização de algoritmos Evolucionários que tem como representante mais utilizado os Algoritmos Genéticos (AG). Eles proporcionam um modo de codificar e evoluir, funções de pertinência, operadores de agregação dos consequentes das regras, diferentes formatos de regras, diferentes operadores de agregação de regras e métodos de defuzzificação (HERRERA, 2008).

A seguir, será apresentada uma descrição sucinta das diferentes abordagens para síntese de sistemas fuzzy utilizando Algoritmos Genéticos com base no que foi apresentado em (HERRERA, 2008) e em (CORDÓN, 2002). O que se deseja aqui é identificar aspectos positivos e negativos presentes em diversas abordagens de síntese de sistemas fuzzy com AGs, com o objetivo de delinear um conjunto de características que seriam desejáveis que o modelo proposto nesta dissertação possuísse.

1.3 Síntese de sistemas Fuzzy Baseados em Regras

Os Algoritmos Genéticos (AGs) são uma técnica de busca global bastante conhecida, com capacidade de explorar espaços de busca muito grandes para encontrar soluções apropriadas necessitando apenas de uma medida de desempenho (aptidão). Além de sua capacidade de encontrar soluções próximas do ótimo em espaços de busca complexos, a forma genérica de representação do problema os torna candidatos adequados a incorporação de conhecimento prévio (HERRERA, 2008). No caso dos sistemas fuzzy este conhecimento prévio pode estar relacionado aos formatos das funções de pertinência e aos seus parâmetros, ao número de regras fuzzy, que variáveis linguísticas utilizar em cada regra, etc. Isto permitiu que o uso do AG no desenvolvimento de diversas abordagens para o projeto de Sistemas Fuzzy, uma vez que eles fornecem um modo eficiente de evoluir de codificar e evoluir os operadores, formato e parâmetros de conjuntos fuzzy, regras com diferentes semânticas e operadores de defuzzificação. Portanto, AGs permitem que seja possível otimizar sistemas fuzzy em relação as decisões de projeto, possibilitando que os projetistas escolham quais os componentes do sistema devem permanecer fixos e quais devem evoluir de acordo com as medidas de desempenho. A Figura 4 mostra a relação entre o processo de aprendizado baseado em AG e o Sistema Fuzzy.

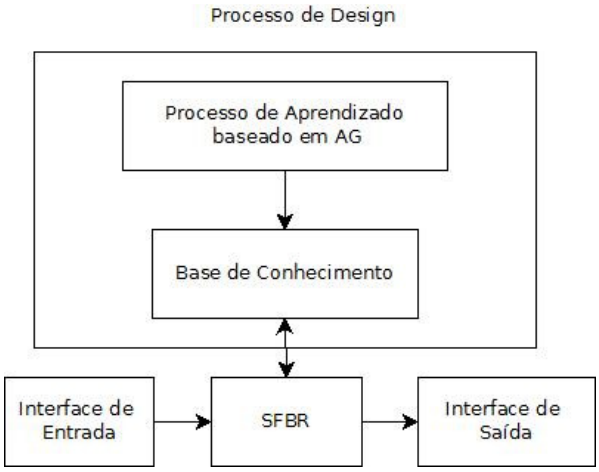

Figura 4: Relação entre o processo de aprendizado baseado em AG e o Sistema Fuzzy. Adaptado (Herrera, 2008)

Herrera (2008) propõe uma taxonomia que distingue duas abordagens: O ajuste genético e o aprendizado genético. No primeiro, supõe-se que a base de conhecimento (BC) já é conhecida e que o AG é aplicado apenas melhorar o desempenho do SFBR sem alterar a base de regras (BR). Isto significa que serão ajustados os parâmetros do SFBR, como por exemplo, os parâmetros das funções de pertinência. No segundo, se prevê a possibilidade de aprender os componentes da BC, além os outros componentes do SFBR.

1.3.1 Ajuste Genético

Para melhorar o desempenho dos SFBR, algumas abordagens procuram ajustar os parâmetros da Base de Dados (BD) ou os do módulo de inferência, uma vez que a base de regras foi obtida seja através de um especialista ou a partir de dados numéricos (GUILLAUME, 2001). Normalmente são considerados três tipos de ajuste. O primeiro se refere ao ajuste apenas dos parâmetros das funções de pertinência, isto é, se ajusta os formatos, mas não o número de termos linguísticos em cada partição fuzzy. O que se observa neste caso é que o ajuste de parâmetros das funções de pertinência pode alterar completamente o significado de cada conjunto fuzzy, o que pode dificultar a interpretação do sistema obtido. Um segundo ponto que deve ser observado diz respeito à aplicação do sistema fuzzy. Quando se trata de uma aplicação de classificação ou de regressão, o conjunto de parâmetros é ajustado levando em conta certo conjunto finito de dados. É possível que o ajuste dos parâmetros realizado para aumentar o desempenho

seja guiado por propriedades específicas deste conjunto de dados, o que pode levar ao *overfitting* do sistema obtido (WITTEN; FRANK, 2005).

Em (CASILLAS et al., 2005) e (KOSHIYAMA, 2014) propõe-se o uso de modificadores linguísticos como forma de ajustar as funções de pertinência. Esta forma tem a vantagem de permitir o ajuste, mas ainda mantendo a clareza do significado.

O segundo método de ajuste consiste em usar expressões parametrizadas para representar os operadores no módulo de inferência. Isto permite que os parâmetros dos operadores sejam ajustados através de AG. Em (ALCALÁ-FDEZ; HERRERA, 2007), este método é utilizado como forma de aumentar a acurácia sem comprometer a interpretabilidade do modelo. Esta é uma estratégia interessante, desde que o ajuste dos operadores não comprometa o seu significado.

Um terceiro método consiste usar métodos de defuzzificação adaptativos. A técnica mais usada na prática, devido ao seu bom desempenho e fácil implementação, é aplicar uma função de defuzzificação para cada conjunto de regras inferido e calcular um valor final com base na média dos resultados. Se operador de média for parametrizado para permitir o ajuste por AG. Da forma semelhante ao segundo método, o uso de operador parametrizado, neste caso, o operador de média pode ser usado para aumentar o desempenho em termos de acurácia, ainda permitindo uma interpretação do resultado obtido. Outra opção consiste na adaptação direta do operador de defuzzificação como visto em (KIM; CHOI; LEE, 2002).

1.3.2 Aprendizado Genético

A segunda grande área descrita por Herrera é o chamado aprendizado genético dos componentes da base de conhecimento (BC). As quatro abordagens mais comumente encontradas serão descritas e comentadas a seguir.

A primeira é o aprendizado genético das regras. A maior parte das abordagens propõe aprender a base de regras (BR) a partir de uma base de dados (BD) pré-definida. A forma tradicional é escolher um número ímpar de termos linguísticos, normalmente entre 3 e 9 , e distribui-los de forma uniforme no universo de discurso de cada variável. Um problema desta abordagem é a explosão do número de regras que aumenta de forma exponencialmente como o número de variáveis (“maldição da dimensionalidade”), sendo que cada uma das regras pode ter um grande número de proposições no

antecedente. Isto pode ter um efeito na interpretabilidade do sistema e mesmo na sua implementação, caso ele seja implementado em sistemas embutidos que possuem restrições de memória mais severas (TORRA, 2002).

A segunda abordagem, a seleção genética de regras consiste em utilizar o AG como método de busca de forma a determinar em um conjunto com um grande número de regras aquelas que são mais úteis para o bom desempenho do sistema. De modo semelhante a abordagem anterior, pode ocorrer a maldição de dimensionalidade, tornando o espaço de busca muito grande e regras com um grande número de antecedentes, o que sugere que se adote um método de seleção de variáveis embutido (GUYON; ELISSEEFF, 2003).

A terceira abordagem consiste em obter toda a base de conhecimento considerando dois métodos diferentes para a obtenção da Base de Dados (BD) e da Base de Regras (BR). O primeiro é chamado de "Aprendizado da Base de dados a priori", onde primeiro se obtém a BD (formato e parâmetros das funções de pertinência, granularidade das partições, funções de escala, etc) para depois evoluir a BR. O segundo método é chamado "Aprendizado embutido da Base de Dados". Neste caso toda fez que uma DB é obtida gera-se uma BR. Com base na avaliação do conjunto BD+BR, pode-se repetir o processo (CORDON; HERRERA; VILLAR, 2001).

A quarta abordagem consiste na geração simultânea da BD e da BR. Ela tem a possibilidade de gerar um sistema fuzzy melhor, mas tem que lidar com um espaço de busca extremamente grande (HOMAIFAR; MCCORMICK, 1995).

Além das descrições apresentadas, encontram-se também modelos híbridos que combinam o ajuste genético com o aprendizado genético. Por exemplo, pode-se combinar a utilização de operadores parametrizados com o aprendizado da BD e da BR. Neste caso, o espaço de busca torna-se ainda maior, o que sugere a utilização de algoritmos mais eficientes de busca ou a utilização de métodos paralelos de processamento(LUQUE; ALBA, 2011).

1.3.3 Aprendizado Genético das Regras

Muitos algoritmos de Aprendizado de Máquinas se baseiam na busca de um bom modelo em um espaço de modelos possíveis. Neste sentido, os AGs podem se apresentar como uma boa opção, pois são mecanismos de busca bastante eficientes e

permitem que a codificação do problema seja feita de forma bastante flexível, o que permite ao projetista escolher diferentes níveis de complexidade para os seu problema. No caso de SFBRs, pode-se considerar desde o nível mais simples, que seria a otimização de alguns parâmetros até a evolução completa da BC. A escolha destes diferentes níveis de complexidade é feita através da codificação utilizada e da cooperação/competição dos cromossomas.

Para a tarefa do aprendizado de regras para um SFBR, os diferentes métodos de aprendizado genético seguem duas abordagens principais para codificar regras em uma população de indivíduos. A primeira é a chamada abordagem de Pittsburgh no qual cada indivíduo representa um conjunto de regras(FREITAS, 2003). Estes indivíduos competem entre si e ao final da evolução, o indivíduo mais apto (segundo os critérios do usuário expressos na função de aptidão) representa a BR que será utilizada. Esta é a estratégia mais intuitiva e também uma das mais utilizadas. O maior problema desta abordagem está no custo computacional. As estratégias mais comuns consistem na simplificação da codificação(CORDÓN, 2002) ou a paralelização da computação (LUQUE; ALBA, 2011).

Na segunda abordagem, cada cromossoma representa apenas uma regra, o que a princípio reduz o custo computacional. Por outro lado perde-se a simplicidade na obtenção da BR final. Isto acontece porque precisam ser escolhidos os indivíduos (regras) que juntos formem uma BR compacta, eficiente e sem contradições ou superposições. Isto normalmente requer um processo de avaliação que pode ser complexo. Dentro desta abordagem, existem três propostas principais: Michigan, Aprendizado Iterativo de Regras e Aprendizado Genético Competitivo-Cooperativo.

Na proposta de Michigan (FREITAS, 2003), os indivíduos representam as regras e o conjunto de regras é representado pela população. As regras são modificadas ao longo do tempo através da interação com o ambiente. Embora existam muitas variações, um sistema Michigan é basicamente composto de: um sistema de inferência fuzzy (SIF) que interage com o ambiente, um mecanismo de geração de regras e um módulo de crédito que confere a credibilidade de cada regra (HERRERA; MAGDALENA, 1997). Como resultado das ações executadas no ambiente pelo SIF, o módulo de crédito acrescenta ou retira crédito das regras. Durante o processo evolutivo, somente as regras mais qualificadas permanecem.

Na proposta de Aprendizado Iterativo de Regras(VENTURINI, 1993), da mesma forma que na proposta Michigan, cada cromossoma representa um indivíduo, mas somente o

melhor indivíduo é considerado parte da solução final. Em cada rodada do AG, os cromossomas competem e apenas a melhor regra é escolhida e a solução global é formada pelas melhores regras obtidas nas diversas execuções do AG. Na proposta de Aprendizado Genético Competitivo-Cooperativo (ISHIBUCHI; NAKASHIMA; MURATA, 1999),(GIORDANA; NERI, 1995), (GREENE; SMITH, 1993) a população inteira ou um subconjunto codifica a BR. Neste caso, os cromossomas competem e cooperam simultaneamente.

Comparando estas propostas, observa-se que, sistemas Pittsburgh tem normalmente o maior custo computacional, Michigan é mais indicada para um aprendizado *on line*. Ela é mais flexível para tratar um modo de aprendizado do tipo incremental enquanto Pittsburgh e o Aprendizado Iterativo são mais indicadas para o aprendizado do tipo *batch*. Michigan deu início à estratégia de evoluir o sistema final de forma separada, como foco nas regras. O maior problema de Michigan é resolver os conflitos existentes entre os resultados obtidos pelas regras individuais e pelo sistema de regras completo. A proposta de Aprendizado Genético Competitivo-Cooperativo apresenta uma estratégia interessante de desenvolver partes conjunto de regras, o que mantém algumas das vantagens vistas em Michigan, mas com uma redução dos problemas gerados em relação ao conflito e sobreposição de regras.

1.4 Tendências na área de pesquisa de Sistemas Fuzzy Genéticos

Em (HERRERA, 2008) e (CORDÓN et al., 2004) é apresentada uma discussão sobre as tendência da pesquisa na área de Sistemas Fuzzy genéticos. O propósito desta seção é discutir alguns dos tópicos apresentados como forma de motivar a escolha de algumas características do modelo proposto e sugerir futuros desdobramentos deste trabalho.

1.4.1 Aprendizado genético multiobjetivo de SFBRs: compromisso entre acurácia e interpretabilidade.

Existe uma tendência na comunidade científica que trabalha na área de modelagem fuzzy de buscar um compromisso entre acurácia e interpretabilidade. A melhoria da interpretabilidade em SFBRs está sendo considerada uma questão muito importante na pesquisa atual, onde não apenas a melhoria da acurácia está recebendo atenção. Mas também a obtenção de regras compactas e interpretáveis. Neste sentido

algoritmos evolucionários multiobjectivo podem ser utilizados para encontrar soluções para os objetivos conflitantes acurácia e complexidade sem privilegiar um em detrimento do outro. Existem diversas publicações nesta área(ISHIBUCHI; MURATA; TÜRKŞEN, 1997), (ISHIBUCHI; YAMAMOTO, 2004), (COCOCCIONI et al., 2007), (CASILLAS; MARTINEZ, 2007), (BERLANGA et al., 2006), (ALCALÁ et al., 2007).

O problema é que enquanto a definição de acurácia é relativamente simples, a definição de interpretabilidade é problemática. Embora exista algum consenso acerca de alguns aspectos (número de regras reduzido, regras devem conter poucas variáveis, os termos linguísticos devem ser compreensíveis, etc...), existe a necessidade de propor novas métricas de interpretabilidade e formalizar noções como simplicidade e interpretabilidade (HERRERA, 2008).

Deve-se acrescentar a esta discussão, que estas definições de interpretabilidade devem abranger não somente sistemas fuzzy baseados em regras, mas também outros tipo de sistemas hierárquicos (TORRA, 2002) e sistemas fuzzy baseados em árvores binárias(ZHANG; ZHANG, 2006) e outros modelos como as Árvores de Padrões Fuzzy(SENGE; HÜLLERMEIER, 2011). Deve-se investigar para estes tipos de sistemas que definições podem ser usadas.

1.4.2 A aplicação de técnica baseadas em AG para minerar regras de associação fuzzy e para novas abordagens em mineração de dados.

As variáveis e termos linguísticos podem contribuir no projeto de regras de associação e para estabelecer relações e identificar padrões em conjunto de dados. Os sistemas Fuzzy Genéticos podem fornecer uma ferramenta para análise de padrões e extração de informações úteis com uma vantagem sobre outras técnicas: a interpretabilidade fornecida pela regras fuzzy SE-ENTÃO. Existem diversas publicações sobre este tema (HONG et al., 2006),(KAYA, 2006),(TSANG; KWONG; WANG, 2007).

É importante mencionar que a interpretabilidade também pode ser obtida com outros modelos. Por exemplo, modelos hierárquicos, além de fornecer interpretabilidade, possibilitam identificar relações hierárquicas entre variáveis e conjuntos de regras.

1.4.3 A aprendizado genético em dados com baixa qualidade (dados ruidosos e vagos)

Quando se projetam modelos baseados em regras, é esperado que alguma acurácia seja sacrificada para obter interpretabilidade. Não se espera que estes modelos superem aqueles do tipo "caixa preta" embora esta seja esta a forma tradicional de validação (SÁNCHEZ et al., 2002). Entretanto, se os desempenhos não forem semelhantes, não se pode afirmar que esta diferença é devido as limitação do algoritmo de aprendizado ou se é devido à própria definição do problema.

A utilização de sistemas fuzzy para classificação e modelagem está relacionada ao tratamento de informações de caráter vago e impreciso. Os algoritmos de agrupamento fuzzy, por exemplo, tem como sua principal motivação o fato de que a definição de fronteiras bem definidas entre agrupamentos não é natural. A transição entre grupos deve ser suave, indicando que um objeto pode pertencer a mais de um grupamento. Entretanto, em muitas situações retira-se esta capacidade de tratamento dos sistemas fuzzy. Por exemplo, em problemas de classificação, o consequente das regras é a atribuição a classe. Neste caso, a avaliação de regras se resume a buscar a regra com maior suporte para cada classe, perdendo-se as propriedades de interpolação e aproximação. Como característica fuzzy resta apenas a informação de que cada regra é ativada com certo grau. Existem poucos *benchmarks* disponíveis como dados vagos para os quais sistemas fuzzy possam ser treinados e testados.

1.4.4 Aprendizado da base de dados e adaptação de contexto

O aprendizado da base de dados abrange a especificação do universo de discurso, o número de rótulos para cada variável linguística, bem como a definição das funções de pertinência associada a cada rótulo. Existem muitas abordagens para o aprendizado do número de funções de pertinência, seus formatos e parâmetros (BOTTA; LAZZERINI; MARCELLONI, 2006; BOTTA et al., 2007; GUDWIN; GOMIDE; PEDRYCZ, 1998). Esta é uma área importante pois é essencial escolher funções de pertinência adequadas ao contexto. Neste aspecto, os sistemas fuzzy genéticos são interessantes pela flexibilidade na codificação dos elementos da base de dados que permitem. (HERRERA, 2008).

Deve-se tomar cuidado com a codificação escolhida para que a quantidade de parâmetros que precisem ser ajustados não fique muito grande, em relação ao conjunto

de dados disponíveis para treinamento, pois isso poderia causar *overfitting*. Neste caso, a base de dados aprendida representará características específicas do conjunto de treinamento e não as relações fundamentais existentes nos dados. Mecanismos para evitar o *overfitting* devem ser empregados (WITTEN; FRANK, 2005).

1.4.5 Utilização de operadores parametrizados tanto para a agregação quanto para a defuzzificação para aumentar o desempenho

A utilização destes operadores parametrizados acrescentam parâmetros aos modelos e também podem contribuir para o *overfitting*, o que sugere que mecanismos contra o *overfitting* devem ser utilizados.

1.4.6 Novas abordagens sistemas do estilo Michigan.

Para isto propõe-se que a aptidão de uma regra esteja relacionada com a recompensa que ela provoca quando é ativada. Esta abordagem evita a obtenção de regras muito gerais, auxilia a obtenção de um conjunto ótimo de regras e possibilita o aprendizado em todo a partição fuzzy(HERRERA, 2008).

Sistemas Michigan são bastante úteis, pois aprendem automaticamente regras fuzzy que podem atuar no ambiente ao mesmo tempo em que dados ou um estímulo são recebidos. Ao contrário dos sistemas Pittisburgh, onde a base de regras completa é evoluída, eles evoluem partes da base de regras ao longo do tempo. Isto os torna ideais para aprendizado por reforço, sistemas adaptativos, mineração de dados e descoberta de conhecimento(HERRERA, 2008). A evolução de partes do sistema em paralelo é uma característica que deve ser enfatizada na geração de novos sistemas fuzzy.

1.4.7 Aprendizado Genético de Diferentes Estruturas

Melhorias nos modelos linguísticos podem ser obtidas considerando estruturas mais flexíveis. Dentre as possibilidades de relaxar a estrutura do modelo usando sistemas fuzzy genéticos pode-se citar: uso de regras fuzzy com duplo consequente, utilização de regras fuzzy ponderadas e a utilização de sistemas *fuzzy* hierárquicos (CORDÓN et al., 2004).

No caso da utilização de sistemas hierárquicos podemos ter aqueles baseados em regras (TORRA, 2002) ou não(SENGE; HÜLLERMEIER, 2011). A proposta de desenvolvimento de sistemas fuzzy não baseados em regras por meio de aprendizado genético ainda não foi amplamente explorado e necessita de investigação adicional.

1.4.8 Síntese de Sistemas Fuzzy baseados em Regras através de Programação Genética

A programação genética (PG) trata da evolução automática de programas (KOZA, 1992). Existem diferentes abordagens onde a PG evolui conjuntos de regras que são internamente representadas como árvores sintáticas (GEYER-SCHULZ, 1997), (BERLANGA et al., 2006), um exemplo pode ser visto na figura 5 (CORDÓN et al., 2004). Dentre os principais desafios para a síntese de SFBR através de programação genética é garantir a estrutura sintática do conjunto de regras. Uma solução é a presentada em (KOSHIYAMA, 2014), que desenvolve um sistema para síntese de SFBR utilizando programação genética multigênica. O *bloat* é outro problema comum em outros métodos de programação genética (BANZHAF, 1994; MILLER; SMITH, 2006; MILLER, 2001).

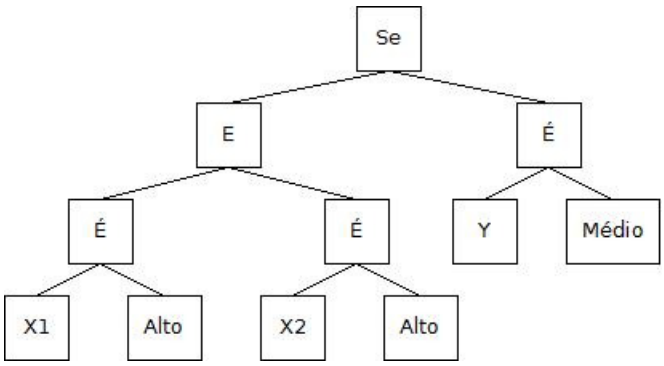

Figura 5: Uma Árvore Sintatica e a regra que ela representa: Se X1 é Alto e X2 é Alto então Y é Médio.

