UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

César Eduardo Corrêa

Método para aumento de interpretabilidade em modelos de Takagi-Sugeno no sistema INFGMN

Florianópolis
2020

César Eduardo Corrêa

Método para aumento de interpretabilidade em modelos de Takagi-Sugeno no sistema INFGMN

Trabalho de Conclusão de Curso submetida ao Curso de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Ciências da Computação.
Orientador: Profº. Mauro Roisenberg, Dr.

Florianópolis
2020

César Eduardo Corrêa

Método para aumento de interpretabilidade em modelos de Takagi-Sugeno no sistema INFGMN

Trabalho de Conclusão de Curso submetida ao Curso de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Ciências da Computação.

Florianópolis, 19 de Novembro de 2020

Profº. Alexandre Gonçalves Silva, Dr.
Coordenador do Curso

Banca Examinadora:

Profº. Mauro Roisenberg, Dr.
Orientador
Universidade Federal de Santa Catarina

Profº. Elder Rizzon Santos, Dr.
Universidade Federal de Santa Catarina

Profª. Silvia Modesto Nassar, Dra.
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Ao meu orientador, Mauro Roisenberg, por aceitar meu pedido de orientação e me apoiar na realização deste trabalho.

Ao Tiago Mazzutti por disponibilizar o código da INFGMN para a realização deste trabalho.

A minha família pela companhia e apoio a todo o tempo.

RESUMO

Interpretabilidade e acurácia são objetivos conflitantes em modelos de aprendizado de máquina, onde a melhora de um geralmente causa a piora do outro. A interpretabilidade dos modelos gerados é importante para que possa ser validado por especialistas do domínio e passar confiabilidade para tratar o problema. Para buscar o melhor equilíbrio entre os dois, uma abordagem comum em Sistemas de Inferência Fuzzy é empregar um Modelo Fuzzy Linguístico e melhorar sua acurácia, ou um Modelo Fuzzy Preciso e melhorar sua interpretabilidade. O INFGMN é um sistema neurofuzzy que utiliza da equivalência entre um Modelo de Mistura de Gaussianas e um Sistema de Inferência Fuzzy para extração de conhecimento, sendo capaz de gerar modelos de Mamdani-Larsen, um Modelo Fuzzy Linguístico. Neste trabalho busca-se expandir as funcionalidades do INFGMN, proporcinando sua aplicação com modelos de Takagi-Sugeno, um Modelo Fuzzy Preciso, e melhorando sua interpretabilidade via distinguibilidade nas partições fuzzy, obtendo um bom equilíbrio entre acurácia e interpretabilidade. Para garantia de distinguibilidade, foi elaborado um método de similaridade de partição fuzzy para guiar a fusão e separação dos conjuntos fuzzy, ponderado pelos pesos das regras para gerar menores alterações nos conjuntos cujas regras descrevam melhor o sistema e não sejam oriundas de dados discrepantes. Os resultados apresentados para problemas *offline* demonstram a eficácia do método de similaridade utilizado, ao conseguir gerar uma partição fuzzy distinguível e, ademais, ser capaz de reduzir o *overfitting*. Para o experimento *online* com perturbações nos dados, os resultados não são muito promissores, mas sugerem ser uma boa alternativa para ambientes estáveis.

Palavras-chave: Lógica Fuzzy, Takagi-Sugeno, Sistemas Neuro-Fuzzy, Modelo de Mistura de Gaussianas, Interpretabilidade, Distinguibilidade, Similaridade, Partição Fuzzy.

ABSTRACT

Interpretability and accuracy are conflicting objectives in machine learning models, where improvement in one usually worsens the other. The interpretability of the generated models is important so that it can be validated by experts in the field and pass reliability to address the problem. To seek the best trade-off between the two, a common approach in Fuzzy Inference Systems is to employ a Linguistic Fuzzy Model and improve its accuracy, or an Accurate Fuzzy Model and improve its interpretability. The INFGMN is a neurofuzzy system that uses the equivalence between a Gaussian Mixture Model and a Fuzzy Inference System for knowledge extraction, being able to generate Mamdani-Larsen models, a Linguistic Fuzzy Model. This work seeks to expand the functionalities of the INFGMN, providing its application with Takagi-Sugeno models, a Precise Fuzzy Model, and improving its interpretability via distinguishability in fuzzy partitions, achieving a good trade-off between accuracy and interpretability. In order to guarantee distinguishability, a fuzzy partition similarity method was developed to guide the merging and separation of fuzzy sets, weighted by the weights of the rules to cause less changes in the sets whose rules better describe the system and are not derived from outliers. The results presented for *offline* problems demonstrate the effectiveness of the similarity method used, in being able to generate a distinguishable fuzzy partition and, in addition, being able to reduce *overfitting*. For the *online* experiment with data disturbances, the results are not very promising, but suggest that it is a good alternative for stable environments.

Keywords: Fuzzy Logic, Takagi-Sugeno, Neuro-Fuzzy Systems, Gaussian Mixture Model, Interpretability, Distinguishability, Similarity, Fuzzy Partition.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANFIS	*Adaptive Neuro-Fuzzy Inference System*
C-INFGMN	*Constrained INFGMN*
eFSM	*Evolving Neural-fuzzy Semantic Memory*
FIS	Sistema de Inferência Fuzzy (do inglês *Fuzzy Inference System*)
GFM	Modelo Fuzzy Generalizado
GMM	Modelo de Mistura de Gaussianas
IGMM	Modelo de Mistura de Gaussianas Incremental
INFGMN	*Incremental Neuro-Fuzzy Gaussian Mixture Network*
LFM	Modelagem Fuzzy Linguística
MISO	*Multi Input Single Output*
ML	Mamdani-Larsen
NFS	Sistema Neuro-Fuzzy (do inglês *Neuro-Fuzzy System*)
PDF	Função Densidade de Probabilidade
PFM	Modelagem Fuzzy Precisa
PSO	*Particle Swarm Optimization*
RMSE	*Root Mean Squared Error*
TS	Takagi-Sugeno

LISTA DE SÍMBOLOS

λ	Função de pertinência fuzzy
μ	Média da função de pertinência
C	Spread da função de pertinência
C	Matriz de covariâncias

SUMÁRIO

1 INTRODUÇÃO

Muitas técnicas de aprendizado de máquina geram modelos do tipo black-box, que não explicam o mapeamento entre entrada e saída. A interpretabilidade dos modelos gerados é importante para que possa ser validado por especialistas do domínio e passar confiabilidade para tratar o problema. Pesquisadores então trabalharam na construção de modelos que aprendiam com dados de treinamento para aquisição de conhecimento. (MAZZUTTI *et al.*, 2017)

Sistemas de Inferência Fuzzy utilizam da lógica fuzzy para modelar sistemas através de regras que manipulam variáveis linguísticas, cujos valores são palavras ao invés de números. Embora palavras sejam imprecisas, isso ajuda a lidar com a imprecisão do mundo e torna o sistema interpretável por imitar o pensamento humano. (MAZZUTTI *et al.*, 2017; MATHWORKS, 2019)

Interpretabilidade e acurácia são objetivos conflitantes, i.e, a melhora em um geralmente causa a piora do outro, gerando o dilema Acurácia-Interpretabilidade. Assim, busca-se o melhor equilíbrio entre os dois. Uma abordagem comum para obter o melhor equilíbrio entre acurácia e interpretabilidade é definir um objetivo principal e buscar formas de suprir o que lhe falta. O objetivo da Modelagem Fuzzy Linguística (LFM) é obter modelos fuzzy com boa interpretabilidade, enquanto a da Modelagem Fuzzy Precisa (PFM) é obter modelos fuzzy com boa acurácia. (CASILLAS *et al.*, 2003; ZHOU; GAN, 2008)

Modelos de Mamdani-Larsen (ML) utilizam regras fuzzy cujas partes do antecedente e consequente fazem uso somente de termos linguísticos (palavras), o que torna-os mais interpretáveis. Já em modelos de Takagi-Sugeno (TS), o consequente é uma função das variáveis de entradas, tornando-os mais precisos. (MATHWORKS, 2019; CASILLAS *et al.*, 2003)

Redes NeuroFuzzy incorporam a capacidade de aprendizado automático das redes neurais com a capacidade de representação de conhecimento dos sistemas fuzzy através de regras. INFGMN é um sistema neurofuzzy que utiliza da equivalência entre um Modelo de Mistura de Gaussianas (GMM) e um Sistema de Inferência Fuzzy (FIS) para extração de conhecimento. (MAZZUTTI *et al.*, 2018)

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Neste trabalho busca-se expandir as funcionalidades do sistema INFGMN, proporcinando sua aplicação com modelos de Takagi-Sugeno e melhorando sua interpretabilidade via distinguibilidade nas partições fuzzy. Isso concederá ao INFGMN uma nova abordagem pelo uso de uma Modelagem Fuzzy Precisa e a melhora de sua

interpretabilidade, obtendo um bom equilíbrio entre acurácia e interpretabilidade.

1.1.2 Objetivos Específicos

- Expandir o sistema INFGMN para trabalhar com modelos de Takagi-Sugeno
- Melhorar a distinguibilidade das partições fuzzy geradas pelo INFGMN
- Avaliar a interpretabilidade do novo sistema.

1.1.3 Estrutura do Trabalho

Fora a introdução, este trabalho é dividido em outros 5 capítulos. O segundo capítulo cobre conceitos referentes a Sistemas de Inferência Fuzzy. No terceiro capítulo estão trabalhos correlatos que implementaram algum tipo de método para melhoria de interpretabilidade em sistemas fuzzy. O quarto capítulo cobre a arquitetura e funcionamento do INFGMN, junto a trabalhos que comporam sua fundamentação teórica. No quinto capítulo está a proposta de melhoria para o sistema INFGMN. No sexto capítulo estão os resultados.

2 SISTEMAS DE INFERÊNCIA FUZZY

Sistema de Inferência Fuzzy (do inglês *Fuzzy Inference System*) (FIS) utilizam da logica fuzzy para modelar a imprecisão de sistemas do mundo real através de regras fuzzy. Suas tomadas de decisão são similares a dos humanos, tornando-os interpretáveis por utilizar o senso comum.

2.1 LÓGICA FUZZY

2.1.1 Conjuntos Fuzzy

Na teoria dos conjuntos clássicos, um dado elemento deve, ou pertencer, ou não pertencer a um determinado conjunto, e.g. $\{x \mid x \in \mathsf{Z} \wedge x \geq 10 \wedge x < 20\}$. Aqui vale a lei do terceiro excluído de Aristóteles: "Of any subject, one thing must be either asserted or denied". Porém, com essa rigidez, torna-se difícil modelar o pensamento humano. Como exemplo, se definirmos pessoas com mais de 1,8m como altas, alguém com 1,79m não seria uma pessoa alta nos conjuntos clássicos. Para lidar com esse empecilho, Zadeh introduziu a teoria dos conjuntos fuzzy para modelar a imprecisão dos conceitos. (MATHWORKS, 2019; ZADEH, 1965)

Um conjunto fuzzy é um conjunto sem um limite ‘crisp’ claramente definido, e pode conter elementos com um grau parcial de pertinência. Enquanto nos conjuntos clássicos o grau de pertinência possui um valor binário (0 ou 1), nos conjuntos fuzzy ele varia no intervalo [0,1]. Com isso, na lógica fuzzy, a verdade torna-se uma questão de grau de certeza. (MATHWORKS, 2019; ZADEH, 1965, 1988)

Um conjunto fuzzy é definido por um conjunto de pares ordenados $A = \{(x, \lambda_A(x)) \mid x \in U\}$, onde U é o universo de discurso, λ_A é a função de pertinência do conjunto A, e $\lambda_A(x)$ é o grau de pertinência do elemento x ao conjunto A. Uma função de pertinência descreve a transição no grau de pertinência do universo de discurso ao conjunto fuzzy, mapeando um espaço de entrada para um valor entre 0 e 1. Exemplos de funções de pertinência incluem: triangular, trapezoidal, gaussiana e sino generalizado. (MATHWORKS, 2019; ZADEH, 1965, 1973)

Uma variável linguística, como o nome sugere, é uma variável cujos valores são palavras/sentenças ao invés de números. Uma proposição "X is A", onde X é a variável linguística e A é uma palavra representada por um conjunto fuzzy, ela restringe os valores que X pode tomar, e indica a distribuição no grau de certeza que X é igual a A, com sua distribuição definida pela função de pertinência de A sobre o domínio de X (universo de discurso). Assim, cada palavra pode ser vista como uma descrição resumida de um subconjunto fuzzy A, representada por uma função de pertinência λ_A, associando objetos do universo de discurso, X, com transição gradual de pertinência para sua descrição. (ZADEH, 1973, 1988)

Com isso, a lógica fuzzy lida com variáveis linguísticas e a computação com palavras que, embora sejam menos precisas, são mais significativas para o pensamento humano, e exploram tolerância a imprecisão, realizando um bom equilíbrio entre significado e precisão. (MATHWORKS, 2019)

2.1.2 Operadores fuzzy

Como os conjuntos fuzzy são super-conjuntos dos conjuntos crisp clássicos, a tabela verdade deve se manter para valores extremos de pertinência. Na descrição com palavras, o complemento de conjuntos é representado pelo modificador NOT, e a intersecção e união de conjuntos representados pelos conectivos AND e OR, respectivamente. Exemplos típicos de operadores são $AND = \min$, $OR = \max$ e $NOT =$ *complemento aditivo*, demonstrado nas tabelas 3, 4, e 5. Outros operadores também podem ser utilizados, como $AND =$ *prod* (produtório), e $OR =$ *probor* (OU probabilístico). (MATHWORKS, 2019; ZADEH, 1973)

Tabela 3 – AND

A	B	$\min(A, B)$
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 4 – OR

A	B	$\max(A, B)$
0	0	0
0	1	1
1	0	1
1	1	1

Tabela 5 – NOT

A	$1 - A$
0	1
1	0

Para operadores AND (intersecção), referidos como T-norma, o mapeamento binário T(.,.) deve satisfazer as seguintes propriedades:

- Fronteira: $T(0, 0) = 0, T(a, 1) = T(1, a) = a$
- Monotonicidade: $T(a, b) \leq T(c, d)$ se $a \leq c$ e $b \leq d$
- Commutatividade: $T(a, b) = T(b, a)$
- Associatividade: $T(a, T(b, c)) = T(T(a, b), c)$

Para operadores OR (união), referidos como T-conorma (ou S-norma), o mapeamento binário S(.,.) deve satisfazer as seguintes propriedades:

- Fronteira: $S(1, 1) = 1, S(a, 0) = S(0, a) = a$
- Monotonicidade: $S(a, b) \leq S(c, d)$ se $a \leq c$ e $b \leq d$
- Commutatividade: $S(a, b) = S(b, a)$
- Associatividade: $S(a, S(b, c)) = S(S(a, b), c)$

2.2 REGRAS IF-THEN

Regras IF-THEN são usadas para formular asserções que compõem a lógica fuzzy, e capturam a relação entre as variáveis de entrada e de saída. A lógica fuzzy então mapeia um espaço de entrada para um espaço de saída através de regras IF-THEN, cujos termos antecedentes, ou premissas, são as variáveis de entradas, e os consequentes, ou conclusões, são as variáveis de saída. (MATHWORKS, 2019; ZADEH, 1973)

2.2.1 Processo de inferência

O processo de inferência de regras fuzzy é composto por 5 etapas: (MATHWORKS, 2019)

1. **Fuzzificação:** Para cada variável linguística de entrada é calculado o grau de pertinência a cada um dos conjuntos fuzzy (termos linguísticos) requeridos pelas regras fuzzy. Como exemplo, em "x is A", é calculado o valor de pertinência de x ao conjunto A.

2. **Operadores fuzzy:** O grau de ativação/suporte único de cada regra é calculado, aplicando-se os operadores correspondentes a cada termo no antecedente. Se o antecedente é composto por um único termo, o grau de pertinência do termo será o grau de suporte da regra.

3. **Implicação:** O valor de suporte é multiplicado pelo peso da regra (um valor no intervalo [0,1]). O resultado é aplicado numa função de implicação ao conjunto do consequente, alterando os graus de pertinência de seus elementos, e o conjunto fuzzy resultante é atribuído à saída da regra. Exemplos de funções de implicação são: *min*, e *prod*.

4. **Agregação:** Caso a variável de saída esteja associada a mais de uma regra, as saídas dessas regras são combinadas num conjunto de saída único por uma função de agregação. Exemplos de funções de agregação são: *max*, *probor*, e *sum*.

5. **Defuzzificação:** Etapa responsável por gerar um valor crisp de saída. Embora o processo de inferência utilize conjuntos fuzzy, o resultado final desejado geralmente é um valor numérico.

2.2.2 Mamdani-Larsen

Modelos de Mamdani são úteis para modelar sistemas especialistas[1] por representar o conhecimento através de regras que utilizam variáveis linguísticas

[1] sistemas que simulam o raciocínio de um especialista.

tanto no antecedente como no consequente, tornando-os altamente interpretáveis. (MATHWORKS, 2019; MAMDANI, 1974)

Modelos de Mamdani definem regras do tipo:

$$\text{IF } x \text{ is } A \text{ AND } y \text{ is } B \text{ THEN } z \text{ is } C$$

onde x, y, z são variáveis linguísticas, e A, B, C são termos linguísticos.

Para modelos de Mamdani, exemplos de métodos de defuzzificação incluem: centróide, bisector, Smallest of Maximum (SOM), Middle of Maximum (MOM), e Largest of Maximum (LOM). O mais popular é o centróide, retornando o centro da área sob a curva de pertinência do conjunto fuzzy de saída. (MATHWORKS, 2019)

2.2.3 Takagi-Sugeno

Modelos de Takagi-Sugeno (TS) são úteis para modelar sistemas não-lineares através da interpolação de múltiplos modelos lineares. Suas regras compõem variáveis linguísticas no antecedente, assim como no modelo de Mamdani, porém o consequente é uma função das variáveis de entrada. (MATHWORKS, 2019; TAKAGI; SUGENO, 1985)

Modelos de TS definem regras do tipo:

$$\text{IF } x \text{ is } A \text{ AND } y \text{ is } B \text{ THEN } z = f(x, y)$$

onde x, y são variáveis linguísticas de entrada, A, B são termos linguísticos, z é a variável de saída, e $f(x, y)$ é a função de saída.

A função de saída é da forma $f(x, y) = a_i x + b_i y + c_i$, onde a_i, b_i, c_i são os coeficientes da função linear. No caso de um sistema TS de ordem-0, $f(x, y)$ é uma constante ($a_i = b_i = 0$). (MATHWORKS, 2019)

Cada regra gera dois valores:

- w_i: o peso (grau de ativação da regra) da i-ésima regra, derivado do antecedente da regra nas 2 primeiras etapas de inferência.
- z_i: saída da i-ésima regra, calculado pela função no consequente da regra.