2 ÁRVORES DE PADRÕES FUZZY

2.1 Definições e conceitos básicos

O modelo de Árvores de Padrões Fuzzy (APF) foi criado com o intuito de representar o conhecimento na forma de um grafo ao invés de representá-lo na forma de regras. A utilização deste tipo de representação hierárquica procura minimizar os problemas existentes em sistemas baseados em regras tais como o aumento exponencial do número de regras, com o aumento do número de entradas e o comprometimento da interpretabilidade quando uma grande quantidade de regras é gerada para atingir os requisitos de acurácia. O primeiro método de indução de APF criado por Huang, Gedeon e Nikravesh(HUANG; GEDEON; NIKRAVESH, 2008) foi utilizado na área do aprendizado de máquinas. O algoritmo de geração da árvore foi posteriormente aperfeiçoado em(SENGE; HÜLLERMEIER, 2011).

As Árvores de Padrões Fuzzy são modelos hierárquicos com uma estrutura de árvore, no qual os nós internos são operadores utilizados nos sistemas fuzzy e as folhas dessas árvores são compostas por termos fuzzy associados a um atributo de entrada. A APF propaga a informação do fundo para o topo, os nós internos recebem os valores de seus antecessores e os combinam utilizando um operador, apresentando a saída para o nível superior. As APFs implementam um mapeamento recursivo, apresentando a saída em um intervalo unitário. Um classificador baseado em árvore de padrões é construído criando uma árvore para cada classe. São inseridos os valores dos atributos que se deseja classificar nas entradas das árvores de cada classe e a predição da classe que esse conjunto de dados pertence é feita escolhendo a árvore que tem o maior valor de saída. Geralmente cada árvore pode ser considerada como uma "descrição lógica[3]" da classe, permitindo uma interpretação mais concreta do problema de aprendizado(SENGE; HÜLLERMEIER, 2011).

Utilizando o arcabouço normalmente utilizado para o aprendizado supervisionado, o modelo de Árvore de Padrões Fuzzy necessita de um conjunto de exemplos para realizar o aprendizado, descrito pela equação 6:

$$T = \{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{n} \subset X \times Y \tag{6}$$

Onde cada instância $x^{(i)}$ é um vetor

[3] A descrição não é inteiramente lógica pois operadores aritméticos também são permitidos.

$$\boldsymbol{x} \in X = X_1 \times X_2 \times \cdots \times X_m$$

X_i é o domínio do atributo i-ésimo atributo A_i. Cada domínio é particionado em um conjunto de funções de pertinência $F_{i,j}$

$$F_{i,j}: X_i \to [0,1] \qquad (j = 1, \ldots, n_i)$$

De tal modo que $\sum_{j=1}^{n_j} F_{i,j}(x) > 0, \forall\, x \in X_i$. *Fi,j* normalmente recebe um rótulo linguístico com "ALTO" ou "BAIXO", e neste caso, é chamado de termo linguístico. Cada instância é associada a um rótulo da classe dado por:

$$y \in Y = \{y_1, y_2, \ldots, y_k\}$$

Ao contrário das árvores de decisão, a entrada da APF está em suas folhas. Cada folha da árvore é rotulada por uma atributo A_i e um conjunto fuzzy $F_{i,j}$ do domínio X_i correspondente. Dada uma instância $\boldsymbol{x}$ = (x_1, x_2, ...,x_m) como entrada o nó folha produz como saída $F_{i,j}(x_i)$, que é o grau de pertinência de x_i em $F_{i,j}(x_i)$. Este grau é propagado para outros nós em direção a raiz.

Além dos termos *fuzzy*, também são utilizados na criação das árvores, t-normas, t-conormas e dois operadores de média. O operador de média (OWA- *ordered weighted average*) é a combinação de *k* números, v_1, v_2,..., v_k, definido pela equação 7:

$$OWA_w(v_1, v_2, \ldots, v_k) \stackrel{\text{def}}{=} \sum_{i=1}^{k} w_i v_{\tau(i)} \tag{7}$$

Onde o τ é uma permutação entre o conjunto de números {1,2,...,*k*} de forma que $v_{\tau(1)} \leq v_{\tau(2)} \leq \ldots \leq v_{\tau(k)}$ e $w=(w_1,w_2,...,w_k)$ é um vetor de pesos que satisfaz $w_i \geq 0$ para i=1,2,...,*k* e o somatório dos pesos de 1 a *k* deverá ser igual a 1. O outro operador de média é o WA (*weighted average*), que é similar ao OWA, porém sem o ranqueamento dos valores de *v*.

O restante dos operadores utilizados na APF são apresentados nas tabelas 1 e 2.

Tabela 1: Operadores Fuzzy T-Norm

Mínimo	$\min(a, b)$
Algébrico	$a * b$
Lukasiewicz	$\max(a - 1 + b, 0)$
Einstein	$\dfrac{a * b}{2 - (a + b - a * b)}$

Tabela 2: Operadores Fuzzy T-Conorm

Máximo	$\max(a, b)$
Algébrico	$a + b - a * b$
Lukasiewicz	$\min(a + b, 1)$
Einstein	$\frac{a + b}{1 + a * b}$

Um exemplo de APF pode ser visto na figura 6, onde a árvore apresentada determina uma classe que representa a qualidade de um vinho. Os atributos de entrada são o teor alcoólico, a acidez, a concentração de sulfatos e de dióxido de enxofre. Estes atributos estão associados a um termo fuzzy que representa um intervalo do universo de discurso do atributo. Na figura 6, o termo fuzzy álcool_baixo, representa o conjunto fuzzy que representa teor alcoólico baixo. O valor de pertinência obtido nos conjuntos fuzzy vai sendo agrupado através de operadores que mantém os resultados parciais no intervalo [0,1]. O valor obtido na saída após todos os agrupamentos das características deve se aproximar de 1 se a árvore representa bem a classe. No caso da figura 6 se a árvore representar bem a classe de vinhos de boa qualidade. Um vinho que tem seus atributos inseridos nas entradas da árvore e tem um valor próximo a 1 na saída será considerado um vinho de boa qualidade.

Para o caso multiclasse, um classificador APF pode ser visto como uma coleção de APFs dada por:

$$\{APF_i | i = 1,2, \dots, k\}$$

Onde a APF_i é a árvore associada a classe com a classe y_i. Dada uma nova instância $\boldsymbol{x}$ para ser classificada, a decisão$\hat{y}$ é tomada em favor da classe cuja a árvore apresenta o maior valor na saída:

$$\hat{y} = argmax_{y_i \in Y}\, APF_i(\boldsymbol{x})$$

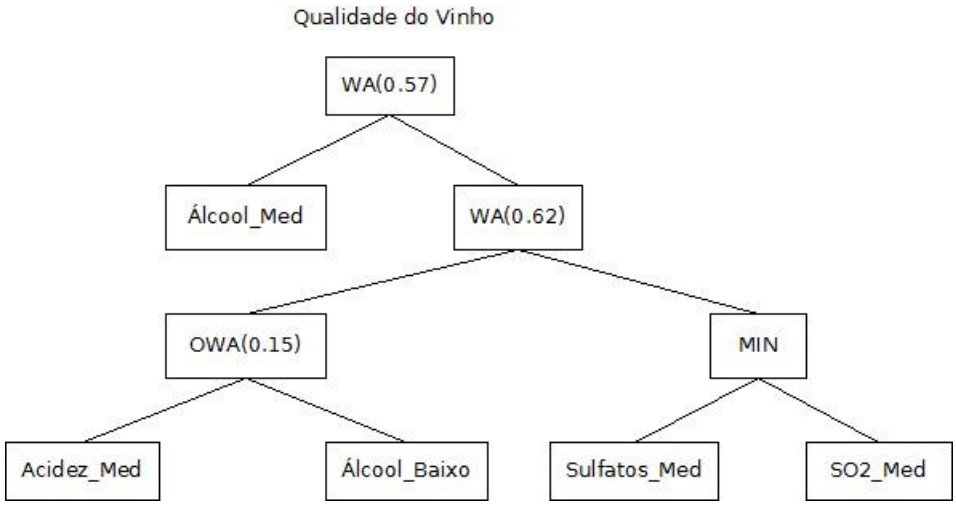

Figura 6: APF que representa a qualidade de um vinho.

O modelo baseado em árvore de padrões mapeia diversas entradas em apenas uma variável de saída. A interpretação da saída produzida pode ser vista como um modelo que simplifica a avaliação global de uma propriedade, avaliando diferentes subcritérios e agregando estas avaliações posteriormente(SENGE; HÜLLERMEIER, 2011). Retornando a figura 6, pode-se observar que a qualidade do vinho está relacionada com duas subárvores. A primeira relaciona o teor alcoólico e a acidez, enquanto a outra trata da concentração de sulfato e de dióxido de enxofre. A informação destas árvores é posteriormente combinada em um nível mais alto. Neste tipo de sistema é possível identificar que as subárvores representam diferentes conhecimentos que devem ser combinados. Também é possível observar as relações entre as variáveis. Ainda na figura 6, observa-se que o teor alcoólico médio (Alcool_med) tem uma grande influência no resultado pois está localizado próximo ao topo da árvore, onde é fornecido o resultado.

2.2 **Bottom-up induction e Top-down induction**

O primeiro método de aprendizado proposto por Huang, Gedeon e Nikravesh (HUANG; GEDEON; NIKRAVESH, 2008), é chamado de Bottom-up Induction. Nele, a indução das árvores procura criar uma representação “lógica” para cada classe de uma forma iterativa. O processo ocorre do fundo para o topo. Em cada iteração do processo, combinam-se as duas melhores árvores candidatas para criar um nova árvore. O uso

desta estratégia é motivado intuitivamente. Ela pode ser vista como uma combinação iterativa para a construção de características complexas, a partir de características mais básicas criadas pelos atributos originais (SENGE; HÜLLERMEIER, 2011).

A combinação de duas árvores candidatas para gerar uma terceira tende a realizar "grandes saltos" no espaço de busca e a gerar uma perda de diversidade, porque após algumas interações todas as árvores candidatas tornam-se semelhantes.

Outro método para o aprendizado de APF foi proposto por Robin Senge e Eyke Hüllermeier chamado de top-down induction(SENGE; HÜLLERMEIER, 2011) . Nesta estratégia, ao invés de juntar duas árvores em uma nova árvore, muito maior e com certa diferença na estrutura, a ideia é fazer pequenas modificações. Isto é feito expandindo um nó de folha. Por exemplo, na figura 7, a árvore formada apenas pela folha A , que representa a junção de um atributo e um termo fuzzy, representando uma característica básica, é substituída por uma característica composta pela mesma folha A e outra folha B, agregadas por um operador média. Os novos operadores são inseridos em níveis mais baixos da árvore, mais ao fundo, isto faz com que cada novo operador tenha uma influência menor no comportamento entrada-saída que os operadores inseridos anteriormente. Este procedimento de realizar pequenas modificações na árvore a cada iteração proporciona uma maior exploração do espaço de busca, o que aumenta a chance de encontrar uma árvore que atenda ao requisitos do usuário.

A procura pela solução na estratégia *top-down* é feita através de um algoritmo de busca heurístico chamado *beam search*, que explora a árvore expandindo o nó mais promissor a partir de um conjunto limitado de opções. A *beam search* é um procedimento de busca do tipo *best-first search*. Na *beam search* apenas um número predeterminado de melhores soluções parciais são mantidas como candidatos. A cada nível da árvore, ela gera todos possíveis sucessores dos nós do nível atual, organizando-os em uma ordem crescente de custo heurístico. Porém, para reduzir os requisitos de memória, este método só armazena um número pré-determinado, (largura do feixe de busca) de melhores sucessores de cada nível. Somente estes melhores sucessores serão expandidos futuramente.

A figura 7 exemplifica a sequência de criação de uma Árvore de Padrões Fuzzy *top-down*, iniciando com uma árvore de padrões primitiva (A), que é um subconjunto *fuzzy* no domínio de um atributo. Ocorre então a expansão das árvores candidatas de forma iterativa selecionando a melhor árvore baseando-se em um critério de medida, até que um critério de término seja atingido.

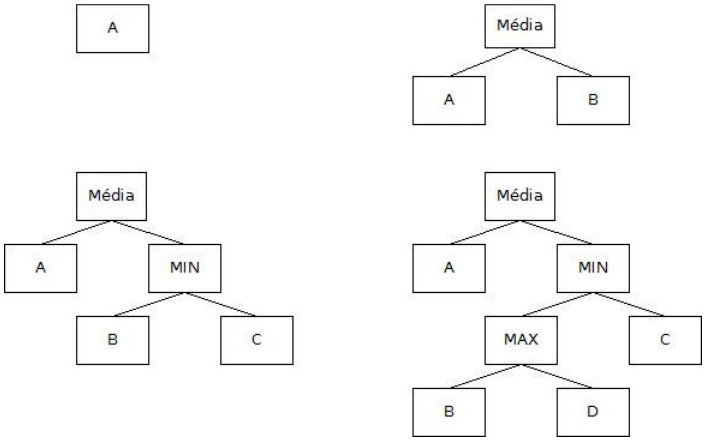

Figura 7: Sequência de criação de uma APF utilizando a estrategia TOP-DOWN.

A estratégia *top-down* possui algumas desvantagens:

- O critério de parada se baseia na melhoria relativa do modelo entre iterações. Este tipo de critério não põe limitações ao crescimento da árvore, podendo levar ao *overfitting*;
- O Algoritmo *Beam search* usa uma busca em largura para construir sua árvore de busca. Em cada nível da árvore ele gera todos os sucessores e os ordena Segundo uma medida de desempenho. Entretanto, ele armazena apenas um determinado número de candidates (dados pela largura do feixe) que serão explorados a seguir. Este procedimento confere ao algoritmo uma característica "gulosa", olhando sempre para o melhor candidato no estágio atual de construção, fazendo com que o algoritmo fique preso em subótimos globais.
- A maldição da dimensionalidade. Se a quantidade de atributos for grande, e a largura do feixe também for, o algoritmo terá problemas em testar todas as possibilidades, pois haverá uma explosão na quantidade de possibilidades. Por outro lado se a largura do feixe for pequena, então apenas uma pequena região do espaço de busca será explorada, o que pode fazer com que o algoritmo obtenha soluções ruins.

3 PROGRAMAÇÃO GENÉTICA

3.1 Introdução

Os Algoritmos Evolucionários são inspirados no princípio darwiniano da evolução das espécies e na genética. Do mesmo modo que a Evolução Natural produz indivíduos mais aptos a sobreviver em um meio-ambiente sujeito a constantes mudanças, os Algoritmos Evolucionários podem ser vistos como procedimentos de otimização que melhoram o desempenho de uma população de soluções em potencial em relação a um problema específico. Segundo este princípio, uma população de indivíduos evolui ao longo de gerações ou ciclos, pela sobrevivência dos mais aptos.

Os principais Algoritmos Evolucionários são: os Algoritmos Genéticos, a Programação Genética, as Estratégias Evolutivas e a Programação Evolutiva. Maiores Detalhes sobre cada um destes algoritmos podem ser vistos em (GOLDBERG, 1989), (MICHALEWICZ, 1996), (SCHWEFEL, 1995), (FOGEL, 1995), (BÄCK; SCHWEFEL, 1993) e (KOZA, 1992).

Uma subárea dos Algoritmos Evolucionários de particular interesse é a Programação Genética (PG). A PG é uma técnica que permite que computadores resolvam problemas sem que precisem ser explicitamente programados para tal (KOZA, 1992). A PG parte de uma declaração de alto nível sobre “o que se necessita ser feito” e cria automaticamente um programa de computador para resolver o problema. Este mecanismo também é conhecido como “programação automática”. (DIAS, 2010)

Nas últimas décadas, a programação genética tem sido utilizada por diversos pesquisadores. (FORREST et al., 2009)(AFZAL; TORKAR, 2011)(ESPEJO; VENTURA; HERRERA, 2010).

Ela é particularmente útil quando:

A relação entre as variáveis é desconhecida ou há pouco conhecimento sobre esta relação e não se dispõe de métodos analíticos capazes de estabelecer esta relação de forma satisfatória.

Situações em que os métodos matemáticos convencionais não podem criar soluções analíticas. Em domínios no qual as soluções analíticas não sejam possíveis, ou que estas soluções tenham um tempo de execução impraticável ou que precisem de uma característica indisponível nos dados, por exemplo, uma base de dados sem ruído.

Nestes casos o uso da programação genética é adequado, pois garante uma boa solução aproximada quando uma solução exata é impraticável.

Problemas em que pequenos aumentos no desempenho são facilmente medidos e altamente recompensados. Existem domínios possuem soluções muito avançadas, sendo difícil melhorar as soluções já existentes, porém nestes domínios um pequeno aumento no desempenho pode ser muito valioso, a programação genética pode ser útil no descobrimento de pequenas relações que podem criar estes pequenos e valiosos aumentos no desempenho.

3.2 Definição e conceitos básicos

Diferente da programação usual que utiliza de linhas de código, a programação genética utiliza uma representação baseada em árvores chamadas árvores sintáticas, que permite a solução do problema de aplicação do operador cruzamento em genótipos de tamanho variável, um exemplo de árvore sintática pode ser visto na figura 8. A figura mostra a representação em forma de árvore do programa, y=ln(x1)+5*x2. (KOZA, 1992)

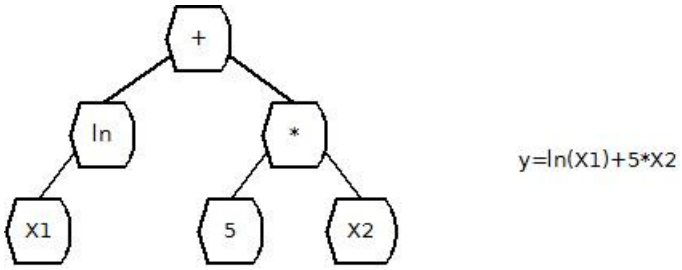

Figura 8: : Exemplo de uma Árvore Sintática

As “folhas” das árvores são compostas por variáveis e constantes, elas são chamadas de terminais, enquanto os nós internos que são compostos por operações são chamados de funções. Os programas podem ser compostos por múltiplos componentes que resolvem separadamente partes do problema global. Estes múltiplos componentes são chamados de *subtrees* ou ramificações. Um exemplo pode ser visto na figura 9. (POLI; LANGDON; MCPHEE, 2008)

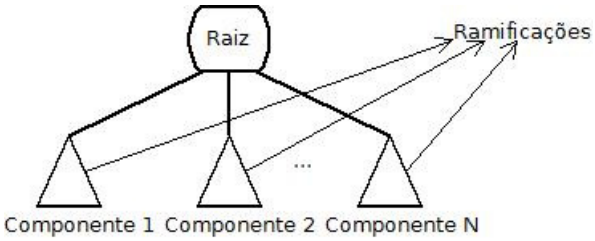

Figura 9: Estrutura de uma Árvore formada por Múltiplos componentes

A variabilidade dinâmica dos programas criados pela programação genética é uma característica importante, pois é difícil especificar o tamanho e a forma da eventual solução, a especificação ou restrição do tamanho e forma da solução pode diminuir o total de possíveis soluções que poderão ser encontradas. Gerando uma solução que não seja tão boa como poderia ser ou mesmo não encontrando uma solução (KOZA, 1992).

Outra característica importante da programação genética é ausência ou pequena necessidade de pré-processamento das entradas e pós-processamento das saídas. O processo evolutivo ocorre dentro do domínio do problema. As saídas já são expressas neste domínio do problema, não sendo necessários processos de tradução ou mapeamento(KOZA, 1992).

A inicialização das árvores é feita tipicamente de forma aleatória. Existem diversas formas de criar as árvores iniciais, os métodos mais simples são os *full* e *grow*. Nestes dois métodos as árvores iniciais são criadas de forma que não excedam um limite de profundidade definido pelo usuário. A profundidade de uma árvore é a quantidade de níveis que a árvore possui da raiz até a folha mais distante, sendo o nível da raiz o nível zero. No método *full* os nós são escolhidos de forma aleatória no conjunto de funções até que a árvore atinja o limite máximo de profundidade, após esta profundidade, somente terminais são escolhidos. A figura 10 mostra a sequência de criação de uma árvore inicial utilizando o método *full*. Neste método todas as folhas estão na mesma profundidade e que não implica em que todas as árvores tenham os mesmos números de nós ou o mesmo formato (POLI; LANGDON; MCPHEE, 2008).

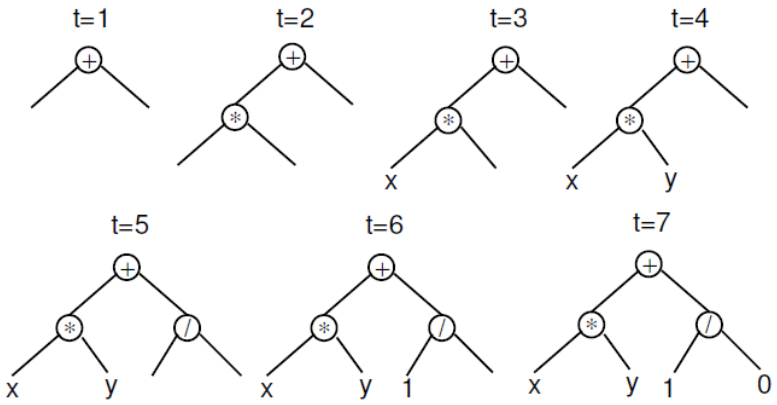

Figura 10: Passo a passo da inicialização de uma árvore utilizando o método *full.* Árvore com profundidade 2 (t=tempo).

O método chamado *grow* permite a criação de árvores de tamanho e formato mais variados. Os nós são escolhidos diretamente do conjunto primitivo, que é a união do conjunto de funções com o conjunto de terminais, até que o limite de profundidade seja atingido. Quando o limite é atingido somente terminais são escolhidos. A figura 11 mostra a sequência de criação de uma árvore inicial utilizando o método *grow* (POLI; LANGDON; MCPHEE, 2008).