Modelos de Sugeno só trabalham com produto como método de implicação, e somatório como método de agregação. Para defuzzificação, os métodos são wtaver (Weighted Average) e wtsum (Weighted Sum). (MATHWORKS, 2019; TAKAGI; SUGENO, 1985)

No método *wtsum*, a saída final equivalerá à saída da etapa de agregação:

$$\sum_{i=1}^{N} w_i z_i$$

No método *wtaver*, a saída da etapa de agregação será ponderada pelo somatório dos graus de ativação das regras:

$$\frac{\sum_{i=1}^{N} w_i z_i}{\sum_{i=1}^{N} w_i}$$

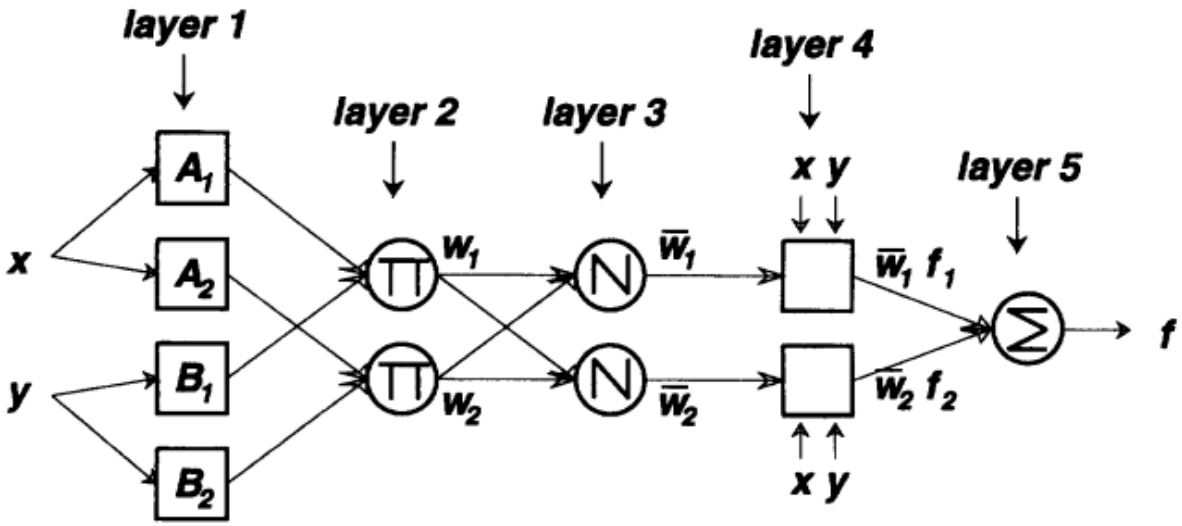
2.3 REDES NEURO-FUZZY

Redes Neuro-Fuzzy, ou Sistema Neuro-Fuzzy (do inglês *Neuro-Fuzzy System*) (NFS), incorporam técnicas de aprendizado automático das redes neurais com a capacidade de representação de conhecimento dos sistemas fuzzy. Além da etapa de aprendizado a partir de conjuntos de treinamento fornencido pelas técnicas de redes neurais, o sistema fuzzy pode ser otimizado com o conhecimento de especialistas por meio da manipulação da base de regras. (MAZZUTTI *et al.*, 2018; JANG; SUN, 1995)

2.3.1 ANFIS

Adaptive Neuro-Fuzzy Inference System (ANFIS) é uma rede *feedfoward* com aprendizagem por descida em gradiente e quadrados mínimos, e gera um FIS do tipo Sugeno. Ela possui 5 camadas, e cada uma desempenha um papel característico equivalente ao processo de inferência de um FIS. A figura 1 ilustra o arquitetura para um sistema com 2 entradas, x e y, e uma saída, f. (JANG; SUN, 1995)

Figura 1 – Arquitetura da rede ANFIS.



Fonte – Jang e Sun (1995)

A camada **1** é responsável pela fuzzificação, e seus nodos são adaptativos. Ela abriga os conjuntos fuzzy das premissas de cada regra, onde cada nodo, atrelado a uma função de pertinência, calcula o grau de pertinência da devida variável de entrada.

A camada **2** é responsável pelo operador fuzzy, e possui nodos fixos (sem parâmetros). Ela realiza a operação AND (produtório) dos valores de pertinência para cada premissa, obtendo o grau de ativação de cada regra.

A camada **3** também possui nodos fixos, e é responsável pela normalização das ativações das regras, a fim de simular a defuzzificação por média ponderada (Weighted Average) na última camada.

A camada **4** é responsável pela implicação, e seus nodos são adaptativos. Ela abriga os consequentes das regras, onde cada nodo, atrelado à função de saída de uma regra, é responsável por calcular seu valor de saída, ponderado pelo grau de ativação normalizado da regra.

A camada **5** é responsável pela agregação/defuzzificação, e possui 1 único nodo fixo que realiza a soma das implicações de cada regra. Como os graus de ativação de cada regra já foram normalizados na camada **3**, a defuzzificação por média ponderada já vem embutida.

2.4 INTERPRETABILIDADE EM SISTEMAS FUZZY

Há três diferentes paradigmas na modelagem de sistemas. De um lado, a modelagem White-box utiliza-se de parâmetros que caracterizam o sistema com significados claros e interpretáveis, porém, sofrem dificuldades quando lidam com sistemas complexos. Do outro lado, a modelagem black-box não utiliza nenhum conhecimento a priori, criando sua estrutura de conhecimento a partir de dados. Isso permite-lhe simular problemas complexos do mundo real precisamente, porém, seus parâmetros não explanam o comportamento do sistema. No meio-termo surge a modelagem grey-box, onde conhecimentos prévios sobre o sistema são considerados, e as partes desconhecidas do sistema são obtidas por meio de abordagens black-box. (CASILLAS *et al.*, 2003; ZHOU; GAN, 2008)

A modelagem fuzzy, que descreve o comportamento de sistemas através da lógica fuzzy estabelecendo relacionamentos entre entradas e saídas, é uma das ferramentas mais sucedidas para desenvolvimento de modelos grey-box, e possui duas características que avaliam sua qualidade (CASILLAS *et al.*, 2003):

Interpretabilidade: Refere-se a capacidade do modelo expressar o comportamento do sistema de uma forma compreensível. Esta é uma propriedade subjetiva e que depende de diversos fatores, os quais serão listados mais a frente.

Acurácia: Refere-se a capacidade do modelo representar precisamente o sistema modelado. Quanto mais perto suas respostas forem do sistema real, maior sua

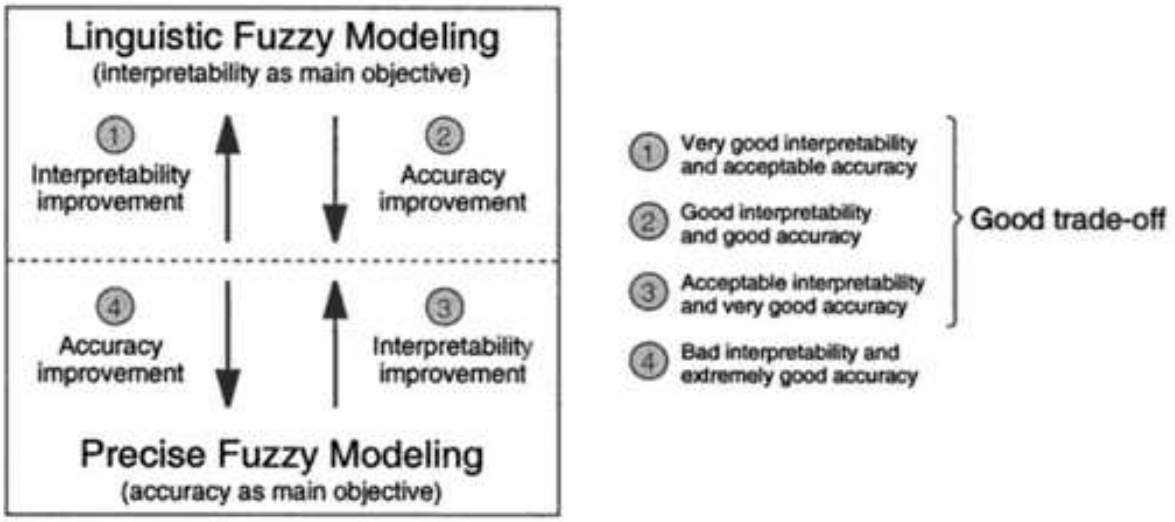
acurácia.

Interpretabilidade e acurácia são objetivos conflitantes, i.e, a melhora de um geralmente causa na piora do outro e, na prática, uma das propriedades prevalece. Uma abordagem comum para obter o melhor equilíbrio entre acurácia e interpretabilidade é definir o objetivo principal e buscar formas de suprir o que lhe falta. Dependendo de qual o objetivo buscado, a modelagem fuzzy pode ser dividida em 2 áreas: (CASILLAS *et al.*, 2003)

Modelagem Fuzzy Linguística (LFM): Modelagem cujo objetivo é obter modelos fuzzy com boa interpretabilidade. Aqui emprega-se principalmente modelos de Mamdani-Larsen.

Modelagem Fuzzy Precisa (PFM): cujo objetivo é obter modelos fuzzy com boa acurácia. Aqui emprega-se principalmente modelos de Takagi-Sugeno.

Figura 2 – Equilíbrio entre acurácia e interpretabilidade na modelagem fuzzy.