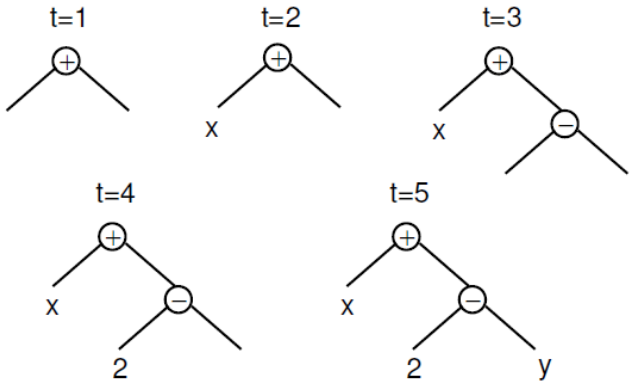

Figura 11: Passo a passo da inicialização de uma árvore utilizando o método grow. Árvore com profundidade 2 (t=tempo).

.

Como estes métodos explicados anteriormente não criam árvores com uma grande variedade de tamanhos e formas, foi introduzido em (KOZA, 1992) uma combinação destes métodos, em um método chamado de *ramped half and half.* Onde metade da população é criada utilizando o método *full* e a outra metade utilizando o método *grow.* Neste método utiliza-se um valor para a profundidade da árvore dentro de um intervalo, o que garante uma maior variedade de tamanho e formas para as árvores. A população inicial não precisa ser totalmente aleatória, se há o conhecimento de

propriedades desejáveis na solução desejada. Podem ser utilizadas árvores com estas propriedades para semear a população inicial. (POLI; LANGDON; MCPHEE, 2008)

Na programação genética a seleção dos indivíduos ocorre de maneira probabilística baseada na aptidão do mesmo. Ou seja, indivíduos com maior aptidão terão uma maior probabilidade de serem selecionados. O método mais comum de seleção utilizado em programação genética é o método chamado torneio. Neste método certa quantidade de indivíduos é escolhida aleatoriamente dentro da população. Eles são comparados entre si e o melhor deles é escolhido para ser o progenitor. Quando é feita a operação de cruzamento, são necessários dois progenitores, neste caso duas seleções do tipo torneio são feitas. Mesmo que este método beneficie os indivíduos mais aptos, ele não exclui os indivíduos com aptidões menores, o que sacrificaria a diversidade do conjunto de soluções. O método torneio aumenta pequenas diferenças de aptidão, pois dá preferencia para o melhor individuo mesmo que haja outros indivíduos com uma pequena diferença de aptidão dentro do torneio (POLI; LANGDON; MCPHEE, 2008).
Com relação aos operadores evolutivos, a programação genética prioriza o uso do operador cruzamento. O operador de cruzamento mais utilizado é o *subtree crossover.* Neste método é escolhido de forma aleatória e independente, um ponto de cruzamento (nó) em dois pais. A cria é formada retirando dos pais as *subtrees* cujas raízes são os pontos de cruzamentos escolhido e combinando o restante das árvores nestes pontos. Este processo é feito com copias dos pais selecionados, não eliminando assim os pais no processo. Um exemplo pode ser visto na figura 12. (POLI; LANGDON; MCPHEE, 2008)

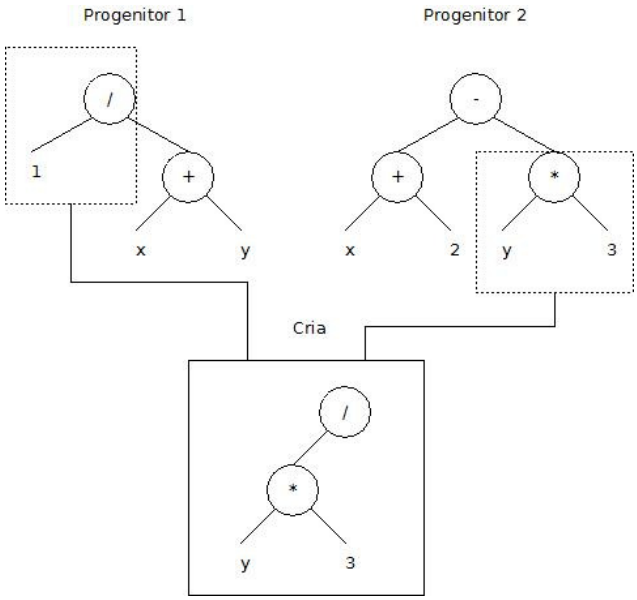

Figura 12: Exemplo de um Subtree crossover

O operador de mutação mais frequentemente utilizado é o *subtree mutation*, neste operador escolhe-se um ponto de mutação aleatoriamente e substitui-se a *subtree* cuja raiz é o ponto de mutação, por uma *subtree* gerada aleatoriamente. Como pode ser visto na figura 13. (POLI; LANGDON; MCPHEE, 2008)

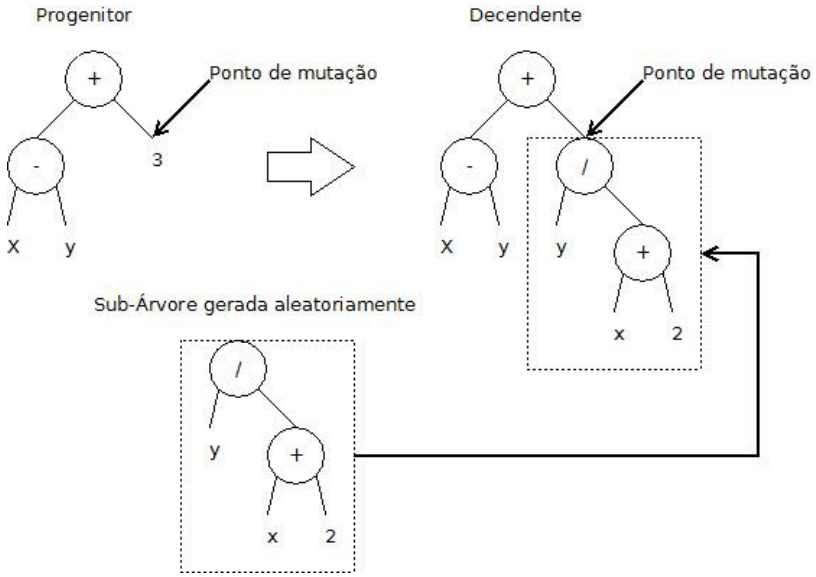

Figura 13: Exemplo de mutação do tipo *subtree*

Outra forma comum de mutação é a *point mutation*, no qual, cada nó da árvore tem a probabilidade de ser modificado por outra função dentro do conjunto de funções. Enquanto no método de *subtree*, apenas uma *subtree* é substituída. No *point mutation*, diversos nós podem ser substituídos em uma mesma aplicação do operador. (POLI; LANGDON; MCPHEE, 2008)

3.3 Code Growth ou Bloat

A despeito do sucesso inicial da Programação Genética, ela não teve um bom desempenho para grandes problemas. O maior fator para esta falha foi a ocorrência do *Bloat* (SOULE, 1998).

Bloat é a tendência dos programas gerados com a PG, ficarem muito maiores que o funcionalmente necessário. Este crescimento além de criar um código que não influência significativamente o desempenho, ele dificulta a busca por um código que vá realmente melhorar o mesmo. Pesquisas sobre o *Bloat* na PG indicam que este crescimento ocorre em qualquer técnica evolucionária que possua representações de tamanho variável (SOULE; HECKENDORN, 2002)

Quando alguém escreve um programa, é esperado que cada linha de código seja útil para algum propósito dentro do programa. Isto pode não ocorrer com os programas gerados pela programação genética. Os programas gerados pela PG podem ter diversas partes que não irão contribuir para o desempenho do programa.

O *Bloat* cria um tempo limite para busca, deixando a criação e a avaliação do programa mais demorada. Ele pode preencher totalmente a memória, levando a exaustão da mesma, diminuindo também a velocidade do processo de busca. A maior parte do *Bloat* consiste em códigos que não contribuem para o desempenho do programa. Estas partes do código são conhecidas como *Introns*. As partes do código que contribuem para o desempenho do programa são chamadas de *Exons*.

Foi Mostrado por Harries (HARRIES; SMITH, 1998) que o crescimento do código pode ser também causado por *Exons* que proporcionam um aumento insignificante no desempenho do programa.

Diversas hipóteses para o *Bloat* foram criadas, uma delas é a *Destructive Hypothesis*. Nesta teoria, é proposto que o *Bloat* ocorre para proteger o programa da operação de crossover, pois é intuitivo, que retirar uma parte de um programa funcional e substituir esta parte com um pedaço aleatório de outro programa, raramente irá aumentar o desempenho do primeiro. Alguns artigos confirmam esta intuição (BANZHAF, 1994; LANGDON; POLI, 1998). Soules (1998) formalizou um argumento demonstrando que a probabilidade de um crossover ser destrutivo é igual a uma fração do código viável. Um código ou *subtree* é considerado viável se existe um código ou *subtree* que substituindo o original irá alterar a aptidão, a saída ou ambos.

Em (MCPHEE; MILLER, 1995) Mcphee e Miller sugerem a existência de uma força secundária chamada de *Replication Accuracy Force* (RAF). O RAF age de forma a aumentar a acurácia dos replicadores conforme a evolução vai progredindo. Por exemplo, se um indivíduo com uma alta aptidão possui uma *subtree* que não é operacional e ocorre um cruzamento com ele mesmo, e ambos os pontos de cruzamento estão em *subtrees* que possuem código não operacional. Então o *crossover* irá criar dois indivíduos com a mesma aptidão porem com estruturas diferentes. Esta prole terá uma alta aptidão provavelmente irá fazer parte da reprodução na geração seguinte. Fazendo a população convergir para uma coleção de indivíduos equivalentes corretos, apresentando uma tendência ao aumento do tamanho das *subtrees* não operacionais. Um código ou *subtree* é considerado operacional se a remoção deste nó ou *subtree* irá modificar a aptidão do programa, da saída, ou ambos (Se a aptidão for baseada no desempenho e não no tamanho do programa).

Outra hipótese é a *Solution Distribution Hypothesis* (SOULE; HECKENDORN, 2002). Esta hipótese explica a ocorrência do *Bloat* não só na Programação Genética, mas também em outros sistemas que usam representações discretas de comprimento variável, ela diz que a aptidão causa o *Bloat* e é baseada na estrutura do espaço de buscas do programa. Em muitos problemas, a quantidade de programas com uma determinada aptidão e um tamanho, por exemplo, maior que y, é muito maior que a quantidade de programas com a mesma aptidão e com tamanho menor que y. Em parte esta propriedade do espaço de busca é causada pelos *Introns*. Dado um programa com aptidão x e tamanho y, é possível criar infinitas variações deste programa adicionando *Introns*. Como programas maiores são mais comuns no espaço de busca, há uma maior probabilidade que um programa de tamanho maior seja escolhido como solução. Esta hipótese segue a ideia que uma técnica de busca estocástica tende a encontrar a solução mais comum dentro do espaço de buscas, neste caso seriam os programas maiores. Ou seja, a seleção baseada em aptidão direciona o processo evolutivo a soluções maiores.

Outra hipótese é a *Removal Bias* Nesta teoria proposta por T. Soule and J. A. Foster (SOULE; FOSTER, 1998). Ela assume que a maior concentração de *Introns* está presente perto das folhas das árvores. Estando os códigos inviáveis concentrados perto das folhas, a remoção de uma pequena *subtree* provavelmente irá afetar um código inviável enquanto a remoção de uma *subtree* maior terá uma maior probabilidade de afetar um código viável. Se uma *subtree* for adicionada no meio de uma seção de código inviável, ela não produzirá outro efeito além do aumento do tamanho da seção. Esta

hipótese diz que existe uma tendência a criação de descendentes com a remoção de pequenas *subtrees* ao invés de *subtrees* grandes, não havendo tendências em *subtrees* adicionadas, esta tendência irá progressivamente criar uma população com indivíduos maiores.

A partir da metade dos anos 90 diversos pesquisadores propuseram extensões para a PG baseados em programas com a forma de grafos, alguns exemplos são o *Parallel Distributed Genetic Programming* (PDGP) que é uma extensão da PG criada para desenvolver programas com alto grau de paralelismo, reutilizando resultado parciais de forma eficiente, outro exemplo é o *Parallel algorithm Discovery and orchestration* (PADO) que usa discriminação linear em combinação com a PG para obter programas de classificação paralelos para sinais e imagens. Outro exemplo que pode ser citado e é de especial interesse nesta dissertação é a Programação Genética Cartesiana.

3.4 Programação Genética Cartesiana

3.4.1 Introdução a Programação genética cartesiana

A Programação genética cartesiana (PGC) (MILLER; THOMSON, 2000) é uma forma de programação genética no qual os programas são representados por grafos. O grafo é codificado em uma sequência linear de inteiros e são representados em uma grade de nós computacionais de n-dimensões, com a extensão definida pelo programador, mas que, em geral possui uma ou duas dimensões. Este tipo de programação genética recebe o nome de Cartesiana, pois os nós são endereçados pelo sistema de coordenadas cartesianas. Uma das motivações da utilização de grafos ao invés da árvore utilizada na programação genética convencional é o fato de que grafos são mais gerais, flexíveis e compactos e podem ser aplicados em diversos domínios (DHARWADKER; PIRZADA, 2011). Outra motivação foi a criação de um algoritmo que fosse mais efetivo no aprendizado de funções booleanas que outros métodos de programação genética. A PGC foi vagamente inspirada no arranjo da arquitetura dos *Field Programmable Gate Arrays* (FPGA). A PGC tem uma maior facilidade de lidar com sistemas de múltiplas saídas. (MILLER; HARDING, 2009; MILLER; THOMSON, 2000; MILLER, 2011). Outras vantagens da PGC são a características de neutralidade, a

redundância e a ausência de um problema chamado *Bloat* comum em outros métodos de programação genética (BANZHAF, 1994; MILLER; SMITH, 2006; MILLER, 2001).

Na PGC, os programas são representados em uma sequência linear de números inteiros, que é denominada cromossomo ou genótipo. Cada inteiro representa um gene. Os genes podem ser divididos em genes de função, conexão ou de saída. A figura 14 apresenta um cromossomo ou genótipo. (MILLER; HARDING, 2009; MILLER; THOMSON, 2000; MILLER, 2011).

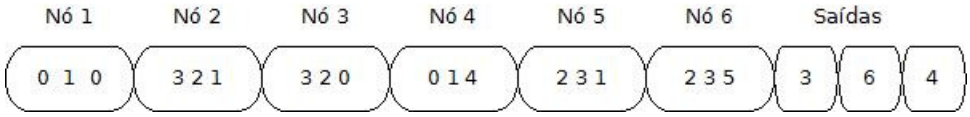

Figura 14: Genótipo ou cromossomo

A junção de um gene de função com alguns genes de conexão formam um nó. A Estrutura de cada nó consiste em uma função (gene de função) em que são aplicados os valores de entrada cuja localização é dada pelo gene de conexão, para gerar uma saída, a função é escolhida dentro de um conjunto de funções previamente definidas de acordo com a aplicação, um exemplo pode ser visto na tabela 3. A entrada de cada nó pode ser uma entrada do sistema ou a saída de um nó de uma coluna anterior. É chamado de *Level-Back*, o máximo de colunas anteriores que um gene de conexão pode se conectar. A quantidade de entradas dos nós é chamada de aridade, e é determinada de acordo com a função que necessita do maior número de entradas entre as funções do conjunto. Tanto o *Level-back* como a aridade, são parâmetros a serem definidos pelo programador (MILLER; HARDING, 2009; MILLER; THOMSON, 2000; MILLER, 2011).

Tabela 3: Exemplo de um conjunto de funções

Referência da função	Função
0	Adição
1	Subtração
2	Multiplicação
3	Divisão

O processo evolutivo ocorre nos genótipos, sendo necessária uma decodificação deste genótipo em fenótipos cuja formatação está no domínio da solução do problema.

Quando o genótipo é decodificado alguns nós podem estar desativados. Isso acontece, pois alguns nós não estão ligados à saída de dados, o que gera um efeito de neutralidade. O programa gerado pela decodificação do genótipo é chamado de fenótipo. Enquanto os genótipos tem um tamanho fixo, os fenótipos podem variar de tamanho. A figura 15 mostra um exemplo de genótipo e o fenótipo correspondente, em uma aplicação cujo objetivo é gerar as funções matemáticas das equações 8, 9, 10 e 11. Nesta figura, os círculos representam os pontos de conexão e os quadrados as funções. Foi utilizado o conjunto de funções da tabela 3 (MILLER, 2011).

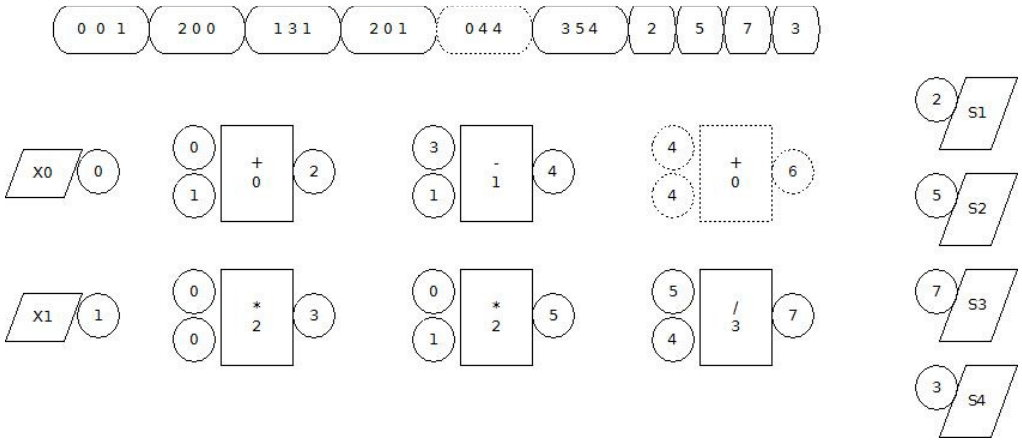

Figura 15: Genótipo e o fenótipo correspondente

$$S1 = X_0 + X_1 \quad (8)$$

$$S2 = X_0 * X_1 \quad (9)$$

$$S3 = \frac{X_0 * X_1}{{X_0}^2 - X_1} \quad (10)$$

$$S4 = {X_0}^2 \quad (11)$$

3.4.2 Mutação

O operador mutação é o operador mais importante na PGC. É comumente utilizada a mutação em um ponto. Neste caso, um gene escolhido aleatoriamente, tem o seu valor modificado para outro que seja válido. Podem ser modificados tantos genes de função, conexão, como de saída. A quantidade de genes que podem ser alterados a cada aplicação do operador mutação geralmente é definida como um porcentagem do total de genes do genótipo e é denominada taxa de mutação. As figuras 16 e 17 mostram o antes

e depois de uma operação de mutação. Neste exemplo, o ultimo gene do genótipo foi alterado, pode-se ver que a alteração deste único gene pode alterar a conexão de diversos nós, mudando significativamente o fenótipo (MILLER; HARDING, 2009; MILLER; THOMSON, 2000; MILLER, 2011).

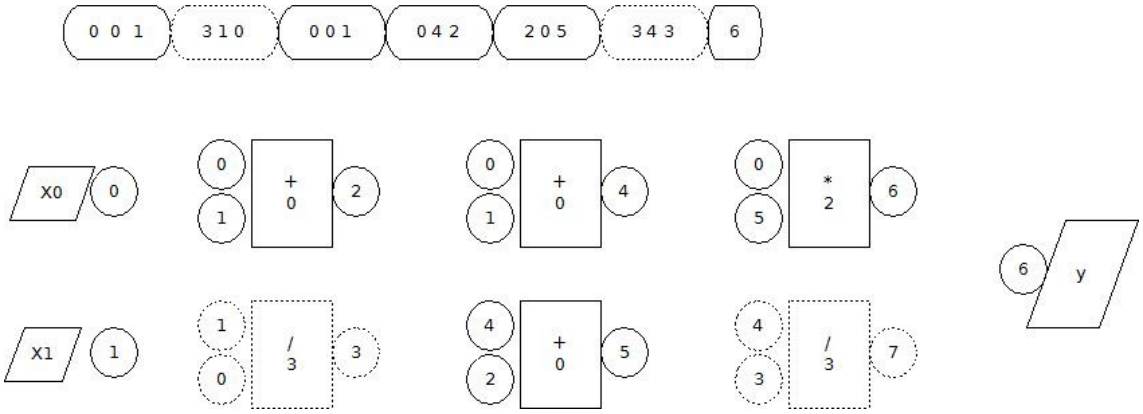

Figura 16: Genótipo-Fenótipo antes da mutação.

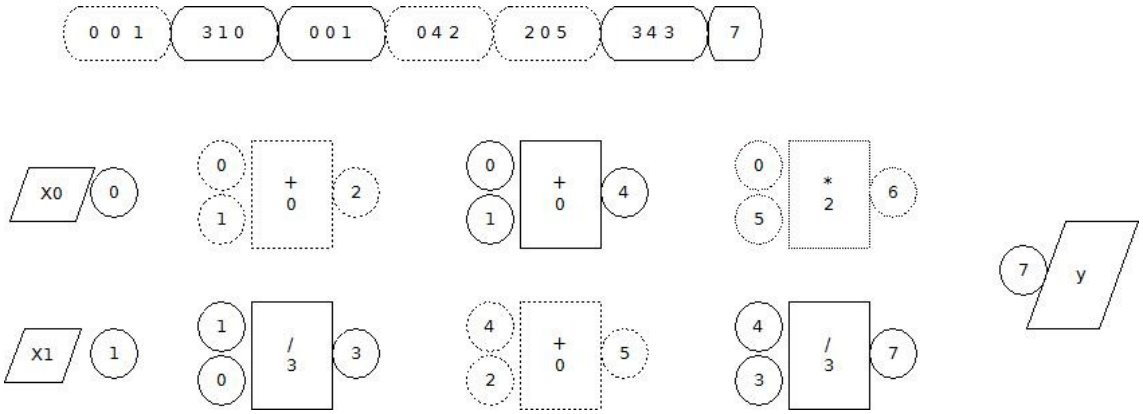

Figura 17: Genótipo-Fenótipo após mutação

3.4.3 Estratégia evolutiva na PGC

O algoritmo mais utilizado na PGC é a estratégia evolutiva 1+λ. Onde normalmente o valor de λ é 4. O algoritmo da estratégia evolutiva pode ser descrito nos seguintes passos: Cinco Indivíduos (genótipos) são criados aleatoriamente. O indivíduo que possui o maior valor da função de aptidão será considerado o mais apto e será promovido para a população da geração seguinte. O operador mutação é aplicado quatro vezes no indivíduo mais apto da geração anterior, gerando quatro novos indivíduos, totalizando cinco indivíduos na população atual. É escolhido novamente o

indivíduo mais apto. Caso haja mais de um individuo com a melhor aptidão, sempre será promovido para a próxima geração aquele individuo que tenha sido criado na geração atual. O processo anterior é repetido até que um critério de parada seja atingido. Este processo de escolha é ilustrado na figura 18.

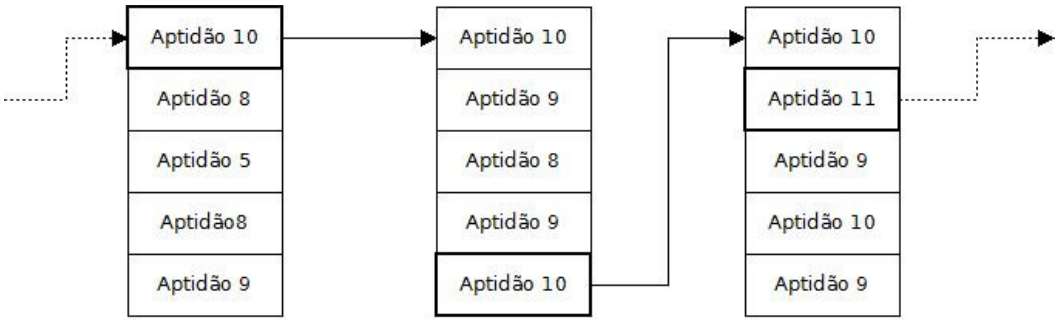

Figura 18: Representação de como é feita a escolha do genótipo dentro da população.

3.4.4 Redundância, Mapeamento Genótipo-Fenótipo e Neutralidade

Na PGC existem três formas de redundância. Uma delas é a redundância de nó, que ocorre quando genes associados a um nó, não fazem parte do grafo que está conectado, ou seja, dos nós que estão conectados a saída através de outros nós ou ligados diretamente. Este tipo de redundância é grande no começo do processo evolucionário pelo fato que muitos nós não estão conectados nas populações iniciais que são criadas aleatoriamente. Esta redundância irá diminuir gradualmente até atingir um valor entre a quantidade de nós necessários para obter uma solução satisfatória e o total de nós permitidos.

Outra forma de redundância é a redundância funcional, que está presente em todas as outras formas de programação genética, no qual uma sub-função implementada no programa, poderia ser implementada de uma forma diferente utilizando um número menor de nós. Este aumento no número de nós com redundância funcional é chamado de *Bloat* (MILLER; SMITH, 2006; MILLER, 2001), como foi descrito na seção anterior. Um programa com muito *Bloat* provoca um aumento desnecessário no consumo de tempo na avaliação de aptidão, uma redução na eficiência dos operadores de variação e o programa pode exibir uma baixa generalização.

Na PGC existem duas formas de *Introns*: os que ocorrem no genótipo e os que ocorrem no fenótipo. Geralmente o genótipo possui uma grande quantidade de genes

inativos e a mutação permite que uma grande quantidade de genótipos com a mesma aptidão, sejam mapeados em um único fenótipo. As constantes mudanças genéticas nestes genótipos com mesma aptidão tem sido apontado como a maior razão da eficiência da PGC na resolução de problemas (A. GARMENDIA-DOVAL, 2006; YU; MILLER, 2001, 2002).

A redundância funcional pode contribuir também de forma positiva aumentando a quantidade de formas com que uma sub-função do programa pode ser feita e a possibilidade de desconectar nós permite a PGC minimizar o problema do *Bloat.* A PGC vem sendo utilizado a mais de uma década em diversas aplicações e foi verificado através das experiências que o *Bloat* não ocorre de maneira significante na PGC, os programas usando a PGC só aumentam de tamanho se for necessário para o aumento da aptidão (MILLER, 2001).

De acordo com (MILLER, 2001), o fato da pouca ocorrência de *Bloat* na PGC está ligada a presença de genes que podem ser ativados e desativados.

A terceira forma de redundância é a redundância de entrada, que ocorre quando algumas funções do conjunto de funções não utilizam todas as entradas possíveis. Esta forma de redundância ocorre quando o conjunto de funções possui funções com diferentes aridades.

O mapeamento genótipo-fenótipo (BANZHAF, 1994) consiste em separar o espaço de busca dos genótipos do espaço de soluções dos fenótipos, muitos genótipos podem resultar em um mesmo fenótipo, ou seja, muitos genótipos com mesma aptidão , essa é uma vantagem deste mapeamento, pois estas variantes neutras são frequentes e importantes para a variabilidade genética, esta característica é chamada de neutralidade.

A importância da neutralidade é reconhecida por diversas teorias modernas da evolução natural das moléculas (KIMURA, 1968)(OHTA; GILLESPIE, 1996) e sua versão presente na programação genética é igualmente importante. A neutralidade faz com que o processo evolutivo ultrapasse regiões onde as saídas teriam baixa “aptidão”, evitando a estagnação em pontos sub-ótimos. Mesmo que uma mutação passiva não altere a aptidão imediatamente, ela pode ser utilizada em uma melhora futura. As diversas experiências feitas com a PGC (Miller, 2011) levam a crer que a busca é mais efetiva quando 95% dos genes estão inativos, resultado que acentua o benefício da neutralidade.

4 MÉTODO PROPOSTO

Este capítulo apresenta um novo método para síntese de Sistemas Fuzzy (SF). Este método procura incorporar estratégias bem sucedidas na síntese de SF e seguir as tendências da pesquisa.

Conforme foi visto no Capítulo 1, uma destas tendências é o aprendizado de diferentes estruturas de Sistemas Fuzzy. O método aqui descrito propõe a utilização de Árvores de Padrões Fuzzy(APF) (HUANG; GEDEON; NIKRAVESH, 2008) que é uma estrutura alternativa aos Sistemas Fuzzy Baseados em Regras. Esta estrutura foi escolhida porque apresenta algumas características interessantes que serão descritas a seguir: (1) as APF possuem um mecanismo de seleção de atributos embutidos que auxilia no tratamento da "maldição da dimensionalidade"; (2) Pode-se também argumentar que são atraentes do ponto de vista de interpretação do resultado obtido, uma vez que elas apresentam uma descrição "lógica"[4] de cada classe; (3) sua estrutura hierárquica permite identificar quais variáveis são mais importantes.

O aprendizado de Sistemas fuzzy por programação genética também se constitui em outra tendência conforme os trabalhos de (GEYER-SCHULZ, 1997), (BERLANGA et al., 2006), (BASTIAN, 2000). Sendo assim, propôs-se a utilização de Programação Genética Cartesiana na indução de APFs. A PGC é um tipo de programação genética na qual os programas são representados por grafos. Esta representação pode ser facilmente utilizada para representar APF. Além disso, a PGC é um algoritmo de busca global capaz de explorar grandes espaços de forma eficiente. A exploração do espaço de busca na PGC não segue a estratégia gulosa do *beam search*, portanto tem mais chances de obter soluções melhores. Além disso, a exploração do espaço não é feita de forma restrita como no caso da *beam search* (que é limitada pela largura do feixe), o que também aumenta as chances de obter soluções melhores.

A utilização da PGC também facilita a utilização de operadores parametrizados, pois o operador de mutação é capaz de alterar os parâmetros destes operadores com o objetivo de melhorar o desempenho.

[4] A descrição não é inteiramente lógica pois operadores aritméticos também são permitidos

A seguir serão descritas as etapas necessárias para a síntese das APFs e as escolhas de projeto realizadas. A figura 19 apresenta um diagrama de blocos do modelo.

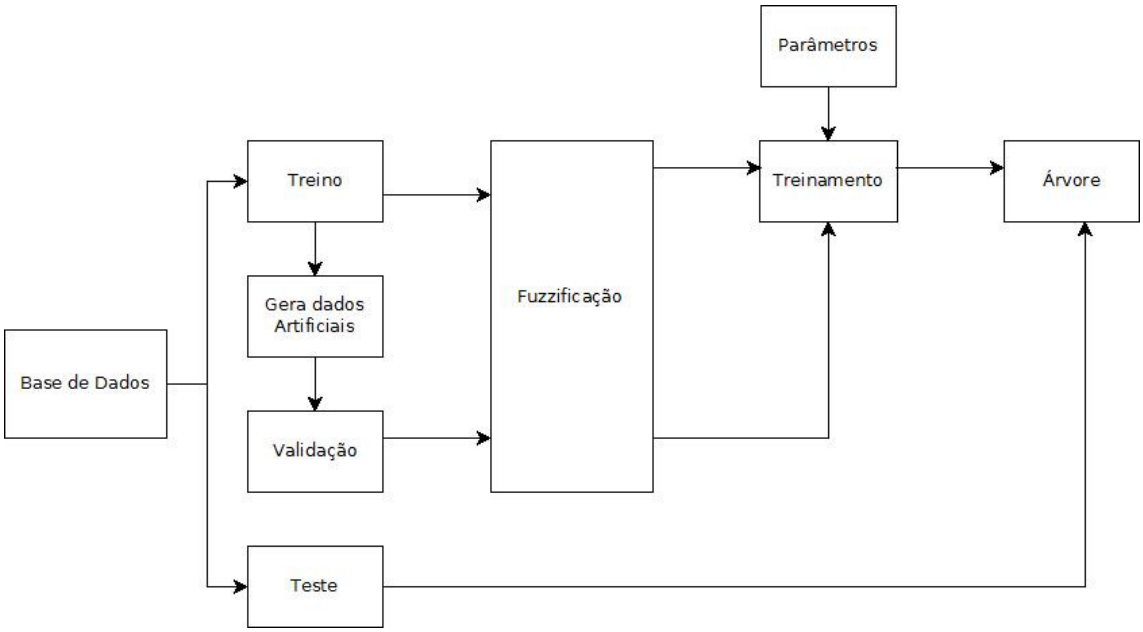

Figura 19: Diagrama de blocos do modelo proposto

4.1 Particionamento Fuzzy

O particionamento do conjunto de dados disponível é normalmente realizado em um conjunto de treinamento, que será utilizado no processo de obtenção da estrutura da APF e no ajuste dos operadores parametrizados, e o conjunto de teste será utilizado posteriormente para se obter uma medida da capacidade de generalização do algoritmo.

Seguindo, o arcabouço descrito no Capítulo 2, tem-se que o conjunto de treinamento é expresso pela Eq. 12:

$$T = \{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{n} \subset X \times Y \tag{12}$$

Onde cada instância $x^{(i)}$ é um vetor

$$\boldsymbol{x} \in X = X_1 \times X_2 \times \cdots \times X_m$$

X_i é o domínio do i-ésimo atributo A_i. Cada domínio é particionado em um conjunto de funções de pertinência $F_{i,j}$

$$F_{i,j}: X_i \rightarrow [0,1] \qquad (j = 1, \ldots, n_i)$$

De tal modo que $\sum_{j=1}^{n_j} F_{i,j}(x) > 0, \forall\, x \in X_i.\ Fi,j$ normalmente recebe um rótulo linguístico.

No método proposto, cada atributo teve seu domínio dividido em cinco termos linguísticos (termos *fuzzy*), denominados “Baixo”, ”Médio-Baixo”, “Médio”, “Médio-Alto”, “Alto”. A figura 20 exemplifica a partição de um atributo, cujo domínio é o intervalo [0,1]. O valor de cada atributo ativa a função de pertinência de cada termo fuzzy produzindo um valor de pertinência no intervalo [0,1].

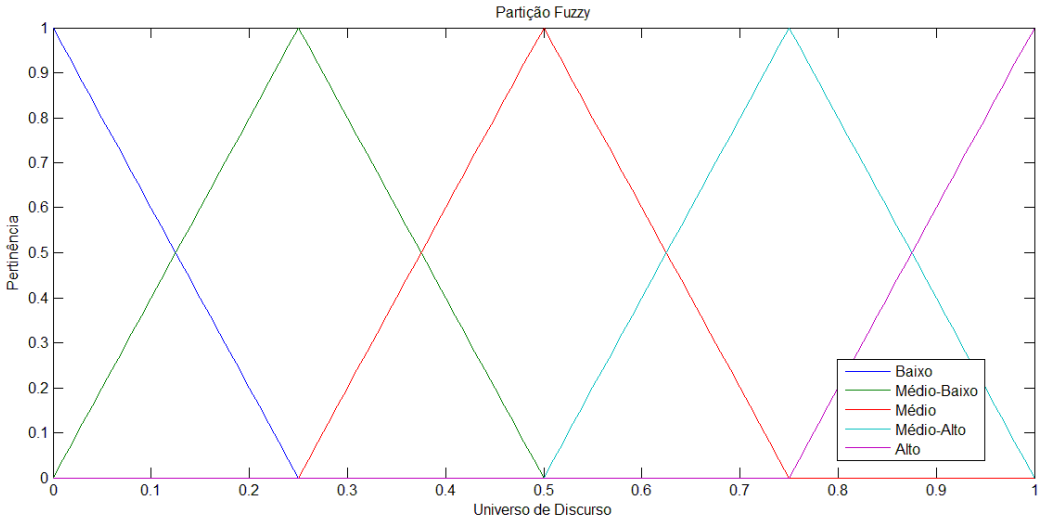

Figura 20: Partição Fuzzy

Cada atributo presente no banco de dados terá um valor de pertinência associado a cada um dos termos fuzzy. Esta relação entre atributos e termos fuzzy serão as entradas da grade computacional da PGC, somando um total de entradas de cinco vezes a quantidade de atributos.

Uma estratégia bastante utilizada para evitar o *overfitting* e a consequente perda de generalização, é a utilização de um terceiro conjunto de dados, chamado de conjunto de validação ou conjunto de ajuste. Normalmente este conjunto de validação é obtido dividindo-se o conjunto de treinamento e utilizando uma parte dele para o treinamento propriamente dito e a outra parte na parada antecipada. Entretanto, em muitas situações onde o conjunto de treinamento é pequeno, pode-se adotar uma outra estratégia: gerar este conjunto de validação artificialmente (DUIN, 1996). Utilizou-se o conjunto de treinamento para estimar a função de densidade de probabilidade que gerou os dados do conjunto. A partir desta função de densidade de probabilidade, dados artificiais são

gerados com características similares ao conjunto de treinamento. A estimativa de densidade de probabilidade pode ser feita através de Janelas de Parzen ou por vizinhos mais próximos (DUDA; HART; STORK, 2012).

4.2 Operadores

Diversos operadores são utilizados na APF, como mostrado na seção 2. Porém com a intenção de aumentar a compreensão da solução (árvore), decidiu-se reduzir a quantidade de operadores. Os operadores utilizados foram reduzidos para os seguintes operadores:

$$\text{Máximo}=\text{Max}(a,b) \tag{13}$$

$$\text{Mínimo}=\text{Mín}(a,b) \tag{14}$$

$$\text{WA}=x*a+(1-x)*b \tag{15}$$

$$OWA=x*Máx(a,b)+(1-x)*Min(a,b) \tag{16}$$

Observe que os operadores WA e OWA são operadores parametrizados. onde *a* e *b* são os valores de entradas dos nós que serão operados e *x* será um valor aleatório dentro do intervalo [0,1], que poderá ser ajustado pelo operador de mutação. Os operadores WA e OWA são utilizados para preencher o espaço entre a maior combinação conjuntiva (t-norma) e a menor combinação disjuntiva (t-conorma). Os operadores foram codificados em números inteiros de acordo com a tabela 4.

Tabela 4: Operadores utilizados é seus respectivos códigos

WA	0
OWA	1
Mínimo	2
Máximo	3

4.3 Treinamento

Esta é a etapa mais importante do modelo, pois nela ocorre o aprendizado. As entradas para este bloco do modelo são os conjuntos de treino, validação e um conjunto de parâmetros definidos previamente. Estes parâmetros são: a quantidade de linhas e

colunas na grade computacional da PGC, o *levelback*, a quantidade máxima de gerações, o tamanho da população, a taxa de mutação do genótipo, a função utilizada para a avaliação do genótipo e seus parâmetros. O primeiro passo nesta etapa é a criação da população de genótipos inicial. Ela é criada de forma aleatória obedecendo aos limites impostos pelos parâmetros linhas, colunas, *levelback* e tamanho da população do modelo. Neste projeto é utilizada a estratégia evolutiva 1+ λ. Sendo escolhido λ=4. O total de genótipos criados pode mudar de acordo com duas variantes da relação genótipo-classes que foram criadas para este modelo.

4.3.1 Variações do Genótipo

Foram criadas duas variantes no projeto para a relação genótipo-classe. Na primeira é criado um genótipo para cada classe como resultado do processo de aprendizado. Neste caso os genótipos possuem apenas uma saída e o processo evolutivo ocorre de maneira independente dos outros genótipos. Esta variante (uma árvore para cada classe) foi utilizada no algoritmo original (HUANG; GEDEON; NIKRAVESH, 2008) e também em uma outra versão do algoritmo (SENGE; HÜLLERMEIER, 2011). Devido a estratégia evolutiva utilizada, a população neste primeiro caso será igual a cinco vezes o número de classes. Na segunda variante é criado apenas um único genótipo com *n* saídas, onde cada saída representa uma classe e as alterações devido à mutação do genótipo podem influenciar diversas classes (árvores) ao mesmo tempo. A população neste caso será sempre igual a cinco. A figura 21 apresenta como seriam o genótipo ou genótipos de saída nas duas variantes. Os genótipos são construídos por uma sequência de números inteiros. Neste modelo foi fixado o número de linhas da PGC em 1, neste caso o tamanho do genótipo irá mudar somente com a variação do número de colunas. Cada coluna possui um nó e cada nó possui 3 genes. O primeiro gene do nó, refere-se a um dos operadores de acordo com a tabela 4. O segundo e terceiro genes do nó representam os genes de conexão, ou seja os pontos onde as entradas desse nó estão conectada, seja uma entrada ou a saída de um nó anterior. A quantidade total de genes em um genótipo será igual ao número de colunas vezes três, mais os genes de saída.

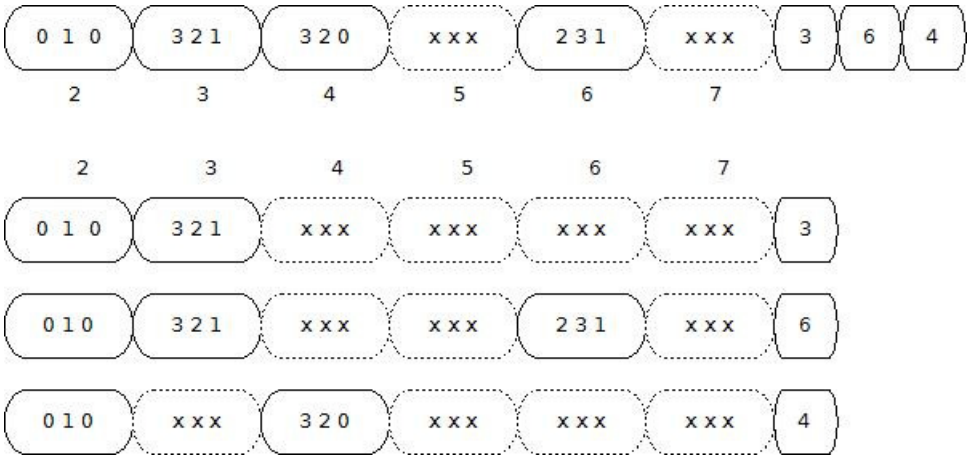

Figura 21: Genótipos das duas variantes do modelo. Neste exemplo os genótipos possuem apenas duas entradas, 0 e 1 e 3 saídas (Classes).

A figura 22 apresenta a diferença entre o algoritmo de treinamento das duas variantes.

```
Criar População Inicial Aleatoria (Linhas,Colunas,Levelback,tamanho da população)
Para Geração de 1 até Limite máximo de gerações
  Para Individuo de 1 até Tamanho da População
    Avaliar(Indivíduo );
  fim
  Aplicar Estrategia Evolutiva 1+ λ;
  Testar Critérios de parada;
fim
```

```
Criar População Inicial Aleatoria (Linhas,Colunas,Levelback,tamanho da população)
Para Geração de 1 até Limite máximo de gerações
  Para Classe de 1 até o total de classes
    Para Indivíduo de 1 até Tamanho da População
      Avaliar(Indivíduo );
    fim
    Aplicar Estrategia Evolutiva 1+ λ;
  fim
  Testar Critérios de parada;
fim
```

Figura 22: Algoritmos do treinamento das duas variantes genótipo-fenótipo.

4.3.2 Avaliação

A avaliação dos genótipos proposta é composta por duas parcelas como pode ser visto na equação 17. Ela apresenta a forma mais simples de função de aptidão multiobjectivo, que é aquela composta por ponderação de objetivos. Estes objetivos desejados são a acurácia e a interpretabilidade.

$$Aptidão = w_1 * AP_1 + w_2 * AP_2 \quad (17)$$

Onde *w*1 e *w*2 são pesos a serem atribuídos, desde que *w*1+*w*2=1.

A primeira parcela refere-se a função de avaliação que trata da acurácia. Diversas funções podem ser utilizadas, como por exemplo, Área sob a Curva ROC AUC (FAWCETT, 2006), F-Measure (FERRI; HERNÁNDEZ-ORALLO; MODROIU, 2009) e RMSE (WITTEN; FRANK, 2005) A similaridade obtida a partir do RMSE (Root Mean Squared Error), que foi utilizada em algoritmos anteriores, é uma medida baseada em distância, definida pelas equações 18,19.

$$Rmse = \sqrt{\frac{\sum_{i=1}^{N}(d_i - o_i)^2}{N}} \tag{18}$$

$$Similaridade = 1 - Rmse \tag{19}$$

O RMSE produz como resultado a raiz quadrada do erro quadrático médio, que é definido a partir da diferença entre valor obtido na saída da árvore (o_i) e o valor alvo (d_i), que deve ser 1 se os valores apresentado na entrada pertencem a um ponto da classe que a árvore deve representar e 0 caso contrário. O RMSE é subtraído de um, resultando na medida de desempenho chamada similaridade.