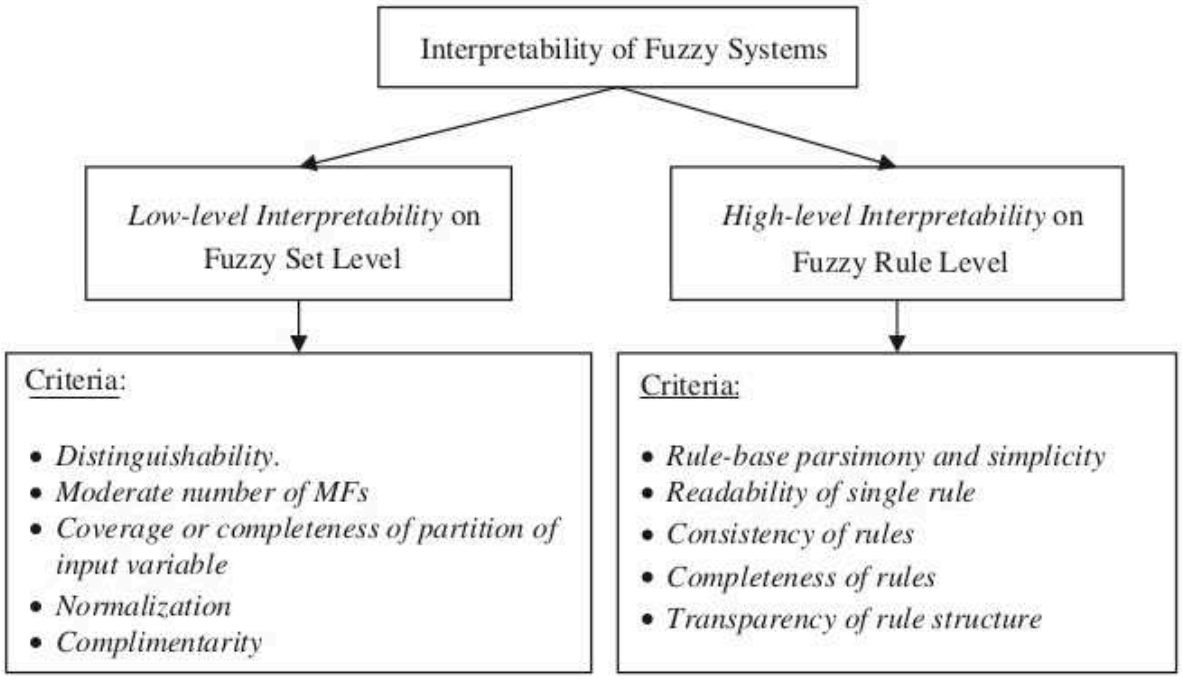


Fonte – Casillas *et al.* (2003)

Dentre as 4 direções apresentadas na figura 2, o melhor equilíbrio está nas abordagens ② (melhora na acurácia de LFM) e ③ (melhora na interpretabilidade de PFM). Embora LFM usem da modelagem interpretável, a abordagem ① (melhora na interpretabilidade de LFM) pode ser útil quando o processo de aquisição de conhecimento gera modelos não tão interpretáveis como desejado, justificando a melhoria. Já a abordagem ④ não se importa com a interpretabilidade, agindo próxima de modelos black-box. (CASILLAS *et al.*, 2003)

Zhou e Gan (2008) definem uma taxonomia para a interpretabilidade em sistemas fuzzy, separando nos conceitos de interpretabilidade de baixo-nível e interpretabilidade de alto-nível. A de baixo-nível é alcançada no nível de conjuntos fuzzy, otimizando as funções de pertinência, enquanto a de alto-nível é alcançada no nível de regras fuzzy, conduzindo uma redução na complexidade geral, como exemplo, a redução do número de regras. Eles também descrevem os critérios semânticos para alcançar interpretabilidade de baixo e de alto-nível, resumido na figura 3:

Figura 3 – Taxonomia de interpretabilidade em Sistemas Fuzzy.



Fonte – Zhou e Gan (2008)

Critérios semânticos para alcançar interpretabilidade de baixo-nível:

Distinguibilidade: Para cada espaço de entrada, os conjuntos fuzzy devem definir claramente seus intervalos de pertinência e serem suficientemente distintos um do outro para poderem ser representados por um termo linguístico com um significado semântico claro.

Número moderado de MFs: O número de MFs deve ser compatível com a quantidade de entidades conceituais que o ser humano consegue lidar. Segundo a psicologia cognitiva, o número de entidades eficientemente armazenadas na memória de curto-prazo é 7 ± 2.

Cobertura ou completude da partição fuzzy: O universo de discurso de uma variável deve ser coberto pelas MFs, e todo ponto deve pertencer a ao menos um conjunto fuzzy.

Normalização: Todo conjunto fuzzy deve ter ao menos um ponto com valor de pertinência 1.

Complementaridade: Para cada ponto no universo de discurso, a soma de seus valores de pertinência a cada conjunto fuzzy deve ser igual a 1 para sistemas fuzzy de probabilidade, ou estar no intervalo (0,1] para sistemas fuzzy de possibilidade.

O objetivo da modelagem fuzzy de baixo-nível é encontrar um equilíbrio entre a interpretabilidade da partição fuzzy e o desempenho do modelo global, gerando uma partição padronizada enquanto mantém a acurácia num nível aceitável. “A fuzzy partition in input space is called a standardized partition if it satisfies all the semantic criteria.” (ZHOU; GAN, 2008)

Critérios para alcançar interpretabilidade de alto-nível:

Simplicidade e parcimônia na base de regras: Em acordo com o princípio da navalha de Occam, o conjunto de regras fuzzy deve ser tão pequeno quanto possível, mas mantendo uma boa acurácia.

Legibilidade de regra: Para melhorar a legibilidade, o número de condições na premissa da regra não deve exceder o limite de 7 ± 2 (número de conceitos que o ser humano consegue lidar eficientemente).

Consistência: Regras com premissas similares devem conter consequentes similares.

Completude: Para qualquer possível vetor de entrada, ao menos uma regra deve ser disparada, para prevenir a indeterminação.

Transparência na estrutura da regra: Por modelos ML utilizarem termos linguísticos tanto no antecedente como no consequente das regras, é aceito como um modelo transparente por padrão. Para modelos TS, cujo consequente é uma função, a transparência é alcançada caso cada consequente seja uma representação do modelo global em sua respectiva região de ativação.

Zhou e Gan (2008) também descrevem 4 esquemas para alcançar interpretabilidade de baixo-nível:

Técnicas de regularização para estimação de parâmetros: Uma vez formalizada matematicamente uma restrição semântica na forma de função de custo para um critério semântico, há potencial de usá-la em funções objetivas. Uma função

objetiva com regularização é da forma $J_{overall} = J_G + \sum_i \lambda_i J_i$, em que J_G é a função de aprendizado global, λ_i são os parâmetros de regularização, e J_i são as funções de restrição semântica.

Otimização multi-objetiva para estimação de parâmetros: Assim como as técnicas de regularização, outra forma de utilizar critérios semânticos na modelagem fuzzy é combinando restrições semânticas com a função global de acurácia. Um exemplo de função multi-objetiva tem a forma $J_{overall} = (1 + \lambda S)J_G$, em que J_G é a função de aprendizado global, e S é a função de restrição semântica.

Fusão de conjuntos fuzzy: A fim de produzir um particionamento distinguível do espaço de entrada, fusão de conjuntos fuzzy é uma escolha natural, sendo guiado, por exemplo, por medidas de similaridade entre conjuntos fuzzy.

Modelagem interativa orientada ao usuário: Devido a interpretação de modelos depender muito do conhecimento a priori do usuário, torna-se uma tarefa subjetiva. Então, um método de modelagem é permitindo aos usuários envolverem-se no processo de geração do modelo, de forma que o sistema final esteja nos termos de validação subjetivos do usuário. NEFCLASS/NEFPROX são 2 implementações deste tipo, e que permitem ao usuário deletar antecedentes e conjuntos fuzzy. Porém, devido às opções oferecidas para manipulação do sistema e com diferentes combinações e estratégias, pode-se levar muito tempo para obter um modelo fuzzy com interpretabilidade e acurácia aceitáveis.

3 REVISÃO DE TRABALHOS RELACIONADOS

3.1 DISTINGUISHABILITY QUANTIFICATION OF FUZZY SETS

Segundo Mencar *et al.* (2007), a distinguibilidade pode ser quantificada por meio de medidas de similaridade e, na análise de interpretabilidade, a mais comumente adotada é a da equação 3.1, onde frequentemente o mínimo e máximo são usados como T-norma e T-conorma, permitindo ser reescrita como a equação 3.2. Uma medida para calcular a distinguibilidade seria então definida como o complemento da similaridade: $\Delta(A, B) = 1 - S(A, B)$.

$$S(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

$$S(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{3.2}$$

Mencar *et al.* (2007) demonstram uma relação entre medida de possibilidade e medida de similaridade, e mencionam que a de possibilidade frequentemente é usada para o cálculo de distinguibilidade, sendo calculada como:

$$\Pi(A, B) = \sup_{x \in U} \min \{\mu_A(x), \mu_B(x)\}$$

i.e, o grau máximo de verdade para algum x em ser A e B ao mesmo tempo, ou, a extensão até a qual A e B sobrepõem. Esta medida corresponde ao grau de separação de conjuntos fuzzy convexos descrito por Zadeh (1965).

Novamente, a distinguibilidade seria calculada como o complemento, neste caso, da medida de possibilidade:

$$\Delta(A, B) = 1 - \Pi(A, B)$$

Para que valha a relação entre as medidas de possibilidade e similaridade, os conjuntos fuzzy A e B devem satisfazer às restrições de normalidade, convexidade e continuidade. Adicionalmente, a derivada segunda deve satisfazer a seguinte restrição:

$$\forall x \in]p_A, x_v[\colon \frac{d^2 \mu_A}{dx^2}(x) \leq 0$$

$$\forall x \in]x_v, p_B[\colon \frac{d^2 \mu_B}{dx^2}(x) \leq 0$$

onde:

$p_A \in \arg_{x \in U} \max(\mu_A(x))$ é o valor de x que maximiza a pertinência em A

$p_B \in \arg_{x \in U} \max(\mu_B(x))$ é o valor de x que maximiza a pertinência em B

$v = \Pi(A, B)$ é a medida de possibilidade entre A e B

$p_A < p_B$, e $x_v \in [p_A, p_B]$, tal que $\mu_A(x_v) = \mu_B(x_v) = v$.