A segunda parcela da avaliação dos genótipos é descrita pela equação 20. Ela procura representar a interpretabilidade.

$$AP_2 = \frac{(NumeroTotaldegees - GeesAtivos)}{NúmeroTotaldegenes} \tag{20}$$

A equação (20) produz um valor maior, quanto menor fora quantidade de genes ativos em comparação a quantidade total de genes. Os genes ativos são aqueles para os quais a saída está conectada a algum outro gen. Esta segunda parcela procura fornecer uma melhor avaliação para árvores menores. Quanto maior for uma árvore, mais complexa será a expressão "lógica" que representará a classe e mais difícil será a compreensão da expressão obtida.

4.3.3 Critérios de Parada

Dentro do método proposto foram criados três critérios de paradas. O primeiro é a parada pela quantidade total de gerações. Esta parada é acionada quando a quantidade

de gerações passadas atinge um valor pré-estabelecido. Geralmente os outros critérios serão acionados antes deste, porem ele é útil para limitar o tempo total de execução caso os outros critérios não sejam acionados.

O segundo critério é acionado se não houver uma melhora significativa da aptidão dentro de uma quantidade determinada de gerações passadas. Entretanto, deve-se levar em consideração a característica de neutralidade da PGC. Portanto, a quantidade de gerações passadas não deve ser muito limitada, pois pode haver genes inativos em um determinado momento que irão aumentar a aptidão de uma forma significativa no futuro.

O terceiro critério é acionado se a aptidão do conjunto de validação na geração atual for menor que a aptidão em um número estipulado de gerações passadas, ou seja, se houver uma queda consistente na aptidão da validação. É passado para a próxima geração o genótipo que representa a geração em que a aptidão do conjunto de validação atingiu o seu máximo.

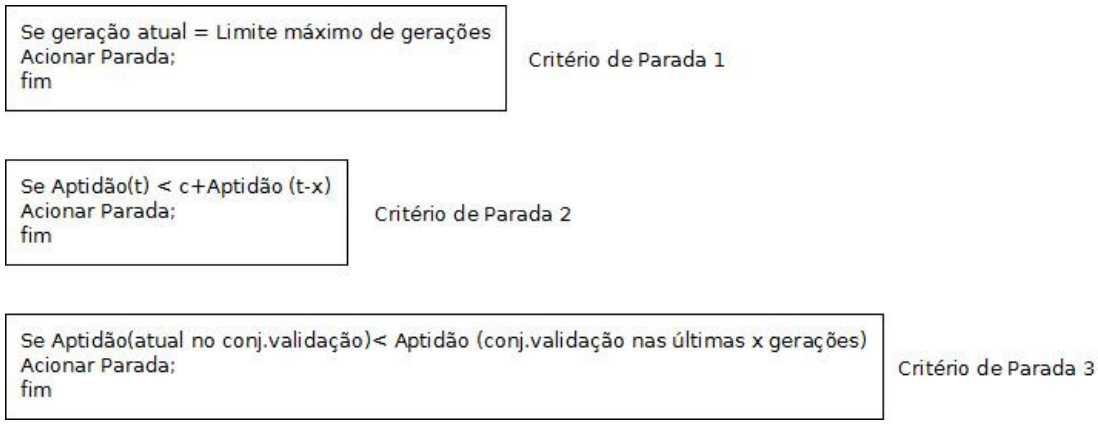

Figura 23: Critérios de Parada.

4.4 Árvore

Como resultado do treinamento, serão obtidos o genótipo ou genótipos de acordo com a variação utilizada. É aplicado um método de decodificação do genótipo, que parte da saída e varre o genótipo para encontrar os genes que estão conectados a esta saída. Este método será repetido para todos os nós conectados até que se tenha a informação de quais os nós que estão conectados para formar a solução e também a ordem de conexão desses nós. Com esta informação será construída a árvore. A Tabela

4 mostra os códigos utilizados para os operadores. A figura 24 mostra a árvore obtida usando a variante de um genótipo para n classes. A figura 25 mostra as árvores obtidas usando a variante de um genótipo para cada classe. As figuras 24 e 25, apresentam as árvores geradas a partir dos genótipos do exemplo da seção 4.2.1

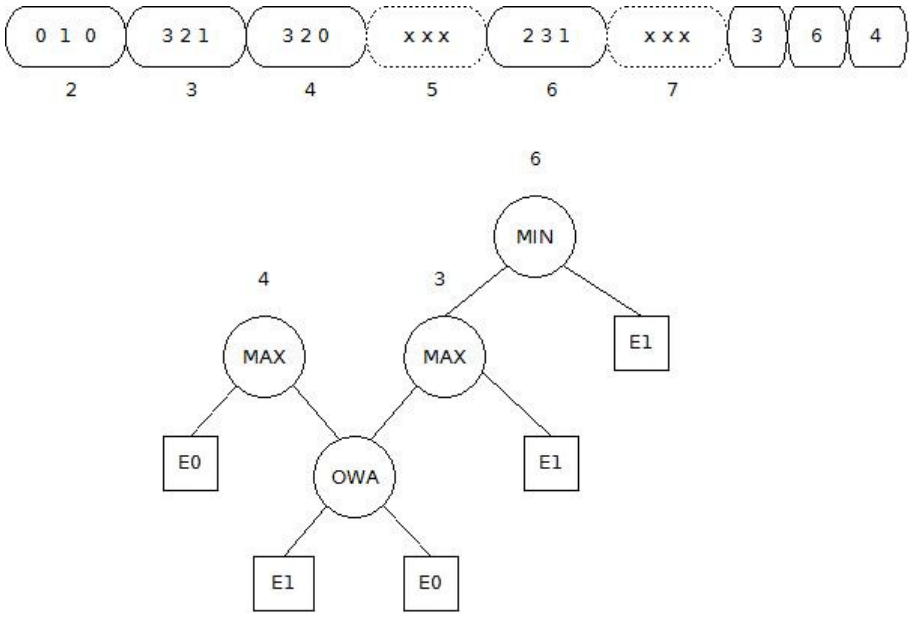

Figura 24: Genótipo e a sua respectiva árvore

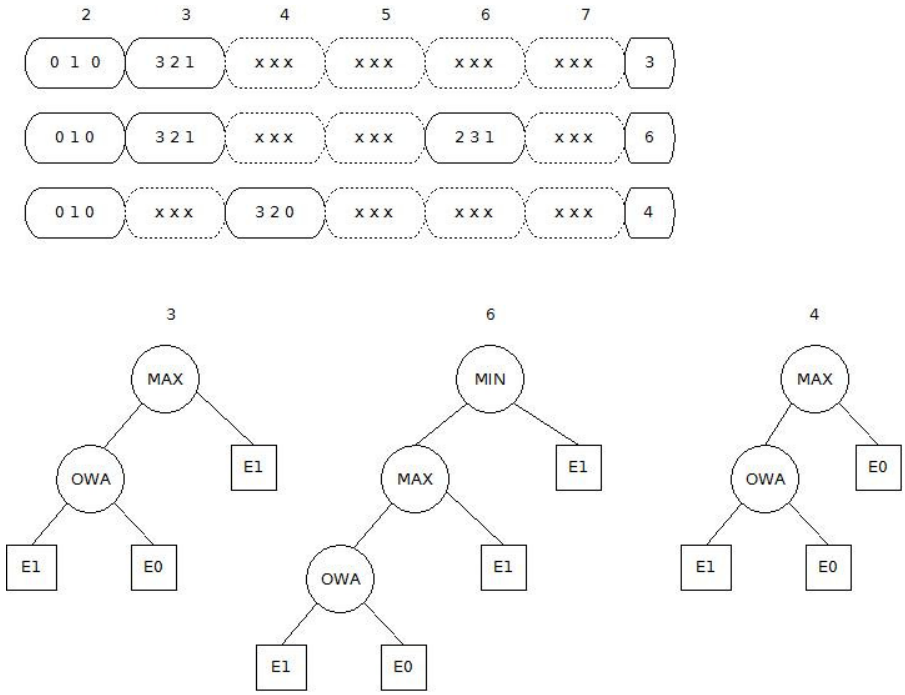

Figura 25: Genótipo e a sua respectiva árvore

5 ESTUDO DE CASOS

5.1 Estudo de casos com bases de dados artificiais

O método de síntese de árvores de decisão fuzzy através de programação genética cartesiana possui dois conjuntos de parâmetros: o primeiro conjunto está relacionado diretamente com a programação genética cartesiana, dentre os quais se podem citar: o número de indivíduos na população, o número de gerações, a função de aptidão, a taxa de mutação, a quantidade de linhas, a quantidade de colunas e o *levelback*. O segundo conjunto de parâmetros está ligado a árvore propriamente dita tais como: a escolha dos operadores, o número de partições fuzzy, o tamanho dos conjunto de treinamento e de teste.

Nesta seção será apresentado um estudo de casos realizado com bases de dados artificiais. Estas bases foram utilizadas em trabalhos anteriores para avaliar algoritmos de aprendizado (KRIJTHE; HO; LOOG, 2012),(BREIMAN, 1996),(LOCAREK-JUNGE; WEIHS, 2010),(SCHIFFNER, 2010). A tabela 5 indica quais foram as bases de dados utilizadas e a quantidade de pontos, atributos e classes de cada uma. As figuras 26, 27, 28 e 29 mostram os conjuntos de dados em duas dimensões.

Tabela 5: : Bases de dados artificiais

Base de Dados	Pontos	Atributos	Classes
Duas Espirais	1000	2	2
Gaussiana	900	4	6
XOR	1000	4	8
Difficult2	300	2	2
Difficult6	300	6	2
Difficult10	300	10	2
Difficult20	300	20	2
Threenorm	1000	20	2

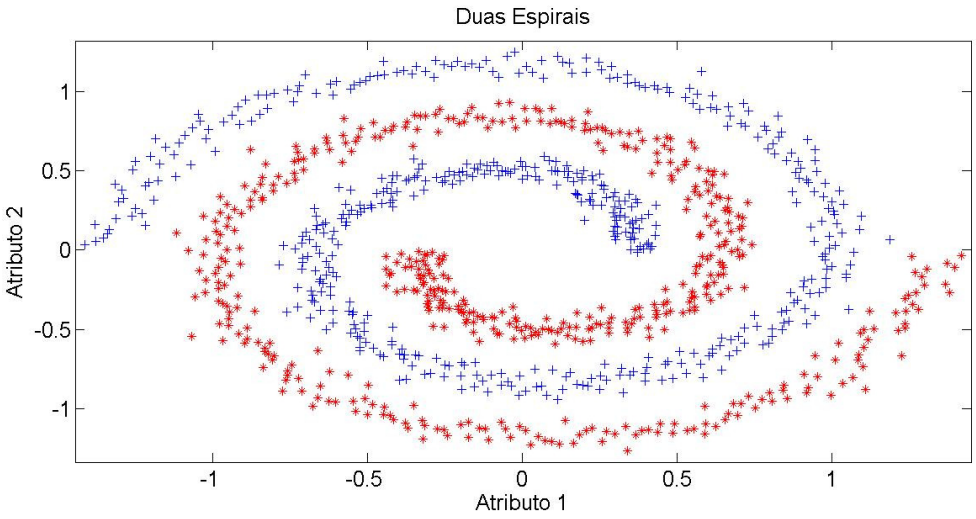

Figura 26: Base de dados - Duas Espirais

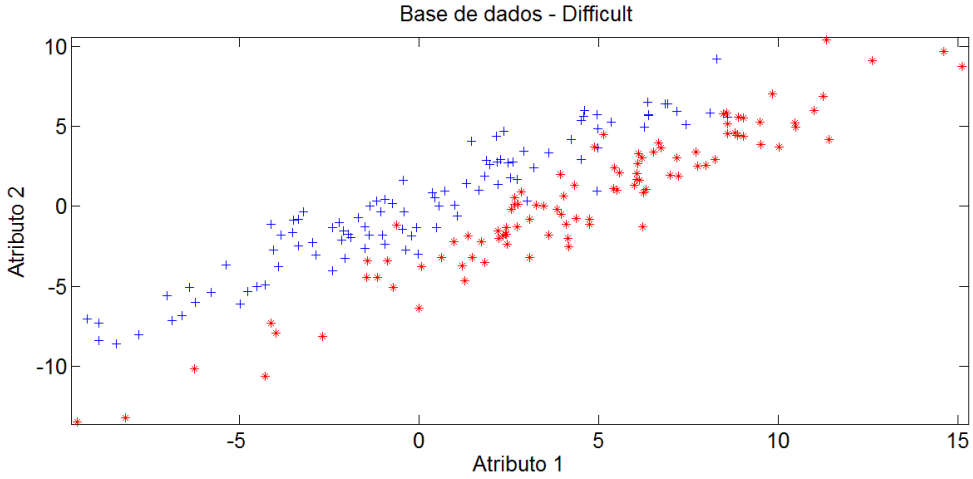

Figura 27: Base de dados - Difficult

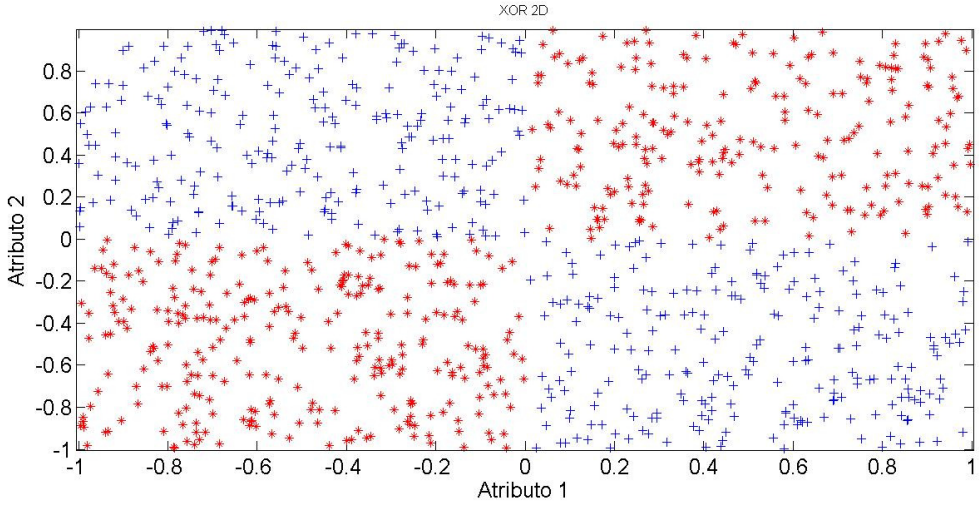

Figura 28: Base de dados - XOR em duas dimensões

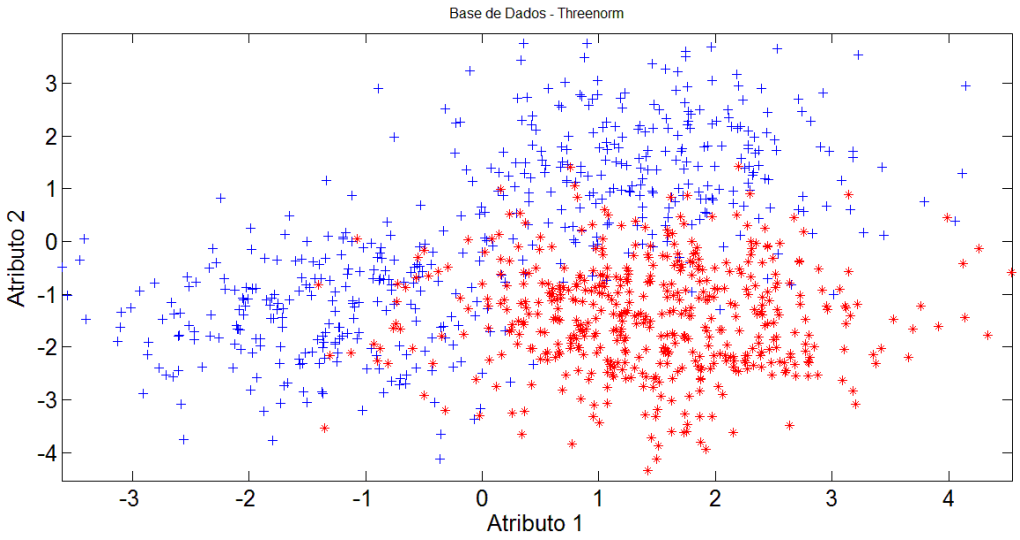

Figura 29: Base de dados - Threenorm

Diversos experimentos foram realizados para apoiar a determinação dos valores dos parâmetros citados anteriormente. Os experimentos foram feitos utilizando a técnica de validação cruzada com cinco partições. Os parâmetros da programação genética foram fixados inicialmente em:

Linhas=1; Colunas=400; *Levelback*=399; Tamanho da população=5; Quantidade de gerações=600; Partições=5; Taxa de mutação=0.1; Função de aptidão= Similaridade a partir do RMSE.

No primeiro experimento foi feita a variação da quantidade de operadores utilizados. Em um cenário são utilizados todos os operadores da APF original. Em um segundo cenário é utilizado um conjunto reduzido de operadores, este conjunto foi descrito no capítulo 4. Como medidas de desempenho foram utilizadas a média da taxa de acerto entre as classes e o AUC, que é a área sob a curva ROC.

Tabela 6: Experimento 1

	Todos os Operadores		Operadores escolhidos	
Bases de dados	Acerto Médio	AUC	Acerto Médio	AUC
Duas Espirais	0.60	0.69	0.67	0.76
Gaussiana	0.54	0.84	0.64	0.91
XOR	0.53	0.79	0.57	0.86
Difficult2	0.77	0.83	0.77	0.89
Difficult6	0.71	0.78	0.75	0.84
Difficult10	0.78	0.90	0.73	0.81
Difficult20	0.69	0.81	0.74	0.83
Threenorm	0.66	0.80	0.69	0.76

Nota-se que os resultados foram próximos. Isto pode ser confirmado pela aplicação do teste de Friedman (DERRAC et al., 2011) mostrando que não existe diferença estatisticamente significativa entre os resultados obtidos com o conjunto completo e com o conjunto reduzido de operadores. Sendo assim, optou-se por utilizar o conjunto reduzido de operadores, pois irá facilitar a compreensão da expressão representada pela árvore obtida. De certo modo pode-se dizer que o conjunto reduzido de operadores pode contribuir para uma melhora na interpretabilidade da solução.

No segundo experimento desenvolvido, foi feita a variação das partições *fuzzy*. Foram criados três cenários, com três, cinco e sete partições.

Tabela 7: Experimento 2 – Acerto e AUC

	3 Divisões		5 Divisões		7 Divisões	
Base de Dados	Acerto Médio	AUC	Acerto Médio	AUC	Acerto Médio	AUC
Duas Espirais	0.61	0.69	0.67	0.76	0.64	0.74
Gaussiana	0.74	0.93	0.64	0.91	0.62	0.88
XOR	0.53	0.86	0.57	0.86	0.55	0.89
Difficult	0.79	0.88	0.77	0.89	0.74	0.80
Difficult	0.69	0.83	0.75	0.84	0.65	0.72
Difficult	0.67	0.72	0.73	0.81	0.71	0.76
Difficult	0.72	0.77	0.74	0.83	0.62	0.66
Threenorm	0.68	0.78	0.69	0.76	0.67	0.73

Os resultados indicam um melhor desempenho utilizando cinco partições do que o obtido com três e sete partições. A tabela 8 apresenta o *rank* médio de cada cenário corroborando o melhor desempenho com cinco partições.

Tabela 8: Experimento 2 – Rank Médio

Rank Médio	3 Partições	5 Partições	7 Partições
Acerto Médio	2.1250	1.2500	2.6250
AUC	2.0625	1.4375	2.5000

Foi aplicado o teste de Friedman e o resultado não apresentou diferenças estatisticamente significantes, exceto entre os cenários de cinco e sete partições nas relações entre as taxas de acerto. A decisão do projeto foi feita em favor ao modelo com cinco partições. Pois além de ter o melhor desempenho, este cenário apresenta um equilíbrio entre uma quantidade grande de partições que iria garantir uma maior definição do universo de discurso e um cenário com poucas partições e consequentemente menos complexo.

No terceiro experimento foi feita a curva de aprendizagem de cada base de dados. A curva de aprendizagem mostra o erro de classificação e o erro de treinamento em relação a variação do tamanho do conjunto de treino. Para se obter resultados mais estáveis foram feitas trinta repetições para cada base de dados.