Dadas as restrições, os autores calculam a similaridade máxima entre os conjuntos A e B, maximizando a intersecção, e minimizando a união:

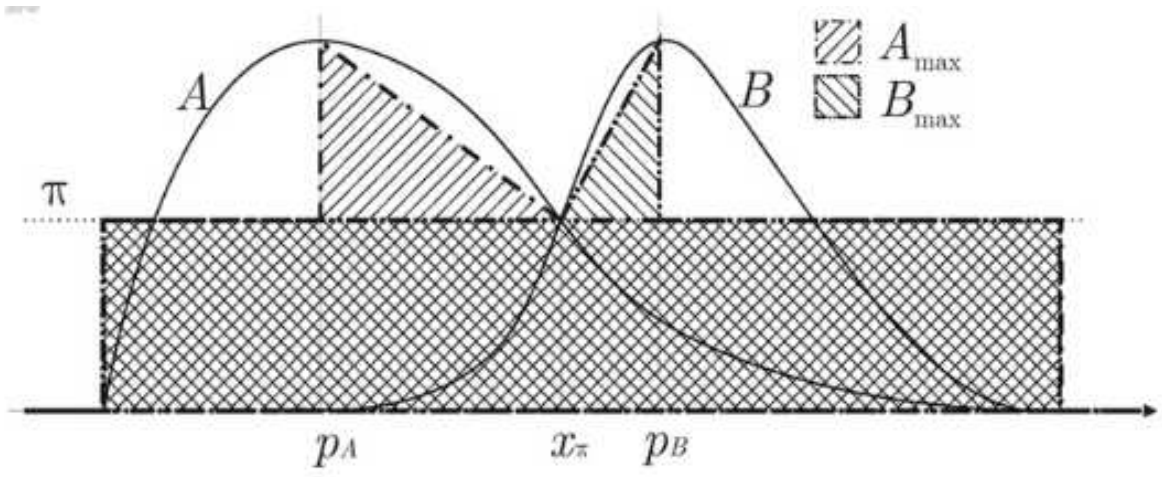
$$S(A_{max}, B_{max}) = \frac{v|\text{supp}A \cup B|}{v|\text{supp}A \cup B| + (1-v)(p_B - p_A)/2}$$

onde:

$|\text{supp}A \cup B|$ é o comprimento do suporte da união de A e B

A figura 4 ilustra a ideia do cálculo:

Figura 4 – Similaridade máxima para uma dada medida de possibilidade.



Fonte – Mencar *et al.* (2007)

A similaridade máxima pode então ser reescrita como:

$$S_{max} = S(A_{max}, B_{max}) = \frac{2v}{2v + r(1-v)}$$

tal que:

$$r = \frac{p_B - p_A}{|\text{supp}A \cup B|}$$

Já que $r > 0$, e $v \in [0, 1]$, então $S_{max} \leq v = \Pi(A, B)$

Um problema surge quando funções gaussianas são adotadas, já que podem não satisfazer à restrição da derivada segunda. Os autores também tratam brevemente funções gaussianas, porém Hefny (2007) é mais abrangente.

3.1.1 Comments on "Distinguishability quantification of fuzzy sets"

Hefny (2007) demonstra o cálculo da medida de similaridade para funções gaussianas, e sua relação com a medida de possibilidade para diferentes combinações de desvios padrões.

Utilizando a mesma medida de similaridade de Mencar *et al.* (2007), Hefny (2007) elabora os cálculos das cardinalidades dos conjuntos envolvendo integrais, e chega nas equações 3.3 a 3.9.

$$S(A,B) = \frac{\Psi}{2-\Psi} \tag{3.3}$$

$$\Psi = \beta + (1-\beta)erf\left[\left(\frac{1}{1-\beta}\right)\sqrt{-ln(v)}\right] - erf\left(\sqrt{(}-ln(v))\right) \tag{3.4}$$

$$v = \Pi(A,B) = exp\left[-\left(\frac{p_A - p_B}{\sigma_A + \sigma_B}\right)^2\right] \tag{3.5}$$

$$\beta = \frac{2\sigma_{min}}{\sigma_{max} - \sigma_{min}} \tag{3.6}$$

$$\sigma_{max} = max(\sigma_A, \sigma_B) \tag{3.7}$$

$$\sigma_{min} = min(\sigma_A, \sigma_B) \tag{3.8}$$

$$erf(x) = \frac{2}{\sqrt{(}\pi)}\int_0^x e^{-t^2}dt \tag{3.9}$$

A relação entre a medida de possibilidade e a de similaridade é descrita pela figura 5, variando conforme o valor de β.

3.2 A LINGUISTICALLY INTERPRETABLE ELANFIS FOR CLASSIFICATION PROBLEMS

CELANFIS foi desenvolvido por Pramod e Pillai (2018). Ele utiliza o algoritmo de sub-clusterização de Chiu, em que para cada instância de treinamento é calculado seu potencial como centro de um cluster (função de pertinência), conforme a densidade de sua vizinhança. O candidato com maior potencial é utilizado para a criação de um cluster, e dos demais é decrescido um fator da sua distância ao novo cluster. Este processo de obter um novo cluster e decrescer o potencial dos demais é repetido até que todos os potenciais estejam abaixo de um limite.

Na etapa de criação das funções de pertinência, os parâmetros são construídos conforme:

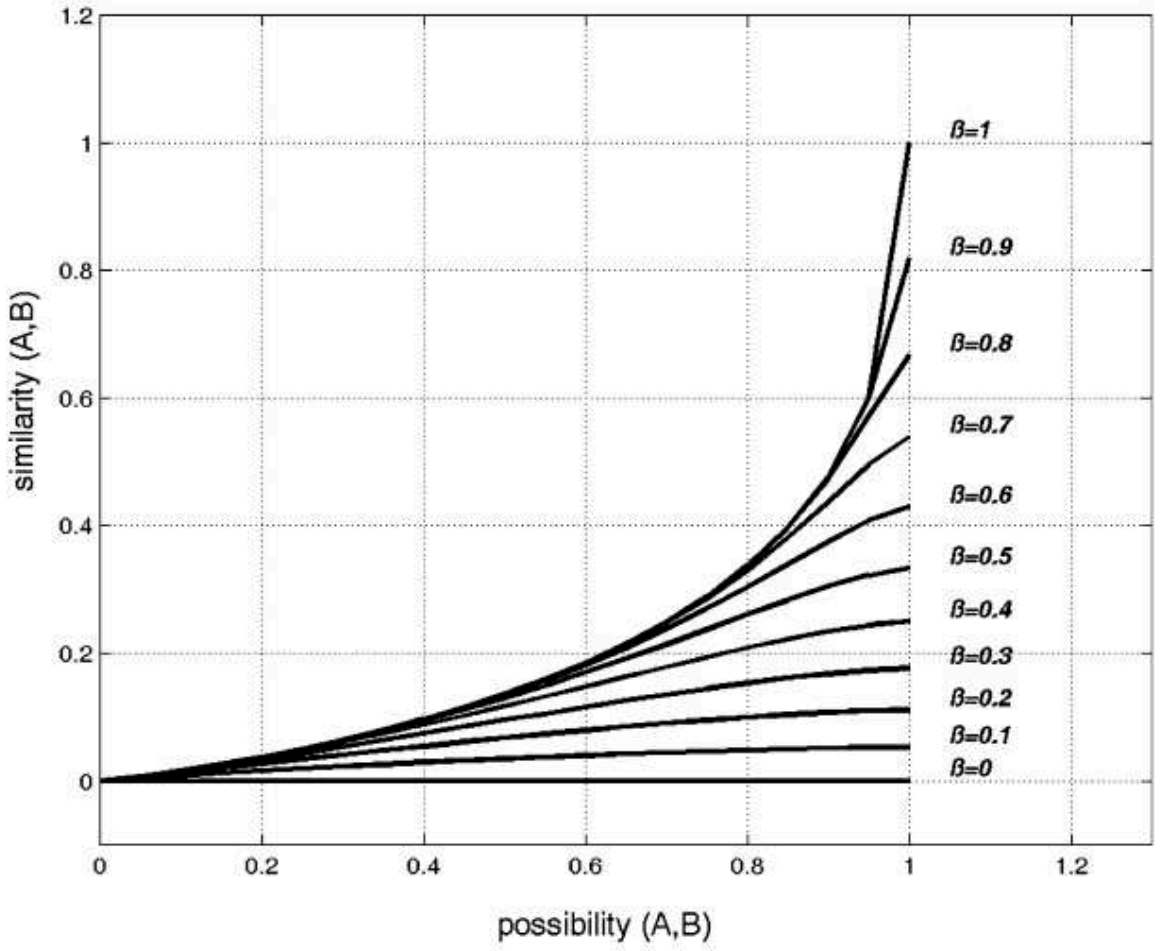
$$c_{id} = x_{di} + (-1 + 2r_1) \times \frac{0.1 \times 2}{L-1}$$

$$a_{id} = (0.8 + 4r_2) \times \min\left(\frac{x_{di} - x_{d(i-1)}}{2}, \frac{x_{d(i+1)} - x_{di}}{2}\right)$$

$$b_{id} = 2 + r_3$$

onde x_{di} é o centro na d-ésima dimensão do i-ésimo cluster, c_{id}, a_{id} e b_{id} são os parâmetros da função sino generalizada correspondente, L é o número de clusters, e r_1, r_2 e r_3 são valores aleatórios entre 0 e 1. r_3 será igual entre funções da mesma dimensão. Assim, em média, as funções terão seus centros conforme os clusters obtidos, e suas larguras serão aproximadamente $\frac{3}{2}$ da distância ao cluster mais próximo.

Figura 5 – Relação entre medida de possibilidade e similaridade de funções Gaussianas.



Fonte – Hefny (2007)

Para cada 2 funções vizinhas de parâmetros (a_1, b_1, c_1) e (a_2, b_2, c_2), com $c_1 < c_2$, a similaridade entre elas é calculada como o grau de pertinência na intersecção das 2 funções, calculado como:

$$S = \frac{1}{1 + \left|\frac{c_2 - c_1}{a_2 + a_1}\right|^{2b_1}}$$

O critério para obter distinguibilidade é definido como $0.4 < S < 0.6$. Caso $S >= 0.9$, as funções são fundidas e seus novos parâmetros aproximarão a união dos 2 conjuntos fuzzy, conforme:

$$c_{new} = \frac{((c_1 - a_1) + (c_2 + a_2))}{2}$$

$$a_{new} = (c_{new} - (c_1 - a_1)) = ((c_2 + a_2) - c_{new})$$

Caso $0.6 < S < 0.9$, os centros são movidos para longe um do outro. E para perto caso $S < 0.4$. Suas larguras são reduzidas no mesmo fator conforme as mu-

danças no centro, assim, os graus de pertinência dos elementos que estavam fora da sobreposição são mantidos similares.

Uma vez que as funções de pertinência satisfaçam as restrições de interpretabilidade, o parâmetro b de cada função é atualizado conforme:

$$b_1 = \frac{\ln(99)}{2 \times \ln\left(\left|\frac{c_2 - c_1}{a_1}\right|\right)}$$

3.3 INTERPRETABILITY CONSTRAINTS FOR FUZZY MODELING IMPLEMENTED BY CONSTRAINED PARTICLE SWARM OPTIMIZATION

Tsekouras, Tsimikas *et al.* (2017) apresentam uma proposta de modelagem baseada em *Particle Swarm Optimization* (PSO), com restrições de desigualdade para manter a interpretabilidade.

As restrições são descritas como:

$$g_{ij} = \mu_{ij} - \mu_{i+1,j} + \gamma\Delta_j \leq 0, i = 1, .., n_j - 1, j = 1, .., p \tag{3.10}$$

$$u_{ij} = \sigma_{ij} + \sigma_{i+1,j} - (\gamma/\phi)\Delta_j \leq 0, i = 1, .., n_j - 1, j = 1, .., p \tag{3.11}$$

$$f_{0j} = \mu_{1j} - L_j - \sigma_{1j}\sqrt{-ln(\alpha)} \leq 0, j = 1, .., p \tag{3.12}$$

$$f_{ij} = \mu_{i+1,j} - \mu_{i,j} - (\sigma_{i+1,j} + \sigma_{ij})\sqrt{-ln(\alpha)} \leq 0, i = 1, .., n_j - 1, j = 1, .., p \tag{3.13}$$

$$f_{n_j,j} = U_j - \mu_{n_j,j} - \sigma_{n_j,j}\sqrt{-ln(\alpha)} \leq 0, j = 1, .., p \tag{3.14}$$

$$\mu_{ij} \in \Omega_j \text{ e } \sigma_{ij} \in \Phi_j, i = 1, .., n_j, j = 1, .., p \tag{3.15}$$

para $n_j \geq 2 \forall j, p \geq 1, \gamma \in S_\gamma, \phi \in S_\phi$, e $\alpha \in S_\alpha$ onde:

p é o número de dimensões de entrada;

n_j é o número de funções de pertinência na j-ésima dimensão de entrada;

μ_{ij} é o centro da i-ésima gaussiana na j-ésima dimensão de entrada;

σ_{ij} é o desvio padrão da i-ésima gaussiana na j-ésima dimensão de entrada;

$\Omega_j = [L_j, U_j]$ é o universo de discurso da j-ésima dimensão de entrada;

$\Delta_j = \frac{(U_j - L_j)}{(n_j - 1)}$ representa o espaçamento ideal entre as funções gaussianas na j-ésima dimensão;

α é o nível de cobertura mínimo das partições;

$S_\alpha = [0.5, 0.6]$ é o intervalo aceitável para α;

ϕ é uma variável para limitar a sobreposição entre gaussianas;

$S_\phi = [0.4, 0.5]$ é o intervalo aceitável para ϕ;

γ é a proporção para definir o espaçamento mínimo entre os centros das gaussianas;

$S_\gamma = [0.5, 0.75]$ é o intervalo aceitável para γ;

$\Phi_j = [(2/3)\gamma\Delta_j, \Delta_j]$ é o intervalo aceitável para os desvios padrões na j-ésima dimensão de entrada;

g_{ij} restringe o espaçamento entre os centros das gaussianas a um mínimo de $\gamma\Delta_j$;

u_{ij} restringe a medida de possibilidade entre as gaussianas a um máximo de $exp(-\Phi^2)$;

$f_{0j}, f_{n_j,j}$ e f_{ij} restringem o nível de cobertura das partições a um mínimo de α;

Dado que a medida de possibilidade é:

$$v = exp\left[-\left(\frac{\mu_{i+1,j} - \mu_{ij}}{\sigma_{i+1,j} + \sigma_{ij}}\right)^2\right]$$

Os seguintes cálculos demonstram que u_{ij} limita a sobreposição entre gaussianas através da medida de possibilidade:

$$\begin{aligned} & u_{ij} \Rightarrow \phi \cdot (\sigma_{ij} + \sigma_{i+1,j}) \leq \gamma\Delta_j \\ & g_{ij} \Rightarrow \gamma\Delta_j \leq -(\mu_{ij} - \mu_{i+1,j}) \\ \therefore \quad & \phi \leq (\mu_{i+1,j} - \mu_{ij})/(\sigma_{ij} + \sigma_{i+1,j}) = \sqrt{-ln(v)} \\ & \implies exp(-\phi^2) \geq v \end{aligned}$$

Portanto, restringindo a medida de possibilidade máxima para v_{max}, onde:

$$\phi \in S_\phi \Rightarrow v_{max} \in [exp(-0.5^2), exp(-0.4^2)] \approx [0.7788, 0.8521]$$

A partir das restrições impostas, deve-se minimizar a função objetivo J:

$$J = \sum_{k=1}^{N} (Y_k - y_k)^2$$

(Y_k a saída esperada, e y_k a saída prevista) sem violar as restrições, e tendo o valor de violação das restrições calculado como:

$$CV = \sum_{j=1}^{p} C_j$$

tal que:

$$\begin{aligned} C_j = \sum_{i=1}^{n_j - 1} & \left(\max\{f_{ij}, 0\} + \max\{g_{ij}, 0\} + \max\{u_{ij}, 0\}\right) \\ & + \max\{f_{0j}, 0\} + \max\left\{f_{n_j,j}, 0\right\} \end{aligned}$$

Eles utilizam as regras de Deb: (1) qualquer solução viável é preferível sobre qualquer solução inviável; (2) entre 2 soluções viáveis, a com melhor função objetivo é preferível; (3) entre 2 soluções inviáveis, a de menor violação das restrições é preferível.

O PSO envolve Np partículas, e cada partícula representa um particionamento do espaço de entrada (um vetor com os parâmetros de todas as funções de pertinência). A cada uma é atribuída uma velocidade.

$$h_i(t+1) = \omega h_i(t) + \varphi_1 U(0,1) \otimes (p_i^{best}(t) - p_i(t)) + \varphi_2 U(0,1) \otimes (p_{best}(t) - p_i(t))$$

onde U(0,1) é um vetor de números aleatórios no intervalo [0,1], ω é a inércia, φ_1 é o parâmetro cognitivo e φ_2 é o parâmetro social. $p_i^{best}(t)$ é a melhor solução encontrada pela i-ésima partícula, e $p_{best}(t)$ é a melhor solução dentre todas as partículas.

Cada partícula é atualizada como:

$$p_i(t+1) = p_i(t) + h_i(t+1)$$

Cada partícula é decomposta num particionamento do espaço de entrada, os parâmetros dos consequentes são estimados (pela regressão de ridge), e a partícula é então validada conforme a função objetivo e o valor de violação das restrições.

Atualiza-se os valores de $p_i^{best}(t)$ e $p_{best}(t)$, se for o caso (pelas regras de Deb).

Esse processo (cálculo da velocidade, atualização da partícula, e validação da partícula, atualização de $p_i^{best}(t)$ e $p_{best}(t)$), denominado módulo PSO, é realizado t_{max} vezes.

Após essa iteração, a cada dimensão de p_{best} que viola alguma restrição, realiza-se a fusão dos 2 conjuntos fuzzy mais similares. Para Q_j pontos equidistantes no j-ésimo universo de discurso Ω_j, a similaridade dos conjuntos A e B é calculada conforme:

$$S(A,B) = \frac{\sum_{k=1}^{Q_j} [A(x_k) \wedge B(x_k)]}{\sum_{k=1}^{Q_j} [A(x_k) \vee B(x_k)]}$$

O novo conjunto fuzzy terá a média dos centros e dos desvios padrões.

Se uma fusão ocorrer, um novo conjunto de partículas aleatórias é gerado, contendo p_{best}, e o processo do módulo PSO é executado 1 vez. Então, são verificadas as similaridades entre os antecedentes das regras de p_{best}, conforme:

$$S(R_\alpha, R_\beta) = \min \{S(A_{\alpha,j}, A_{\beta,j})\}$$

A fusão de regras será requerida caso $S(R_\alpha, R_\beta) \geq \theta$, para $\theta \in (0,1)$ informado pelo usuário.

Enquanto ocorra uma fusão de conjuntos, sem precisar de uma fusão de regras, todo esse processo de otimização fica se repetindo. Caso contrário, finaliza-se o algoritmo de otimização (executando uma última vez o módulo PSO t_{max} vezes para explorar bem o espaço de soluções viáveis).

3.4 EFSM — A NOVEL ONLINE NEURAL-FUZZY SEMANTIC MEMORY MODEL

O eFSM possui 5 camadas. Na camada 1 (Input Layer), cada nodo corresponde a uma variável de entrada. A camada 2 (Antecedent Layer) consiste dos conjuntos fuzzy de entrada. Na camada 3 (Rule Layer) são mantidas as regras do tipo Mamdani-Larsen. A camada 4 (Consequent Layer) consiste dos conjuntos fuzzy de saída. Na camada 5 (Output Layer), cada nodo corresponde a uma variável de saída.