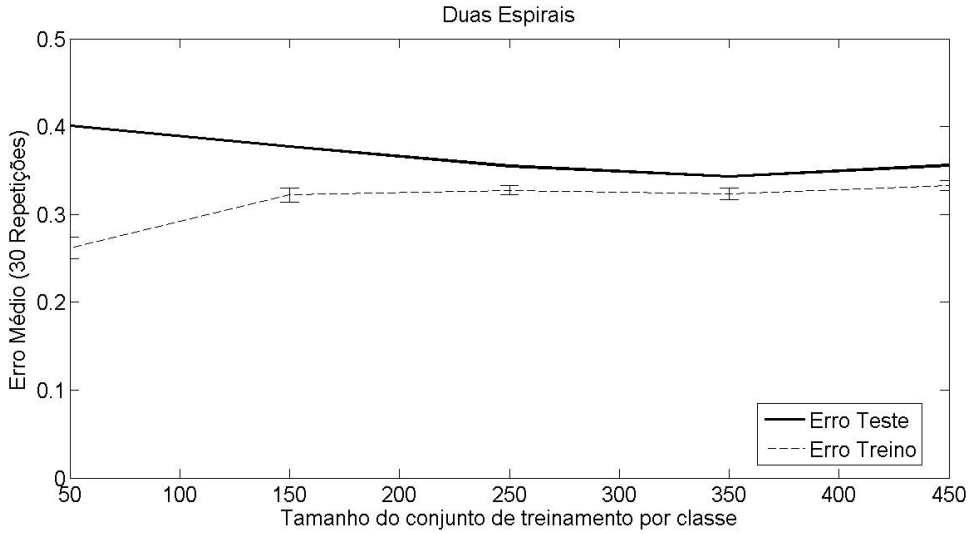

Figura 30: Curva de Aprendizado da base de dados Duas Espirais

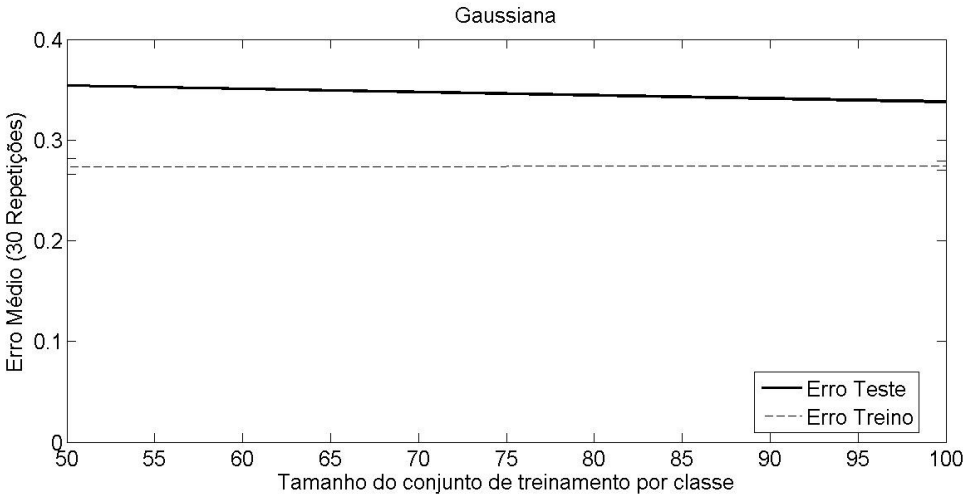

Figura 31: Curva de Aprendizado da base de dados Gaussiana

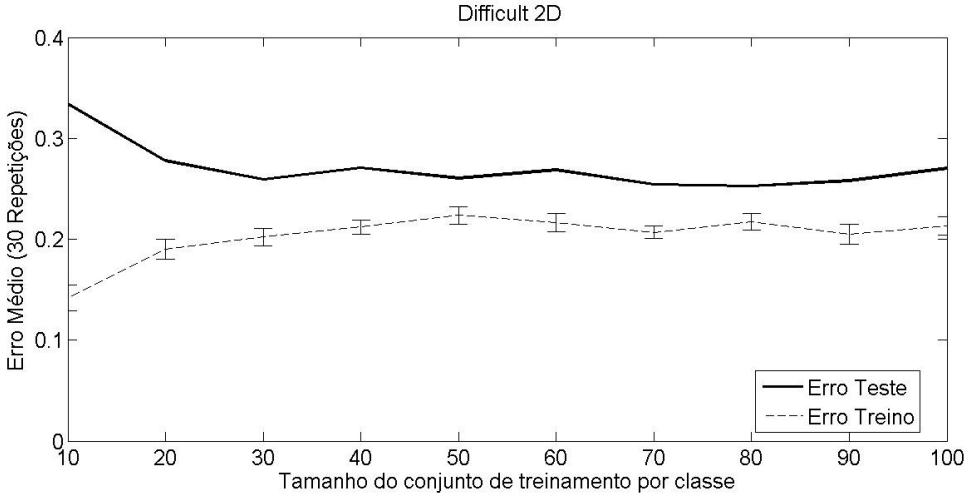

Figura 32: Curva de Aprendizado da base de dados Difficult 2D

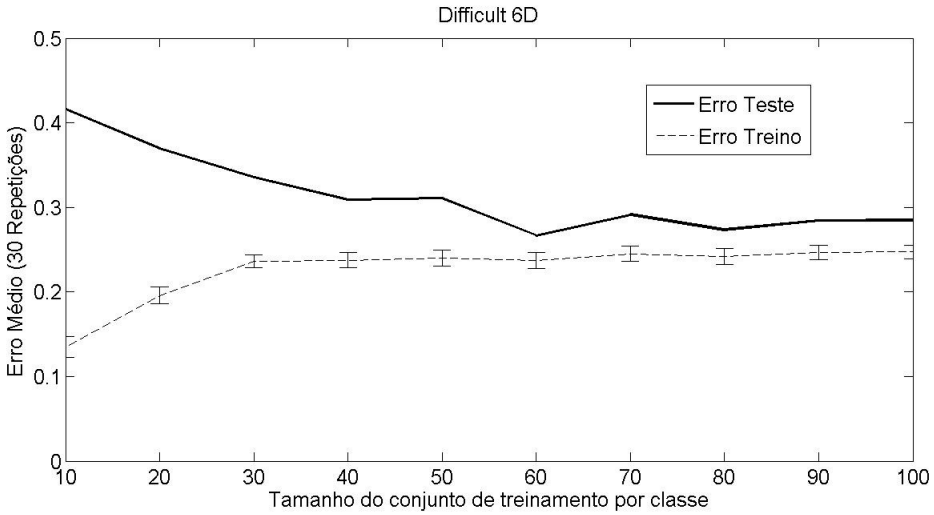

Figura 33: Curva de Aprendizado da base de dados Difficult 6D

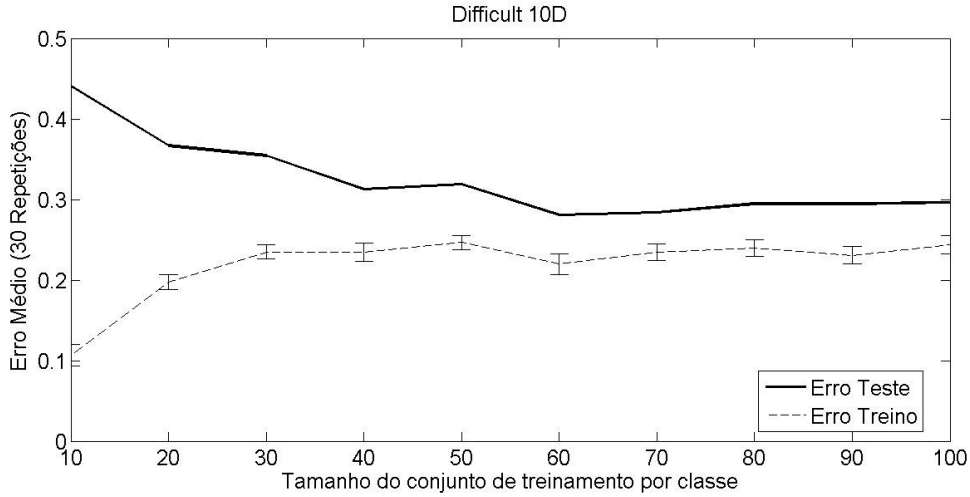

Figura 34: Curva de Aprendizado da base de dados Difficult 10D

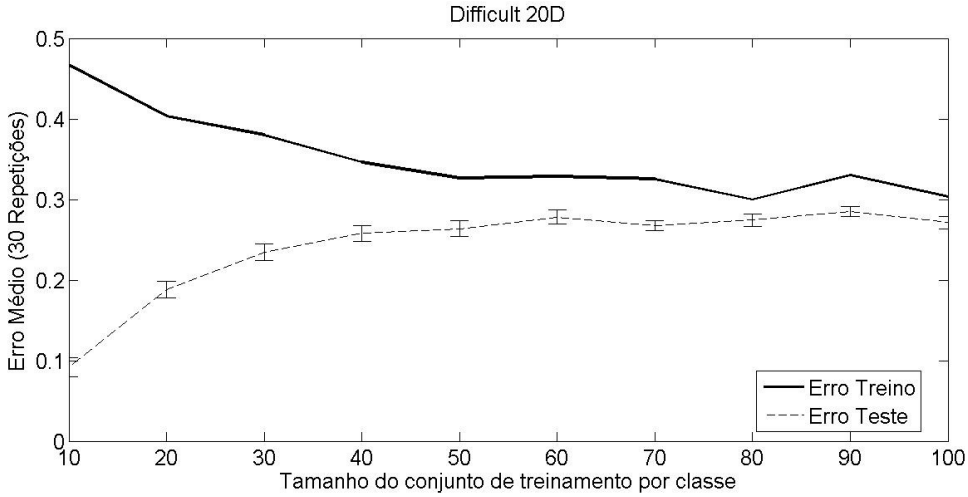

Figura 35: Curva de Aprendizado da base de dados Difficult 20D

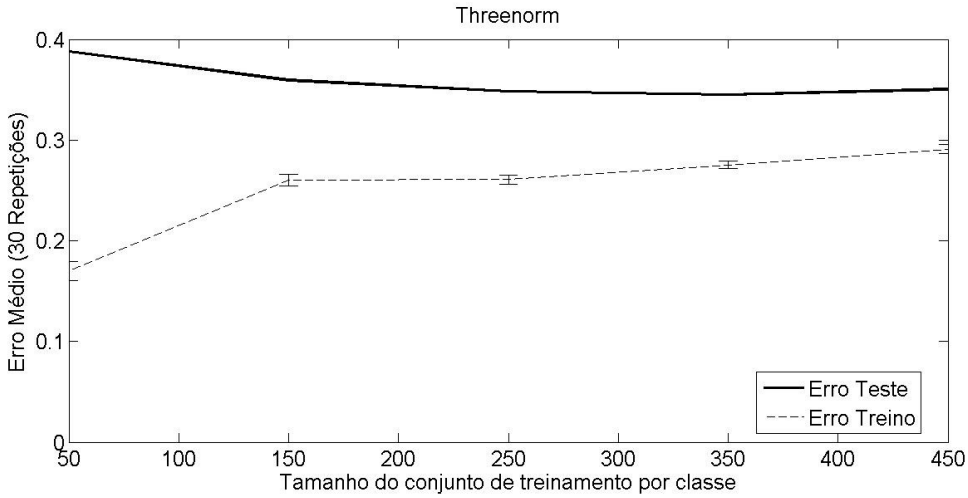

Figura 36: Curva de Aprendizado da base de dados Threenorm

A partir dos gráficos pode-se falar que os resultados obtidos foram os esperados. Nota-se uma maior variação do erro, quando o treinamento tem um menor número de pontos e então quando o classificador possui um número de pontos de treinamento maior, o erro se estabiliza. O fato do erro de treinamento estabilizar indica que não ocorre um treinamento excessivo, que prejudicaria a generalização do modelo.

O quarto experimento foi realizado para verificar a capacidade de a programação genética gerar árvores de decisão que possuíssem desempenho similar quando se utiliza os mesmos parâmetros e os mesmos conjuntos de treinamento e teste. Para que isso fosse possível foi utilizado o *Repeated Hold* out com dez repetições ao invés da

validação cruzada com cinco pastas. As figuras 37,38,39,40 mostram as curvas de evolução das classes de duas base de dados.

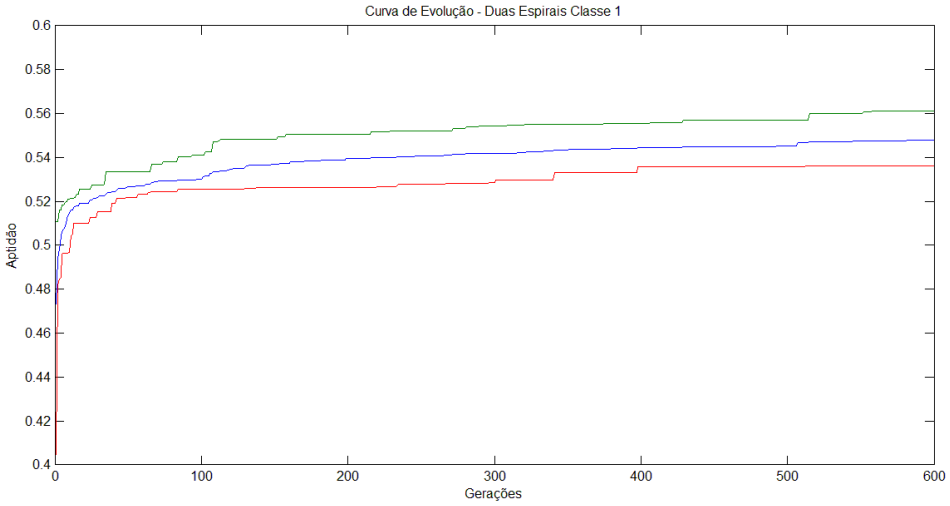

Figura 37: Curva de evolução da base de dados Duas Espirais, classe 1.

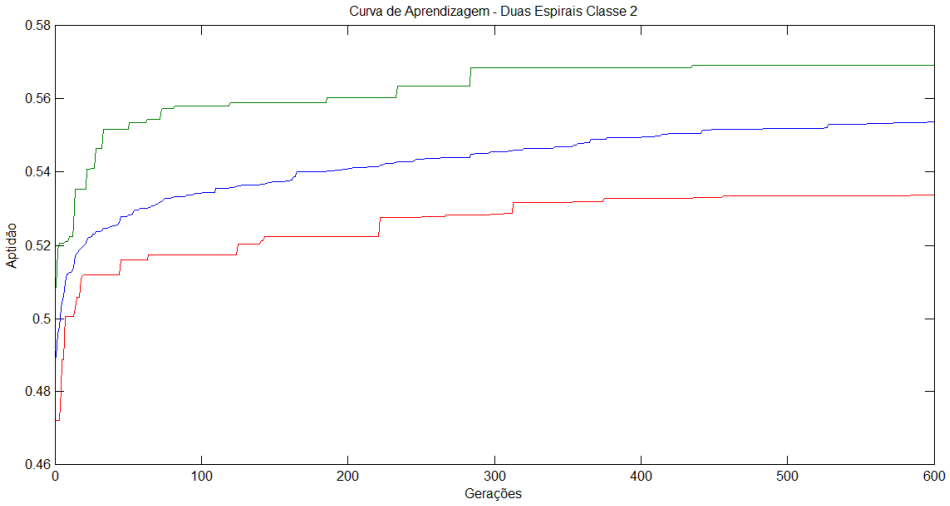

Figura 38: Curva de evolução da base de dados Duas Espirais, classe 2.

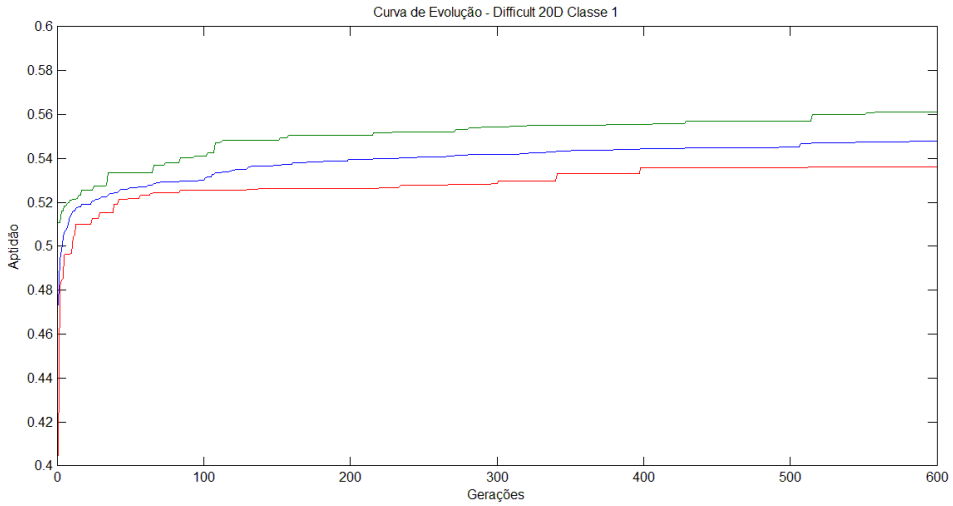

Figura 39: Curva de evolução da base de dados Difficult 20D, classe 1.

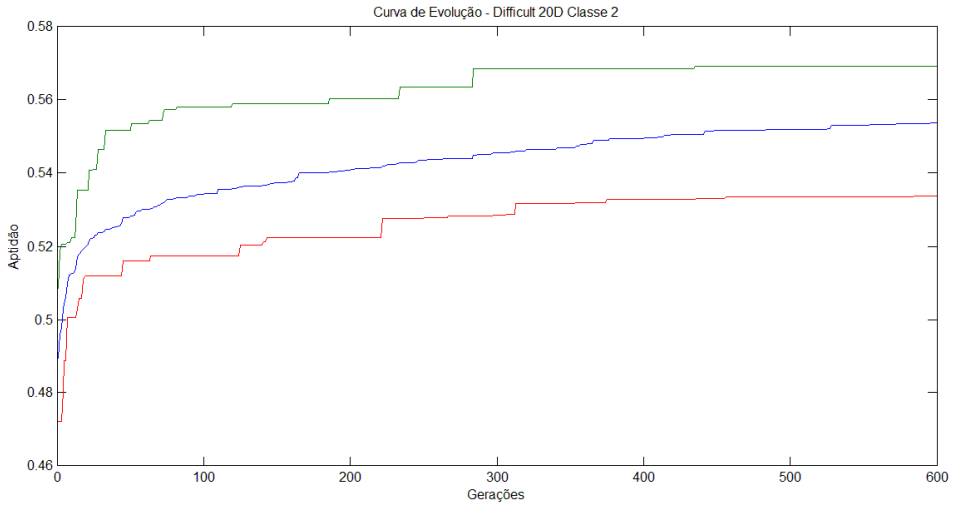

Figura 40: Curva de evolução da base de dados Difficult 20D, classe 2.

As curvas de evolução obtidas mostram o valor da aptidão médio, mínimo e máximo dentre as dez repetições feitas. Note-se que os valores estão próximos, denotando uma consistência do processo evolutivo.

A altura da árvore é um dos fatores que influem na complexidade da expressão representada na árvore. Sendo assim, no quinto experimento, avaliou-se o desempenho do sistema com a variação dos pesos das duas parcelas do critério de avaliação. O peso w_1 é relativo a medida de similaridade no treinamento e w_2 é relativo ao tamanho da árvore. O campo profundidade indica a profundidade média da árvore de cada classe, ou

seja, a distancia em níveis, entre a raiz da árvore a folha mais distante. O campo D apresenta os resultados de desempenho na forma x/y, onde x é o acerto médio e y é a área sob a curva ROC.

Tabela 9: Experimento 5 – Parte I

	w_1=1 w_2=0		w_1=0.7 w_2=0.3	
	Profundidade	D	Profundidade	D
Duas Espirais	[6.4 8.4]	0.67/0.76	[3.8 5.0]	0.65/0.71
Gaussiana	[5.8 6.4 7.0 7.0 8.0 7.2]	0.64/0.91	[2.8 3.0 2.2 2.8 2.8 3.0]	0.68/0.92
XOR	[6.2 7.6 6.0 7.8 6.6 7.0 7.2 6.0]	0.57/0.86	[4.2 3.8 3.0 3.0 3.2 3.4 3.4 2.8]	0.50/0.76
Difficult2D	[7.4 9.0]	0.77/0.89	[1.8 2.6]	0.70/0.83
Difficult6D	[7.0 6.8]	0.75/0.84	[2.8 1.6]	0.71/0.78
Difficult10D	[3.8 5.6]	0.73/0.81	[2.4 2.6]	0.77/0.83
Difficult20D	[4.2 5.2]	0.74/0.83	[2.4 2.0]	0.72/0.80
Threenorm	[4.6 3.2]	0.69/0.76	[1.8 2.2]	0.68/0.74

Tabela 10: Experimento 5 – Parte II

	w_1=0.5 w_2=0.5		w_1=0.3 w_2=0.7	
	Profundidade	D	Profundidade	D
Duas Espirais	[3.0 3.2]	0.65/0.68	[1.8 2.0]	0.63/0.64
Gaussiana	[2.4 3.2 2.4 1.6 2.6 2.4	0.67/0.90	[1.4 1.0 1.4 1.4 1.2 1.8]	0.59/0.87
XOR	[2.4 2.6 2.8 2.2 2.2 3.2 2.8 2.4]	0.40/0.71	[1.8 2.2 2.6 1.6 2.2 2.0 2.0 2.0]	0.35/0.68
Difficult2D	[2.2 3.8]	0.79/0.85	[1.8 1.0]	0.71/0.79
Difficult6D	[2.0 1.8]	0.75/0.81		0.70/0.82
Difficult10D	[1.4 1.8]	0.72/0.78		0.70/0.75
Difficult20D	[1.4 2.4]	0.71/0.79	[2.0 1.4]	0.69/0.85
Threenorm	[2.2 2.0]	0.66/0.72	[1.4 1.4]	0.64/0.70

Verifica-se que conforme se aumenta a limitação do tamanho da árvore o desempenho diminui. Implicando em um *trade-off* entre a acurácia (taxa de acerto) e interpretabilidade (árvores menores, mais simples). A partir dos resultados obtidos foram escolhidos os pesos w_1 =0,7 e w_2=0.3, por terem um bom resultado, e ainda limitarem o crescimento demasiado da árvore.

No sexto experimento foi feita a variação da função utilizada para definir a aptidão durante o treinamento. As funções utilizadas foram a “Similaridade a partir do RMSE” e a “AUC”.

Tabela 11: Experimento 6

	AUC		Similraridade a partir do RMSE	
Base de dados	Acerto Médio	AUC	Acerto Médio	AUC
Duas Espirais	0.61	0.76	0.67	0.77
Gaussiana	0.59	0.91	0.64	0.93
XOR	0.46	0.86	0.57	0.93
Difficult2D	0.77	0.89	0.77	0.92
Difficult6D	0.73	0.84	0.75	0.89
Difficult10D	0.65	0.81	0.73	0.89
Difficult20D	0.77	0.83	0.74	0.88
Threenorm	0.67	0.76	0.69	0.78

A partir dos resultados verifica-se que a função de medida do erro no treinamento que apresenta um melhor desempenho é a similaridade a partir do ‘RMSE’. Foi efetuado o teste de Friedman e foi observada uma diferença estatisticamente significativa entre as medidas. Por este motivo a similaridade a partir do RMSE foi definido como a medida padrão.