Ele possui 2 fases: aprendizado estrutural e aprendizado de parâmetros.

No aprendizado estrutural, inicialmente, o eFSM verifica a necessidade de criação de novas regras, calculando o grau de ativação na premissa de cada regra, e também o grau de ativação inversa no consequente de cada regra (tratando como se fosse uma premissa, cujas variáveis são as de saída). Ele seleciona uma premissa e uma conclusão de regra que melhor representem o dado de treino, respeitando um limite mínimo de ativação. A premissa e a conclusão não precisam necessariamente serem da mesma regra. Dependendo de quais partes foram selecionadas, ele decide por criar ou não uma nova regra, tentando compartilhar conclusões entre regras (i.e. mesma conclusão para diferentes premissas). As regras que forem criadas só serão adicionadas ao sistema na etapa 3.

Na segunda etapa do aprendizado estrutural, o eFSM realiza a fusão de conjuntos fuzzy sobrepostos. Inicialmente os conjuntos fuzzy, cuja função de pertinência são do tipo gaussiana, são aproximados por uma função triangular cujos parâmetros são: $[(\mu - 2\sigma \ln 2), (\mu), (\mu + 2\sigma \ln 2)]$.

A medida de similaridade é calculada como:

$$S(A, B) = \frac{Area(\Delta_A \cap \Delta_B)}{Area(\Delta_A)} \tag{3.16}$$

onde A, B são funções gaussianas, e Δ_A, Δ_B são as aproximações triangulares de A e B.

Os conjuntos fuzzy com similaridade maior que um dado limite são fundidos, conforme:

$$\mu^* = \frac{1}{2}\Big\{ \min\left(\mu_A - \sigma_A \cdot \sqrt{\ln 2}, \mu_B - \sigma_B \cdot \sqrt{\ln 2}\right) + \max\left(\mu_A + \sigma_A \cdot \sqrt{\ln 2}, \mu_B + \sigma_B \cdot \sqrt{\ln 2}\right) \Big\} \tag{3.17}$$

$$\sigma^* = \frac{1}{2 \cdot \sqrt{\ln 2}}\Big\{ \min\left(\xi_i^{max}, \{ \max\left(\mu_A + \sigma_A \cdot \sqrt{\ln 2}, \mu_B + \sigma_B \cdot \sqrt{\ln 2}\right) - \min\left(\mu_A - \sigma_A \cdot \sqrt{\ln 2}, \mu_B - \sigma_B \cdot \sqrt{\ln 2}\right) \}\right) \Big\} \tag{3.18}$$

onde A e B são os conjuntos a serem fundidos, (μ_i, σ_i) são o centro e o desvio do conjunto i, e (μ^*, σ^*) serão os parâmetros da nova função gaussiana.

Após a etapa de fusão de conjuntos fuzzy, é possível haver regras idênticas (redundantes), as quais são podadas na terceira etapa do aprendizado estrutural.

Caso as regras que foram criadas na primeira etapa não tenham se tornado redundantes após a etapa de fusão de conjuntos, elas são adicionadas ao sistema na quarta etapa.

Na quinta etapa, é realizada a deleção de regras obsoletas, conforme um atributo de potencial calculado para cada regra. Com a deleção de regras, podem haver conjuntos fuzzy sem uma regra atrelada, os quais também são removidos do sistema.

3.5 SÍNTESE COMPARATIVA

O Celanfis é um NFS. Ele restringe seus conjuntos (sino generalizado) a um grau de possibilidade entre 0.4 e 0.6, fundindo conjuntos com grau de possibilidade maior que 0.9, e comprimindo ou alongando os demais para o intervalo mencionado. Isso garante distinguibilidade, e cobertura de 0.4.

O Constrained PSO é um sistema evolucionário-fuzzy, com restrições que guiam o processo de otimização. Ele garante distinguibilidade via restrição no grau de possibilidade máxima entre conjuntos, e espaçamento mínimo entre os centros dos conjuntos. Garante cobertura das partíções a um nível mínimo ajustável, e garante também consistência de regras pela fusão de regras similares no antecedente.

O eFSM é um NFS incremental. Ele usa uma medida de similaridade para guiar a fusão de conjuntos, garantindo distinguibilidade pela dissimilaridade. Por ser um modelo incremental, e haver deleção de regras obsoletas e conjuntos fuzzy inutilizados, não há garantia de cobertura a um nível mínimo nítido[0] para a partição fuzzy. Mas, justamente pela obsolescência, poderia se avaliar tais valores como fora da partição fuzzy. O eFSM garante também consistência de regras.

O INFGMN é um NFS incremental que constrói suas regras a partir das componentes de um GMM, e cujos conjuntos fuzzy advém das PDFs marginais. Como tais PDFs marginais podem estar altamente sobrepostas, não garante distinguibilidade. Por ser um modelo incremental, e haver poda de componentes (regras) obsoletas, não há garantia de cobertura a um nível mínimo nítido[0] para a partição fuzzy. Mas, justamente pela obsolescência, poderia-se avaliar tais valores como fora da partição fuzzy. Mesmo sem distinguibilidade, pode-se evitar a criação de componentes sobrepostas por meio do parâmetro τ_{nov}, que conduz a criação de novas componentes, e assim gerar uma base de regras consistente. Mas, caso haja dados inconsistentes no conjunto de treino, pode gerar inconsistência.

A tabela 6 faz um comparativo entre os critérios de interpretabilidade alcançados por cada um dos trabalhos relacionados.

---

[0] Funções gaussianas possuem suporte infinito, mas a pertinência tende a 0.

[1] Constrained PSO

Tabela 6 – Tabela comparativa

Critérios	Celanfis	C-PSO[1]	eFSM	INFGMN	C-INFGMN
Distinguibilidade	+	+	+	-	+
Cobertura	+	+	+/-	+/-	+/-
Consistência nas regras	+	+	+	+/-	+

Fonte – Elaborado pelo autor.

4 INCREMENTAL NEURO-FUZZY GAUSSIAN MIXTURE NETWORK

4.1 MODELO DE MISTURA DE GAUSSIANAS

Um Modelo de Mistura de Gaussianas (GMM) é uma Função Densidade de Probabilidade (PDF) representada pela soma ponderada de suas componentes gaussianas. (MAZZUTTI *et al.*, 2018)

Sua equação é:

$$G(x; p, \mu, C) = \sum_{j=1}^{J} p(j) N^D(x; \mu_j, C_j) \tag{4.1}$$

tal que:

$$N^D(x; \mu_j, C_j) = \frac{exp\{-\frac{1}{2}(x-\mu_j)^T C_j^{-1}(x-\mu_j)\}}{\sqrt{(2\pi)^D \mid C_j \mid}} \tag{4.2}$$

$$0 \leq p(j) \leq 1, \forall j \tag{4.3}$$

$$\sum_{j=1}^{J} p(j) = 1 \tag{4.4}$$

onde:

D é a dimensão do modelo;

J é o número de componentes gaussianas no modelo;

x é o vetor D-dimensional de entrada;

p é o vetor J-dimensional de proporções;

$p(j)$ é a proporção da j-ésima componente gaussiana;

μ_j é o vetor D-dimensional de médias da j-ésima componente;

μ é a matriz $(D \times J)$ dos vetores de médias concatenados;

C_j é a matriz $(D \times D)$ de covariância da j-ésima componente;

C é a matriz $(D \times DJ)$ das matrizes de covariância concatenados;

$N^D(x; \mu_j, C_j)$ é a função densidade normal multivariada (D-dimensional) da j-ésima componente;

4.2 MODELO DE MISTURA DE GAUSSIANAS INCREMENTAL

Segundo Engel e Heinen (2010), o Modelo de Mistura de Gaussianas Incremental (IGMM) foi projetado para aprender um GMM a partir de um fluxo de dados de maneira incremental e não-supervisionada. Ele consegue lidar com o dilema estabilidade-plasticidade, i.e, se um novo dado deve ser assimilado pelo modelo, ou causar uma mudança estrutural para acomodar a nova informação. Com isso, novas componentes são adicionadas à mistura sob demanda, através de um critério de novidade.

O IGMM contém 4 parâmetros: σ_{ini}, τ_{nov}, t_{max} e sp_{min}. O σ_{ini} é um parâmetro de configuração da variância inicial de cada componente, e não é crítico para o modelo. Recomenda-se uma fração da variância total de cada dimensão, e.g. $\sigma_{ini} = (x_{max} - x_{min})/10$. Por outro lado, τ_{nov}, o critério de novidade, define o quão distante um vetor tem de estar de cada componente para a criação de uma nova, e é um parâmetro crítico. Se $\tau_{nov} = 1$, uma nova componente será criada para cada novo dado, e caso $\tau_{nov} = 0$, a mistura será composta por uma única componente. Adicionalmente, os parâmetros t_{max} e sp_{min} definem quando uma componente não é mais necessária para o modelo, ocorrendo sua remoção quando $t_j > t_{max}$ e $sp_j < sp_{min}$, sendo sp_j o número de instâncias assimiladas pela componente j, e t_j o tempo que ela levou para isso. (ENGEL; HEINEN, 2010; MAZZUTTI *et al.*, 2018)

As probabilidades a priori, $p(j)$ (proporção da componente), são ajustadas e normalizadas para satisfazer as equações 4.3 e 4.4 de uma GMM, e adicionalmente a seguinte equação (MAZZUTTI *et al.*, 2018):

$$\int_{-\infty}^{+\infty} p(x|j)dx = 1 \tag{4.5}$$

onde $p(x|j)$ é a probabilidade de se observar um vetor x pertencente à componente j,e é computada como a PDF de j:

$$p(x|j) = N^D(x; \mu_j, C_j) \tag{4.6}$$

Um vetor x é reconhecido como não pertencente a nenhuma componente j caso:

$$p(x|j) < \frac{\tau_{nov}}{(2\pi)^D \mid C_j \mid}, \forall j \tag{4.7}$$

Nesse caso, uma nova componente com média $\mu_j^* = x$ e matriz de covariância $C_j^* = \sigma_{ini}$ é adicionada à mistura.