No sétimo experimento realizado, foi feita a variação da taxa de mutação, os resultados são apresentados na tabela 12.

Tabela 12: Experimento 7 - Parte I

	Taxa de mutação=0.3		Taxa de mutação=0.2	
	Acerto Médio	AUC	Acerto Médio	AUC
Duas Espirais	0.64	0.68	0.64	0.70
Gaussiana	0.77	0.81	0.77	0.87
XOR	0.41	0.70	0.45	0.74
Difficult	0.62	0.90	0.67	0.91
Difficult	0.71	0.79	0.69	0.78
Difficult	0.75	0.80	0.70	0.78
Difficult	0.72	0.79	0.72	0.78
Threenorm	0.66	0.73	0.66	0.74

Tabela 13: Experimento 7 – Parte II

	Taxa de mutação=0.1		Taxa de mutação=0.05	
	Acerto Médio	AUC	Acerto Médio	AUC
Duas Espirais	0.67	0.76	0.65	0.71
Gaussiana	0.77	0.89	0.80	0.89
XOR	0.57	0.86	0.57	0.79
Difficult	0.64	0.91	0.67	0.91
Difficult	0.75	0.84	0.74	0.81
Difficult	0.73	0.81	0.74	0.81
Difficult	0.74	0.83	0.68	0.76
Threenorm	0.69	0.76	0.66	0.73

Novamente foi feito o teste de Friedman, que determinou que há uma diferença estatisticamente significante entre os resultados e que de acordo com o rank médio dos resultados a taxa de mutação que apresenta o melhor desempenho é 0.1.

5.2 Algoritmos classificadores para comparação

Os algoritmos utilizados para comparações foram o Support Vector Machine Linear (SVM-L)(VAPNIK, 1999) o K-nearest Neighbors (KNN)(WEBB, 2002), o Random Forest (RF) (BREIMAN, 2001) e Support Vector Machine Radial (SVM-R) (VAPNIK, 1999). Os algoritmos escolhidos são conhecidos por terem um bom desempenho em um grande número de *benchmarks* (CARUANA; NICULESCU-MIZIL, 2006).

Máquinas de Vetores de Suporte (*Support vector machines* - SVM) é um sistema de aprendizado baseados na teoria do aprendizado estatístico (HASTIE; TIBSHIRANI; FRIEDMAN, 2011) e tem sido utilizado com sucesso em diversos problemas de classificação e regressão (BYUN; LEE, 2002), (SAPANKEVYCH; SANKAR, 2009). Para um problema de classificação com duas classes, a forma básica do SVM é o classificador linear. O SVM linear faz a classificação construindo um hiperplano que irá separar as classes de uma forma ótima. O hiperplano ótimo é o que cria uma margem máxima. A margem é definida como a distância de uma amostra do conjunto de treinamento e o hiperplano. Pode ser provado que esta solução particular tem a maior capacidade de generalização. Esta formulação pode ser generalizada pela

aplicação de um mapeamento não linear no conjunto de treino. Os dados são transpostos para um novo espaço com muitas dimensões, onde as classes são separadas mais facilmente e um hiperplano ótimo pode ser encontrado. É frequentemente usada para realizar este mapeamento não linear uma função de base radial, sendo frequentemente o primeiro mapeamento não linear a se considerar. Embora a superfície de decisão (hiperplano) seja linear no espaço com muitas dimensões, quando observado no espaço original ele não é mais linear, indicando que o SVM pode ser aplicado também para dados que não são linearmente separáveis. (GOLDBAUM et al., 2002).

O K Nearest Neighbour (KNN) é um dos algoritmos mais simples e mais elegantes de classificação em reconhecimento de padrões (KUNCHEVA, 2004). KNN propõe um aprendizado baseado em instâncias, ou aprendizado preguiçoso, significando que na fase de aprendizagem, ele simplesmente armazena um conjunto de instâncias rotuladas (conjunto de treinamento). Quando um novo ponto tem que ser classificado, o algoritmo encontra uma quantidade K de instâncias dentro do conjunto de treinamento perto do ponto que se deseja classificar, usando uma função de similaridade geralmente baseada na distância euclidiana. A classificação é realizada em favor da classe que é mais representada no conjunto dos K objetos mais próximos. Se K = 1, então o objeto é simplesmente atribuído à classe de seu vizinho mais próximo.

Florestas Aleatórias (*Random Forests* - RF) é um método de aprendizado que produz e combina diversas árvores de decisão (BREIMAN, 2001). Se usado para classificação, sua saída será a classe que possui um maior número de árvores de decisão apontando como correta, ao passo que se for utilizado para a regressão, apresenta a média dos resultados das árvores individuais. O algoritmo foi desenvolvido por Breiman(2001) apresentando dois aspectos chave: A ideia de *"bagging"* criada por Breiman e a seleção aleatória de características (HO, 1998), (AMIT; GEMAN, 1997). O

uso de *bagging* ajuda a reduzir a variância, fazendo a média de muitos modelos ruidosos, porém aproximadamente não tendenciosos. Árvores são candidatas ideais para o *bagging*, pois elas podem capturar nos dados, estruturas com interação complexa que tenham erro médio baixo. (HASTIE; TIBSHIRANI; FRIEDMAN, 2011).

A seleção de características aleatória auxilia a redução da variância do *bagging*, reduzindo a correlação entres as árvores. Isto é realizado no processo de crescimento da árvore, através da seleção aleatória das variáveis de entrada. Particularmente, no processo de crescimento de uma árvore individual em uma base de dados "*bootstrapped*", antes de cada divisão, um subconjunto de $m \leq p$ entre as p variáveis de entrada, é selecionado de forma aleatória para serem os candidatos a calcular a melhor divisão do conjunto de treinamento. O classificador RF é rápido e apresenta um desempenho no nível do estado da arte (CARUANA; NICULESCU-MIZIL, 2006). Ele pode lidar com uma grande quantidade de variáveis de entrada e oferece uma estimação interna do erro de generalização durante o processo de criação das árvores. Outra característica importante é a habilidade de ranquear a importância das variáveis, aparentemente para medir a força de predição de cada variável. Também podendo computar proximidade entre os objetos, o que é útil para formação de *cluster*, detectar limites e visualizar os dados (pela escala).

5.3 Conjuntos de dados

Foram utilizados para testar e comparar o algoritmo diversas bases de dados que estão disponíveis no UCI *machine learning repositor*, no STATLIB e no LIBSVM. A tabela 14 apresenta todas as bases de dados utilizadas e suas respectivas quantidades de pontos, atributos e classes.

Tabela 14: Bases de dados Utilizadas

Base de Dados	Pontos	Atributos	Classes
Iris	150	4	3
Wine	178	13	3
Sonar	208	60	2
Pima	768	8	2
Balance	625	4	3
Habeman	306	3	2
Lupus	87	3	2
Breast_Cancer	683	9	2
Australian	690	14	2
Analcatdata-lawsuit	264	4	2
Ionosphere	351	33	2
Bupa	345	6	2
Transfusion	748	4	2

5.4 Resultados obtidos

Nesta seção serão apresentados alguns estudos experimentais realizados para se obter uma ideia do desempenho do modelo criado, comparado com os outros classificadores citados anteriormente.

A estimativa do erro de generalização foi feita através de uma validação cruzada com cinco partições e cinco repetições.

Foram utilizadas duas medidas de desempenho, a taxa de acertos de classificação e a área sob a curva ROC (FAWCETT, 2006). Para os casos onde a base de dados possui mais de uma classe, a AUC fornecida é dada pela média das AUCs calculadas na modalidade “um contra todos”.

Os parâmetros dos outros classificadores foram obtidos da seguinte forma: O número de vizinhos K foi fixado em 1; o número de árvores geradas no classificador RF foi igual a 50 e o número de atributos sob o qual é feita a partição foi igual a 1. Os parâmetros de regularização do SVM de base linear e de base radial foram determinados a partir de busca pelo melhor desempenho em uma validação cruzada interna. Para cada partição, foi feita uma validação cruzada que utilizava apenas o conjunto de treinamento da respectiva partição. Este mesmo método foi utilizado para encontrar o valor do parâmetro *r* que define a base radial. Esta forma de encontrar os parâmetros é baseada

na validação cruzada aninhada (CAWLEY; TALBOT, 2010) é tida como uma forma de evitar o *overfit*. As tabelas 15 e 16 apresentam o desempenho dos algoritmos com relação a taxa de acerto e o AUC respectivamente. A tabela 17 apresenta o *rank* médio obtido por cada algoritmo.

Tabela 15: Taxa de acerto

	SVM-L	KNN	RF	SVM-R	APF1	APF2	APF-nout
Iris	0.96	0.94	0.95	0.95	0.96	0.96	0.93
Wine	0.98	0.95	0.98	0.98	0.91	0.93	0.82
Sonar	0.80	0.87	0.88	0.91	0.77	0.77	0.67
Pima	0.77	0.70	0.77	0.71	0.73	0.75	0.62
Balance	0.87	0.77	0.85	0.95	0.83	0.84	0.31
Habeman	0.72	0.67	0.65	0.71	0.75	0.76	0.70
Breast_Cancer	0.97	0.95	0.96	0.96	0.96	0.96	0.96
Australian	0.86	0.80	0.79	0.85	0.85	0.84	0.86
Ionosphere	0.87	0.86	0.90	0.92	0.88	0.90	0.71
Lupus	0.74	0.68	0.62	0.73	0.73	0.75	0.71
Bupa	0.67	0.61	0.66	0.67	0.686	0.65	0.44
Transfusion	0.76	0.71	0.7	0.73	0.77	0.77	0.65
Analcatdata-Lawsuit	0.99	0.96	0.97	0.96	0.96	96	0.80

Tabela 16: AUC

	SVM-L	KNN	RF	SVM-R	APF1	APF2	APF-nout
Iris	0.93	0.98	0.99	0.99	0.99	1.00	0.99
Wine	1.00	0.98	1.00	1.00	0.99	0.99	0.97
Sonar	0.89	0.95	0.94	0.98	0.85	0.84	0.79
Pima	0.83	0.72	0.82	0.79	0.79	0.77	0.68
Balance	0.93	0.88	0.95	1.00	0.91	0.94	0.55
Habeman	0.70	0.62	0.57	0.63	0.67	0.75	0.56
Breast_Cancer	0.99	0.99	0.99	0.98	0.99	0.99	0.99
Australian	0.93	0.86	0.92	0.91	0.91	0.89	0.87
Ionosphere	0.87	0.97	0.97	0.98	0.92	0.90	0.75
Lupus	0.86	0.67	0.72	0.78	0.84	0.89	0.81
Bupa	0.71	0.64	0.72	0.73	0.68	0.73	0.52
Transfusion	0.74	0.60	0.64	0.66	0.72	0.70	0.59
Analcatdata-Lawsuit	1.00	0.97	0.99	0.99	0.99	0.99	0.83

Tabela 17: Rank Médio

Rank Médio	SVM-L	KNN	RF	SVM-R	APF1	APF2	APF-nout
Taxa de Acerto	2.42	5.53	4.10	3.15	3.61	3.10	6.00
AUC	2.96	5.30	3.38	3.11	3.76	3.30	6.15

Três variantes da APF foram utilizadas nos testes, a variante APF1, possui genótipos com 400 nós evoluídos em 600 gerações. A variante APF2 possui genótipos com 50 nós evoluídos em 1250 gerações. A terceira variante APF-nout trata-se do algoritmo com apenas um genótipo com n saídas, uma saída para cada classe. Foi Realizado o teste de Friedman (DERRAC et al., 2011) nos resultados apresentados nas tabelas 15 e 16. Neste teste foi determinado o valor de α=0.05. O teste demonstrou que não há diferença estatisticamente significativa entre os algoritmos, exceto pelo algoritmo APF-nout, que apresentou um desempenho inferior aos demais, cuja possível causa seja a interferência das mutações que são benéficas ao processo de classificação de uma classe, mas maléficas a outra, retardando assim o processo evolutivo.

Os dois algoritmos APF1 e APF2 obtiveram um bom desempenho, se aproximando dos outros classificadores, mas com a vantagem de proporcionar um modelo que pode ser interpretado. A tabela 17 apresentou o *rank* médio entre os resultados obtidos para todas as bases de dados. O que deixa mais claro o fato que a APF tem um desempenho compatível com estes outros bons algoritmos.

Também foi comparado o modelo proposto, com as versões da APF apresentadas anteriormente em (SENGE; HÜLLERMEIER, 2011) e em (HUANG; GEDEON; NIKRAVESH, 2008). Da estratégia Top-Down criada por Senge e Hüllermeier foi utilizada para a comparação o PTTDE.25, que possui como parâmetros: *beam*=5 e Parada Adaptativa=0.25%. Da estratégia Bottom-Up criada por Huang foi utilizada para comparação o PT10, que possui como parâmetros: tamanho das árvores candidatas=2 e profundidade máxima da árvore=10. As características comparadas foram a taxa de acerto, a profundidade média das árvores e a quantidades de avaliações efetuadas pelo algoritmo. A tabela 18 apresenta a taxa de acerto para cada algoritmo e a tabela 19 o *rank* médio.

Tabela 18: Taxa de Acerto

	APF1	APF2	PTTDE.25	PT10
Iris	0.96	0.96	0.95	0.95
Wine	0.91	0.93	0.98	0.94
Sonar	0.77	0.77	0.80	0.69
Pima	0.73	0.75	0.76	0.75
Balance	0.83	0.84	0.87	0.85
Habeman	0.75	0.76	0.74	0.74
Breast_Cancer	0.96	0.96	0.96	0.96
Australian	0.85	0.84	0.85	0.85
Ionosphere	0.88	0.90	0.91	0.88
Lupus	0.73	0.75	0.77	0.77
Bupa	0.60	0.65	0.70	0.69
Transfusion	0.77	0.77	0.77	0.77
Analcatdata-Lawsuit	0.96	96	0.94	0.95

Tabela 19: Rank Médio

Rank Médio	APF1	APF2	PTTDE.25	PT10
Taxa de Acerto	2.96	2.42	1.96	2.65

Foi aplicado o teste de Friedman nos resultados apresentados na tabela 18 e o resultado do teste foi que não há diferença estatisticamente significativa entre os algoritmos. O *rank* médio dá uma noção de que não há uma grande diferença entre os algoritmos.

A segunda comparação foi feita com relação a profundidade média das árvores. Esta comparação foi feita entre o APF2 e o PTTDE.25, que foram os algoritmos com os melhores resultados na comparação anterior. Esta comparação foi feita para o pior caso do APF2, onde o peso da avaliação que favorece árvores menores foi zerado.

Tabela 20: Profundidade Média das Árvores

	APF2	PTTDE.25
Iris	2.6	3.33
Wine	2.33	10
Sonar	1.5	10
Pima	2	4
Balance	4.3	1.66
Habeman	6	2
Breast_Cancer	2.75	5
Australian	2	5
Ionosphere	2	14
Lupus	4.75	4
Bupa	4.25	10
Transfusion	4.75	5
Analcatdata-Lawsuit	4.5	9

A partir dos resultados apresentados na tabela 20, pode-se concluir que o APF2 possui um desempenho melhor que o PTTDE.25 com relação a profundidade média das árvores, gerando árvores com uma menor profundidade. Isso se dá possivelmente devido ao fato de que o algoritmo de busca do tipo *beam-search* utilizado na estratégia Top-Down do PTTDE.25 ter uma característica "gulosa", escolhendo sempre os melhores candidatos, sem voltar atrás depois de escolhido, ignorando candidatos que podem não ter um bom desempenho individualmente mas teriam um desempenho melhor em um combinação com um sub-árvore escolhida posteriormente. Observa-se também que quanto maior for a quantidade de atributos na base de dados, maior será a diferença entre as profundidades médias.

A terceira comparação feita foi a quantidade de avaliações necessárias para se chegar aos resultados vistos anteriormente. Foram comparados o PTTDE.25, APF1 e o APF2. Para o APF1 e APF2, foi feito o calculo da quantidade de avaliação para o pior caso possível, que seria se não houver parada antecipada. Os resultados serão apresentados na tabela 21.

Tabela 21: Quantidade de Avaliações

	FPT	FPT2	PTTDE.25
Iris	7.2k	15k	33k
Wine	7.2k	15k	346.5k
Sonar	4.8k	10k	990k
Pima	4.8k	10k	24k
Balance	7.2k	15k	3.9k
Habeman	4.8k	10k	2.7k
Breast_Cancer	4.8k	10k	22.5k
Australian	4.8k	10k	63k
Lawsuit	4.8k	10k	54k
Ionosphere	4.8k	10k	519k
Bupa	4.8k	10k	99k
Transfusion	4.8k	10k	18k
Lupus	4.8k	10k	12k

Nesta comparação nota-se que o PTTDE.25 necessita de mais avaliações em média para chegar aos resultados vistos anteriormente em comparação com o APF1 e APF2. Esta característica se acentua o quanto maior for a quantidade de atributos na base de dados, devido a uma explosão no número de possibilidades na estratégia Top-Down, o que não ocorre para o modelo proposto.

CONCLUSÃO

A Incapacidade humana de analisar a grande quantidade de informação que está sendo gerada nos dias de hoje despertou um grande volume de interesse nas áreas de descoberta de conhecimento, mineração de dados e aprendizado de máquinas. Tem-se particular interesse por conseguir adquirir e representar o conhecimento de uma forma linguística, que pode ser mais facilmente compreendida por seres humanos. Neste aspecto, a Teoria dos Conjuntos Fuzzy, em especial os Sistemas Fuzzy Baseados em Regras (SFBR) trazem grandes contribuições ao fornecer uma interface entre padrões quantitativos e estruturas de conhecimento qualitativas expressas em termos de linguagem natural.

A obtenção de um SFBR pode não ser uma tarefa simples. A "maldição da dimensionalidade" e o compromisso entre acurácia e interpretabilidade se constituem em grandes desafios. Para a solução destes problemas, diversas soluções foram propostas como a utilização de sistemas fuzzy hierárquicos (WANG; ZENG; KEANE, 2006),(SOUZA; VELLASCO; PACHECO, 2002),(GONCALVES et al., 2006), (HUANG; GEDEON; NIKRAVESH, 2008), (SENGE; HÜLLERMEIER, 2011).

Este trabalho apresenta um método para indução de forma automática de um modelo fuzzy hierárquico chamado de Árvores de Padrões Fuzzy (APF) utilizado na tarefa de classificação. O método de aprendizado do APF foi substituído e em seu lugar foi utilizada a Programação Genética Cartesiana (PGC). A PGC é um método de busca global capaz de explorar espaços de busca bastante grandes de forma eficiente e a representação dos programas na forma de grafos pode ser facilmente utilizada para representar APFs.

Foram realizados diversos estudos de casos para obter uma melhor compreensão do funcionamento do método, desenvolver estratégias para obter uma melhor generalização e avaliar o desempenho dos classificadores fuzzy gerados.

Nos estudos de casos observou-se que o método proposto consegue gerar classificadores com um desempenho competitivo em relação a classificadores bastante populares por seu bom desempenho em um grande número de aplicações como as máquinas de vetores suporte e florestas aleatórias.

Também se verificou que o método proposto apresenta desempenho similar aos algoritmos propostos por (SENGE; HÜLLERMEIER, 2011) e (HUANG; GEDEON; NIKRAVESH, 2008), porem com um número de avaliações menor e com a geração de árvores mais compactas. Isto se deve ao fato que o modelo proposto não sofre com a "maldição da dimensionalidade" e faz a busca de uma forma global.

As propostas para trabalhos futuros no desenvolvimento desta pesquisa estão listadas a seguir:

- Extensão do modelo para tarefas de previsão e controle;
- Incorporação de modificadores linguísticos: conforme descrito em (CASILLAS et al., 2005) e (KOSHIYAMA, 2014), o uso de modificadores linguísticos pode ser usado como uma forma de ajustar as funções de pertinência e ainda manter a interpretabilidade. Esta característica pode ser incorporada no método proposto utilizando os modificadores em conjuntos com os termos fuzzy na etapa de fuzzificação ou disponibilizando o modificador como um operador adicional;
- Aperfeiçoamento do algoritmo multiobjectivo: A versão atual utiliza uma forma rudimentar para o tratamento de multiobjectivos. Ele pode ser aperfeiçoado pela implementação do ranqueamento de soluções baseado no conceito de dominância de Pareto (ZHOU et al., 2011);
- Estudo de sensibilidade para gerar dados artificiais: atualmente o conjunto de dados artificiais que pode ser gerador para validação utiliza algoritmos para estimar a função densidade de probabilidade que gerou os dados utilizando Janelas de Parzen ou K Vizinhos mais próximos. Pode ser útil investigar outros algoritmos que possam estimar estas funções;
- Incorporação de algumas ideias de florestas aleatórias: Atualmente sintetiza-se apenas uma árvore por classe. É possível sintetizar mais de uma árvore por classe e combinar estes resultados de forma semelhante ao realizado em florestas aleatórias. Além disso, poderia ser incorporar o "*bagging*" que é outra ideia presente em florestas aleatórias. O uso de *bagging* ajuda a reduzir a variância, fazendo a média de muitos modelos ruidosos;
- Utilização de um subsistema *fuzzy* como operador: Além de utilizar os operadores descritos, também é possível utilizar um subsistema *fuzzy*. Ele pode ter a forma de um sistema baseado em regras com duas entradas e uma saída

conforme apresentado em (ZHANG; ZHANG, 2006), ou como na forma de células Quadtree ou BSP (SOUZA; VELLASCO; PACHECO, 2002).

REFERÊNCIAS

A. GARMENDIA-DOVAL, J. M. Cartesian Genetic Programming and the Post Docking Filtering Problem. p. 225–244, 2006.

ABRAHAM, A. Adaptation of Fuzzy Inference System Using Neural Learning. In: NEDJAH, N.; MOURELLE, L. DE M. (Eds.). **Fuzzy Systems Engineering**. Studies in Fuzziness and Soft Computing. [s.l.] Springer Berlin Heidelberg, 2005. p. 53–83.

AFZAL, W.; TORKAR, R. On the application of genetic programming for software engineering predictive modeling: A systematic review. **Expert Systems with Applications**, v. 38, n. 9, p. 11984–11997, 2011.

ALCALÁ, R. et al. **A Multi-Objective Genetic Algorithm for Tuning and Rule Selection to Obtain Accurate and Compact Linguistic Fuzzy Rule-Based Systems**.

ALCALÁ-FDEZ, J.; HERRERA, F. Increasing fuzzy rules cooperation based on evolutionary adaptive inference systems. **Int. J. Intell. Syst.**, v. 22, p. 1035–1064, 2007.

AMIT, Y.; GEMAN, D. Shape Quantization and Recognition with Randomized Trees. **Neural Comput.**, v. 9, n. 7, p. 1545–1588, out. 1997.

BÄCK, T.; SCHWEFEL, H.-P. An Overview of Evolutionary Algorithms for Parameter Optimization. **Evol. Comput.**, v. 1, n. 1, p. 1–23, mar. 1993.

BANZHAF, W. **Genotype-Phenotype-Mapping and Neutral Variation - A Case Study in Genetic Programming**Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature. **Anais**...: PPSN III.London, UK, UK: Springer-Verlag, 1994.

BASTIAN, A. Identifying fuzzy models utilizing genetic programming. **Fuzzy sets and systems**, v. 113, n. 3, p. 333–350, 2000.

BERLANGA, F. et al. **Multiobjective Evolutionary Induction of Subgroup Discovery Fuzzy Rules: A Case Study in Marketing**Proceedings of the 6th Industrial Conference on Data Mining Conference on Advances in Data Mining: Applications in Medicine, Web Mining, Marketing, Image and Signal Mining. **Anais**...: ICDM'06.Berlin, Heidelberg: Springer-Verlag, 2006.

BERNARD, J. A. Use of a rule-based system for process control. **Control Systems Magazine, IEEE**, v. 8, n. 5, p. 3–13, 1988.

BOTTA, A. et al. **Exploiting Fuzzy Ordering Relations to Preserve Interpretability in Context Adaptation of Fuzzy Systems**Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International. **Anais**... In: FUZZY SYSTEMS CONFERENCE, 2007. FUZZ-IEEE 2007. IEEE INTERNATIONAL. jul. 2007

BOTTA, A.; LAZZERINI, B.; MARCELLONI, F. **Context Adaptation of Mamdani Fuzzy Systems through New Operators Tuned by a Genetic Algorithm**2006 IEEE

International Conference on Fuzzy Systems. **Anais**... In: 2006 IEEE INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS. 2006

BREIMAN, L. **[Bias, Variance, and] Arcing Classifiers**. University of California at Berkeley, Berkeley, California: Statistics Department, University of California, Berkeley, fev. 1996. Disponível em: <http://stat-reports.lib.berkeley.edu/accessPages/460.html>.

BREIMAN, L. Random Forests. **Machine Learning**, v. 45, n. 1, p. 5–32, 1 out. 2001.

BUCKLEY, J. J. Universal fuzzy controllers. **Automatica**, v. 28, n. 6, p. 1245–1248, nov. 1992.

BYUN, H.; LEE, S.-W. **Applications of Support Vector Machines for Pattern Recognition: A Survey**Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines. **Anais**...: SVM '02.London, UK, UK: Springer-Verlag, 2002Disponível em: <http://dl.acm.org/citation.cfm?id=647230.719394>.

CARUANA, R.; NICULESCU-MIZIL, A. **An Empirical Comparison of Supervised Learning Algorithms**Proceedings of the 23rd International Conference on Machine Learning. **Anais**...: ICML '06.New York, NY, USA: ACM, 2006Disponível em: <http://doi.acm.org/10.1145/1143844.1143865>.

CASILLAS, J. **Accuracy Improvements in Linguistic Fuzzy Modeling**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

CASILLAS, J. et al. Genetic tuning of fuzzy rule deep structures preserving interpretability and its interaction with fuzzy rule set reduction. **IEEE Transactions on Fuzzy Systems**, v. 13, n. 1, p. 13–29, fev. 2005.

CASILLAS, J.; MARTINEZ, P. **Consistent, Complete and Compact Generation of DNF-type Fuzzy Rules by a Pittsburgh-style Genetic Algorithm**Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International. **Anais**... In: FUZZY SYSTEMS CONFERENCE, 2007. FUZZ-IEEE 2007. IEEE INTERNATIONAL. jul. 2007

CAWLEY, G. C.; TALBOT, N. L. C. On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. **J. Mach. Learn. Res.**, v. 11, p. 2079–2107, ago. 2010.

CLIFTON, C. **data mining (computer science) -- Britannica Online Encyclopedia**. Disponível em: <http://www.britannica.com/EBchecked/topic/1056150/data-mining>.

COCOCCIONI, M. et al. A Pareto-based Multi-objective Evolutionary Approach to the Identification of Mamdani Fuzzy Systems. **Soft Comput.**, v. 11, n. 11, p. 1013–1031, maio 2007.

CORDÓN, O. **Genetic Fuzzy Systems: Evolutionary Tuning And Learning Of Fuzzy Knowledge Bases**. Singapore: Wspc, 2002.

CORDÓN, O. et al. Ten years of genetic fuzzy systems: current framework and new trends. **Fuzzy Sets and Systems**, Genetic Fuzzy Systems: New Developments. v. 141, n. 1, p. 5–31, 1 jan. 2004.

CORDÓN, O. A Historical Review of Evolutionary Learning Methods for Mamdani-type Fuzzy Rule-based Systems: Designing Interpretable Genetic Fuzzy Systems. **Int. J. Approx. Reasoning**, v. 52, n. 6, p. 894–913, set. 2011.

CORDÓN, O.; DEL JESUS, M. J.; HERRERA, F. A proposal on reasoning methods in fuzzy rule-based classification systems. **International Journal of Approximate Reasoning**, v. 20, n. 1, p. 21–45, jan. 1999.

CORDON, O.; HERRERA, F.; VILLAR, P. Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base. **IEEE Transactions on Fuzzy Systems**, v. 9, n. 4, p. 667–674, ago. 2001.

DERRAC, J. et al. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. **Swarm and Evolutionary Computation**, v. 1, n. 1, p. 3–18, mar. 2011.

DHARWADKER, A.; PIRZADA, S. **Applications of Graph Theory**. [s.l.] CreateSpace Independent Publishing Platform, 2011.

DIAS, D. M. Programação Genética Linear com Inspiração Quântica. 2010.

DUDA, R. O.; HART, P. E.; STORK, D. G. **Pattern Classification**. [s.l.] John Wiley & Sons, 2012.

DUIN, R. P. W. A Note on Comparing Classifiers. **Pattern Recogn. Lett.**, v. 17, n. 5, p. 529–536, maio 1996.

ESPEJO, P. G.; VENTURA, S.; HERRERA, F. A Survey on the Application of Genetic Programming to Classification. **IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews**, v. 40, n. 2, p. 121–144, mar. 2010.

FANTUZZI, C.; ROVATTI, R. **On the approximation capabilities of the homogeneous Takagi-Sugeno model**, Proceedings of the Fifth IEEE International Conference on Fuzzy Systems, 1996. **Anais**... In: , PROCEEDINGS OF THE FIFTH IEEE INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS, 1996. set. 1996

FAWCETT, T. An Introduction to ROC Analysis. **Pattern Recogn. Lett.**, v. 27, n. 8, p. 861–874, jun. 2006.

FAYYAD, U. M.; PIATETSKY-SHAPIRO, G.; SMYTH, P. Advances in Knowledge Discovery and Data Mining. In: FAYYAD, U. M. et al. (Eds.). Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996. p. 1–34.

FERRI, C.; HERNÁNDEZ-ORALLO, J.; MODROIU, R. An experimental comparison of performance measures for classification. **Pattern Recognition Letters**, v. 30, n. 1, p. 27–38, 1 jan. 2009.

FOGEL, D. B. **Evolutionary Computation: Toward a New Philosophy of Machine Intelligence**. Piscataway, NJ, USA: IEEE Press, 1995.

FORREST, S. et al. **A Genetic Programming Approach to Automated Software Repair**Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation. **Anais**...: GECCO '09.New York, NY, USA: ACM, 2009Disponível em: <http://doi.acm.org/10.1145/1569901.1570031>.

FREITAS, A. A. A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery. In: GHOSH, D. A.; TSUTSUI, P. D. S. (Eds.). **Advances in Evolutionary Computing**. Natural Computing Series. [s.l.] Springer Berlin Heidelberg, 2003. p. 819–845.

GACTO, M. J.; ALCALÁ, R.; HERRERA, F. Interpretability of Linguistic Fuzzy Rule-based Systems: An Overview of Interpretability Measures. **Inf. Sci.**, v. 181, n. 20, p. 4340–4360, out. 2011.

GEYER-SCHULZ, A. **Fuzzy rule-based expert systems and genetic machine learning**. [s.l.] Physica-Verlag, 1997.

GIORDANA, A.; NERI, F. Search-intensive Concept Induction. **Evol. Comput.**, v. 3, n. 4, p. 375–416, dez. 1995.

GOLDBAUM, M. H. et al. Comparing machine learning classifiers for diagnosing glaucoma from standard automated perimetry. **Investigative Ophthalmology & Visual Science**, v. 43, n. 1, p. 162–169, jan. 2002.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

GOMIDE, F.; GUDWIN, R.; TANSCHEIT, R. **Conceitos fundamentais da teoria de conjuntos fuzzy, lógica fuzzy e aplicações**. In: PROC. 6 TH IFSA CONGRESS-TUTORIALS. 1995

GONCALVES, L. B. et al. Inverted Hierarchical Neuro-fuzzy BSP System: A Novel Neuro-fuzzy Model for Pattern Classification and Rule Extraction in Databases. **Trans. Sys. Man Cyber Part C**, v. 36, n. 2, p. 236–248, mar. 2006.

GREENE, D. P.; SMITH, S. F. Competition-Based Induction of Decision Models from Examples. **Machine Learning**, v. 13, n. 2-3, p. 229–257, 1 nov. 1993.

GUDWIN, R.; GOMIDE, F.; PEDRYCZ, W. Context Adaptation in Fuzzy Processing and Genetic Algorithms. **INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS**, v. 13, p. 948, 1998.

GUILLAUME, S. Designing Fuzzy Inference Systems from Data: An Interpretability-oriented Review. **Trans. Fuz Sys.**, v. 9, n. 3, p. 426–443, jun. 2001.

GUYON, I.; ELISSEEFF, A. An Introduction to Variable and Feature Selection. **J. Mach. Learn. Res.**, v. 3, p. 1157–1182, mar. 2003.

HARRIES, K.; SMITH, P. Code Growth, Explicitly Defined Introns and Alternative Selection Schemes. **Evolutionary Computation**, v. 6, p. 339–360, 1998.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition**. 2nd ed. 2009. Corr. 7th printing 2013 edition ed. New York, NY: Springer, 2011.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. 2nd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

HERRERA, F. Genetic fuzzy systems: taxonomy, current research trends and prospects. **Evolutionary Intelligence**, v. 1, n. 1, p. 27–46, 1 mar. 2008.

HERRERA, F.; MAGDALENA, L. **Genetic Fuzzy Systems: A Tutorial**. [s.l: s.n.].

HO, T. K. The Random Subspace Method for Constructing Decision Forests. **IEEE Trans. Pattern Anal. Mach. Intell.**, v. 20, n. 8, p. 832–844, ago. 1998.

HOMAIFAR, A.; MCCORMICK, E. Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. **IEEE Transactions on Fuzzy Systems**, v. 3, n. 2, p. 129–139, maio 1995.

HONG, T.-P. et al. A GA-based Fuzzy Mining Approach to Achieve a Trade-off Between Number of Rules and Suitability of Membership Functions. **Soft Comput.**, v. 10, n. 11, p. 1091–1101, jun. 2006.

HUANG, Z.; GEDEON, T. D.; NIKRAVESH, M. Pattern Trees Induction: A New Machine Learning Method. **IEEE Transactions on Fuzzy Systems**, v. 16, n. 4, p. 958–970, ago. 2008.

HÜLLERMEIER, E. Fuzzy Methods in Machine Learning and Data Mining: Status and Prospects. **Fuzzy Sets Syst.**, v. 156, n. 3, p. 387–406, dez. 2005.

ISHIBUCHI, H.; MURATA, T.; TÜRKŞEN, I. B. Single-objective and Two-objective Genetic Algorithms for Selecting Linguistic Rules for Pattern Classification Problems. **Fuzzy Sets Syst.**, v. 89, n. 2, p. 135–150, jul. 1997.

ISHIBUCHI, H.; NAKASHIMA, T.; MURATA, T. Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems. **IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics**, v. 29, n. 5, p. 601–618, out. 1999.

ISHIBUCHI, H.; YAMAMOTO, T. Fuzzy rule selection by multi-objective genetic local search algorithms and rule evaluation measures in data mining. **Fuzzy Sets and Systems**, Genetic Fuzzy Systems: New Developments. v. 141, n. 1, p. 59–88, 1 jan. 2004.

KAYA, M. Multi-objective Genetic Algorithm Based Approaches for Mining Optimized Fuzzy Association Rules. **Soft Comput.**, v. 10, n. 7, p. 578–586, maio 2006.

KIM, D.; CHOI, Y.-S.; LEE, S.-Y. An Accurate COG Defuzzifier Design Using Lamarckian Co-adaptation of Learning and Evolution. **Fuzzy Sets Syst.**, v. 130, n. 2, p. 207–225, set. 2002.

KIMURA, M. Evolutionary rate at the molecular level. **Nature**, v. 217, n. 5129, p. 624–626, 17 fev. 1968.

KOSHIYAMA, A. **GPFIS: Um Sistema Fuzzy-Genético Genérico baseado em Programação Genética**. Dissertação de Mestrado—Rio de Janeiro: PUC, 2014.

KOZA, J. R. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. Cambridge, MA, USA: MIT Press, 1992.

KRIJTHE, J. H.; HO, T. K.; LOOG, M. **Improving cross-validation based classifier selection using meta-learning**2012 21st International Conference on Pattern Recognition (ICPR). **Anais**... In: 2012 21ST INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION (ICPR). nov. 2012

KUNCHEVA, L. I. **Combining Pattern Classifiers: Methods and Algorithms**. [s.l.] Wiley-Interscience, 2004.

LANGDON, W. B.; POLI, R. **Fitness causes bloat: Mutation**In. **Anais**...Springer-Verlag, 1998

LOCAREK-JUNGE, H.; WEIHS, C. **Classification as a Tool for Research: Proceedings of the 11th IFCS Biennial Conference and 33rd Annual Conference of the Gesellschaft für Klassifikation e.V., Dresden, March 13-18, 2009**. [s.l.] Springer, 2010.

LUQUE, G.; ALBA, E. **Parallel genetic algorithms theory and real world applications**. Berlin; New York: Springer, 2011.

MCPHEE, N. F.; MILLER, J. D. **Accurate Replication in Genetic Programming**Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95. **Anais**...Morgan Kaufmann, 1995

MEHRAN, K. Takagi-Sugeno Fuzzy Modeling for Process Control. **School of Electrical, Electronic and Computer Engineer, Newcastle University**, 2008.

MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)**. London, UK, UK: Springer-Verlag, 1996.

MILLER, J. **What bloat? Cartesian Genetic Programming on Boolean problems**2001 GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE LATE BREAKING PAPERS. **Anais**...2001

MILLER, J. F. **Cartesian Genetic Programming**. 2011 edition ed. Heidelberg : New York: Springer, 2011.

MILLER, J. F.; HARDING, S. L. **Cartesian Genetic Programming**Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation

Conference: Late Breaking Papers. **Anais**...: GECCO '09.New York, NY, USA: ACM, 2009Disponível em: <http://doi.acm.org/10.1145/1570256.1570428>.

MILLER, J. F.; SMITH, S. L. Redundancy and computational efficiency in Cartesian genetic programming. **IEEE Transactions on Evolutionary Computation**, v. 10, n. 2, p. 167–174, abr. 2006.

MILLER, J. F.; THOMSON, P. Cartesian Genetic Programming. In: POLI, R. et al. (Eds.). **Genetic Programming**. Lecture Notes in Computer Science. [s.l.] Springer Berlin Heidelberg, 2000. p. 121–132.

NAUCK, D.; KLAWONN, F.; KRUSE, R. **Foundations of Neuro-Fuzzy Systems**. New York, NY, USA: John Wiley & Sons, Inc., 1997.

OHTA, T.; GILLESPIE, J. H. Development of neutral and nearly neutral theories. **Theoretical population biology**, v. 49, n. 2, p. 128–142, 1996.

PEDRYCZ, W. **Fuzzy control and fuzzy systems**. [s.l.] Research Studies Press, 1989.

POLI, R.; LANGDON, W. B.; MCPHEE, N. F. **A Field Guide to Genetic Programming**. [s.l.] Lulu Enterprises, UK Ltd, 2008.

PONGRACZ, R.; BOGARDI, I.; DUCKSTEIN, L. Application of fuzzy rule-based modeling technique to regional drought. **Journal of Hydrology**, v. 224, n. 3, p. 100–114, 1999.

ROSS, T. J. **Fuzzy Logic with Engineering Applications, Third Edition**. 3 edition ed. Chichester, U.K: Wiley, 2010.

SÁNCHEZ, L. et al. Some relationships between fuzzy and random set-based classifiers and models. **International Journal of Approximate Reasoning**, v. 29, n. 2, p. 175–213, fev. 2002.

SANDRI, S.; CORREA, C. Lógica nebulosa. 1999.

SAPANKEVYCH, N. I.; SANKAR, R. Time Series Prediction Using Support Vector Machines: A Survey. **Comp. Intell. Mag.**, v. 4, n. 2, p. 24–38, maio 2009.

SCHIFFNER, J. B. B. Bias-Variance Analysis of Local Classification Methods. p. 49–57, 2010.

SCHWEFEL, H.-P. **Evolution and Optimum Seeking**. 1 edition ed. New York: Wiley-Interscience, 1995.

SENGE, R.; HÜLLERMEIER, E. Top-Down Induction of Fuzzy Pattern Trees. **Trans. Fuz Sys.**, v. 19, n. 2, p. 241–252, abr. 2011.

SOULE, T. **Code Growth in Genetic Programming**. Moscow, ID, USA: University of Idaho, 1998.

SOULE, T.; FOSTER, J. A. **Removal bias: a new cause of code growth in tree based evolutionary programming**, The 1998 IEEE International Conference on Evolutionary

Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence. **Anais**... In: , THE 1998 IEEE INTERNATIONAL CONFERENCE ON EVOLUTIONARY COMPUTATION PROCEEDINGS, 1998. IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE. maio 1998

SOULE, T.; HECKENDORN, R. B. An Analysis of the Causes of Code Growth in Genetic Programming. **Genetic Programming and Evolvable Machines**, v. 3, n. 3, p. 283–309, set. 2002.

SOUZA, F. J.; VELLASCO, M. M. R.; PACHECO, M. A. C. Hierarchical Neuro-fuzzy Quadtree Models. **Fuzzy Sets Syst.**, v. 130, n. 2, p. 189–205, set. 2002.

SUN, C.-T.; JANG, J.-S. **A neuro-fuzzy classifier and its applications**, Second IEEE International Conference on Fuzzy Systems, 1993. **Anais**... In: , SECOND IEEE INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS, 1993. 1993

TAKAGI, T.; SUGENO, M. Fuzzy identification of systems and its applications to modeling and control. **Systems, Man and Cybernetics, IEEE Transactions on**, n. 1, p. 116–132, 1985.

TORRA, V. A review of the construction of hierarchical fuzzy systems. **International Journal of Intelligent Systems**, v. 17, n. 5, p. 531–543, 1 maio 2002.

TSANG, C.-H.; KWONG, S.; WANG, H. Genetic-fuzzy Rule Mining Approach and Evaluation of Feature Selection Techniques for Anomaly Intrusion Detection. **Pattern Recogn.**, v. 40, n. 9, p. 2373–2391, set. 2007.

VAPNIK, V. **The Nature of Statistical Learning Theory**. 2nd edition ed. New York: Springer, 1999.

VENTURINI, G. **SIA: A Supervised Inductive Algorithm with Genetic Search for Learning Attributes Based Concepts**Proceedings of the European Conference on Machine Learning. **Anais**...: ECML '93.London, UK, UK: Springer-Verlag, 1993Disponível em: <http://dl.acm.org/citation.cfm?id=645323.649578>.

WANG, D.; ZENG, X.-J.; KEANE, J. A. A Survey of Hierarchical Fuzzy Systems (Invited Paper). 2006.

WEBB, A. R. Density Estimation – Nonparametric. In: **Statistical Pattern Recognition**. [s.l.] John Wiley & Sons, Ltd, 2002. p. 81–122.

WITTEN, I. H.; FRANK, E. **Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

YU, T.; MILLER, J. **Neutrality and the Evolvability of Boolean Function Landscape**Genetic Programming, Proceedings of EuroGP'2001, volume 2038 of LNCS. **Anais**...Springer-Verlag, 2001

YU, T.; MILLER, J. F. **Finding Needles in Haystacks Is Not Hard with Neutrality**Proceedings of the 5th European Conference on Genetic Programming.

Anais...: EuroGP '02.London, UK, UK: Springer-Verlag, 2002Disponível em: <http://dl.acm.org/citation.cfm?id=646810.704112>.

ZADEH, L. A. Fuzzy sets. **Information and Control**, v. 8, n. 3, p. 338–353, jun. 1965.

ZHANG, W. **Complete Anytime Beam Search**Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence. **Anais**...: AAAI '98/IAAI '98.Menlo Park, CA, USA: American Association for Artificial Intelligence, 1998Disponível em: <http://dl.acm.org/citation.cfm?id=295240.295709>.

ZHANG, X.; ZHANG, N. **Universal Approximation of Binary-tree Hierarchical Fuzzy Systems with Typical FLUs**Proceedings of the 2006 International Conference on Intelligent Computing: Part II. **Anais**...: ICIC'06.Berlin, Heidelberg: Springer-Verlag, 2006Disponível em: <http://dl.acm.org/citation.cfm?id=1882540.1882565>.

ZHOU, A. et al. Multiobjective evolutionary algorithms: A survey of the state of the art. **Swarm and Evolutionary Computation**, v. 1, n. 1, p. 32–49, mar. 2011.