Para a assimilação de novos dados, o IGMM segue uma versão incremental do processo iterativo de 2 passos do algoritmo Estimation/Maximization (EM).

No passo E, estima-se a probabilidade de pertinência de um vetor x (considerado pertencente à mistura) a cada uma das componentes, conforme o Teorema de Bayes:

$$p(j|x) = \frac{p(x|j)p(j)}{\sum_{j=1}^{J} p(x|j)p(j)}, \forall j \tag{4.8}$$

No passo M, os parâmetros de cada componente são ajustados para assimilarem o novo dado, conforme a seguinte equação recursiva:

$$sp_j^* = sp_j + p(j|x) \tag{4.9}$$

$$\mu_j^* = \mu_j + \frac{p(j|x)}{sp_j^*}(x - \mu_j) \tag{4.10}$$

$$\begin{aligned} C_j^* =& C_j - (\mu_j^* - \mu_j)(\mu_j^* - \mu_j)^T \\ &+ \frac{p(j|x)}{sp_j^*}\left[(x - \mu_j^*)(x - \mu_j^*)^T - C_j\right] \end{aligned} \tag{4.11}$$

$$p(j)^* = \frac{sp_j^*}{\sum_{q=1}^{J} sp_q^*} \tag{4.12}$$

onde $sp_j^*, \mu_j^*, C_j^*, p(j)^*$ são os novos valores de $sp_j, \mu_j, C_j, p(j)$, respectivamente.

4.3 EQUIVALÊNCIA GMM E FIS

Segundo Gan *et al.* (2005), sistemas fuzzy aditivos podem ser divididos em 3 tipos: Mamdani-Larsen (ML), Takagi-Sugeno (TS) e Modelo Fuzzy Generalizado (GFM). Enquanto possuem a mesma forma no antecedente (termos representados por conjuntos fuzzy), eles distinguem-se no consequente:

$$\text{ML:} \quad R^k: \text{ IF } x_1 \text{ is } A_1^k \wedge \cdots \wedge x_D \text{ is } A_D^k \text{ THEN } y \text{ is } B^k(b_k, v_k) \tag{4.13}$$

$$\text{TS:} \quad R^k: \text{ IF } x_1 \text{ is } A_1^k \wedge \cdots \wedge x_D \text{ is } A_D^k \text{ THEN } y \text{ is } f^k(x) \tag{4.14}$$

$$\text{GFM:} \quad R^k: \text{ IF } x_1 \text{ is } A_1^k \wedge \cdots \wedge x_D \text{ is } A_D^k \text{ THEN } y \text{ is } B^k(f^k(x), v_k) \tag{4.15}$$

onde A_i^k é um subconjunto fuzzy da i-ésima variável de entrada x_i na k-ésima regra R^k, B^k é um subconjunto fuzzy de saída e v_k é o peso da k-ésima regra.

No consequente do modelo de ML, $B^k(b_k, v_k)$ é um conjunto fuzzy com centróide b_k e área v_k, calculados pelas fórmulas:

$$v_k = \int_y \phi^k(y)dy \tag{4.16}$$

$$b_k = \frac{\int_y y\phi^k(y)dy}{\int_y \phi^k(y)dy} \tag{4.17}$$

onde ϕ^k é a função de pertinência de B^k.

No consequente do modelo de TS, $f^k(x)$ é uma função linear ou não-linear que descreve a relação entrada-saída, onde uma função linear teria a forma:

$$f^k(x) = b_{k0} + x_1 b_{k1} + \cdots + x_D b_{kD} \tag{4.18}$$

E, segundo Gan *et al.* (2005), combinando as propriedades de ML e TS, Azeem et al. propôs o GFM como um conjunto fuzzy com centróide variável na forma $B^k(f^k(x), v_k)$.

Para modelos fuzzy com agregação aditiva, e conjunção e implicação multiplicativas (condições para a equivalência entre GMM e GFM), as fórmulas para obter as saídas defuzzificadas de ML, TS e GFM são, respectivamente (GAN *et al.*, 2005):

$$y^o = \sum_{k=1}^{K} \frac{\lambda^k(x) v_k}{\sum_{k'=1}^{K} \lambda^{k'}(x) v_{k'}} b_k \tag{4.19}$$

$$y^o = \sum_{k=1}^{K} \frac{\lambda^k(x)}{\sum_{k'=1}^{K} \lambda^{k'}(x)} f^k(x) \tag{4.20}$$

$$y^o = \sum_{k=1}^{K} \frac{\lambda^k(x) v_k}{\sum_{k'=1}^{K} \lambda^{k'}(x) v_{k'}} f^k(x) \tag{4.21}$$

4.3.1 GMM para GFM

Considerando um caso *Multi Input Single Output* (MISO) com vetor de entrada x e saída y, Gan *et al.* (2005) demonstraram que a fórmula geral da mistura de gaussianas pode ser reescrita como:

$$G(x, y) = \sum_{j=1}^{J} p(j) N^{D+1} \left(\begin{bmatrix} x \\ y \end{bmatrix} ; \begin{bmatrix} \mu_{j,x} \\ \mu_{j,y} \end{bmatrix}, C_j \right) \tag{4.22}$$

onde

$$C_j = \begin{bmatrix} \{\sigma_{j,id}\}_{D \times D} & \{\sigma_{j,i(D+1)}\}_{D \times 1} \\ \{\sigma_{j,(D+1)d}\}_{1 \times D} & \{\sigma_{j,(D+1)(D+1)}\}_{1 \times 1} \end{bmatrix} \tag{4.23}$$

para $i = d = 1, 2, \ldots, D$, $\mu_{j,x} = [\mu_{j,1} \cdots \mu_{j,D}]^T$ e $\mu_{j,y} = \mu_{j,(D+1)}$.

Gan *et al.* (2005) derivaram a PDF marginal de x como:

$$G(x) = \int_{-\infty}^{+\infty} G(x, y) dy \tag{4.24}$$

$$= \sum_{j=1}^{J} \int_{-\infty}^{+\infty} p(j) N^{D+1} \left(\begin{bmatrix} x \\ y \end{bmatrix} ; \begin{bmatrix} \mu_{j,x} \\ \mu_{j,y} \end{bmatrix}, C_j \right) dy \tag{4.25}$$

$$= \sum_{j=1}^{J} p(j) N^{D}(x; \mu_{j,x}, \sigma_{j,xx}) \tag{4.26}$$

onde $\sigma_{j,xx}$ é o menor (minor) da matriz de covariâncias, C, após eliminar a (J+1)-ésima linha e (J+1)-ésima coluna. Então, geraram a PDF condicional de y como:

$$G(y \mid x) = \frac{\sum_{j=1}^{J} p(j) N^{D+1} \left(\begin{bmatrix} x \\ y \end{bmatrix} ; \begin{bmatrix} \mu_{j,x} \\ \mu_{j,y} \end{bmatrix}, C \right)}{\sum_{j'=1}^{J} p(j') N^{D}(x; \mu_{j',x}, \sigma_{j',xx})} \tag{4.27}$$

Gan *et al.* (2005) derivaram a esperança condicional de y como:

$$E[y|x] = \frac{\sum_{j=1}^{J} p(j) \prod_{d=1}^{D} N^{D}(x_d; \mu_{j,d}, \sigma_{j,dd}) \left(\mu_{j,y} - \frac{[x - \mu_{j,x}]' \sigma_j^{xy}}{\sigma_j^{yy}} \right)}{\sum_{j'=1}^{J} p(j') \prod_{d'=1}^{D} N^{D}(x_d; \mu_{j',d}, \sigma_{j',dd})} \tag{4.28}$$

onde $\sigma_{j,xx}$ é uma partição da matriz de covariância, C, enquanto σ_j^{xy} e σ_j^{yy} são partições de sua inversa.

Assim, Gan *et al.* (2005) demonstraram que as equações 4.21 e 4.28 são equivalentes sob as seguintes condições:

i O número de regras, K, na base de regras do GFM é igual ao número de componentes, J, no GMM, i.e.:

$$K = J$$

ii O peso de cada regra, v_k, é dado pela correspondente probabilidade a priori, $p(j)$, do modelo de mistura, i.e.:

$$v_k = p(j)$$

iii A função de regressão na parte THEN das regras do GFM é linear, e é uma função de todas as variáveis de entrada, dada por:

$$f^k(x) = \mu_{j,y} - \frac{[x - \mu_{j,x}]^t \sigma_j^{xy}}{\sigma_j^{yy}}$$

iv A função de pertinência de cada uma das variáveis da parte IF é uma função gaussiana, i.e.:

$$\lambda^k(x_d) = N^1(x_d; \mu_{j,d}, \sigma_{j,dd})$$

para $d = 1, 2, \cdots, D$.

v O sistema fuzzy é aditivo, com conjunção e implicação multiplicativas.

4.3.2 GFM para ML e TS

A fórmula de defuzzificação de ML, representada na equação 4.19, é igual a de GFM, representada em 4.21, quando a função de regressão é reduzida a uma constante, i.e, $f^k(x) = b_k$. Isso é obtido ao zerar o vetor de covariâncias entrada-saída $\sigma_{k,xy}$, consequentemente zerando σ_k^{xy} e eliminando as variáveis x da função. Assim, para o caso especial de ML, a condição 3 é alterada para $f^k(x) = \lambda_{j,y}$.

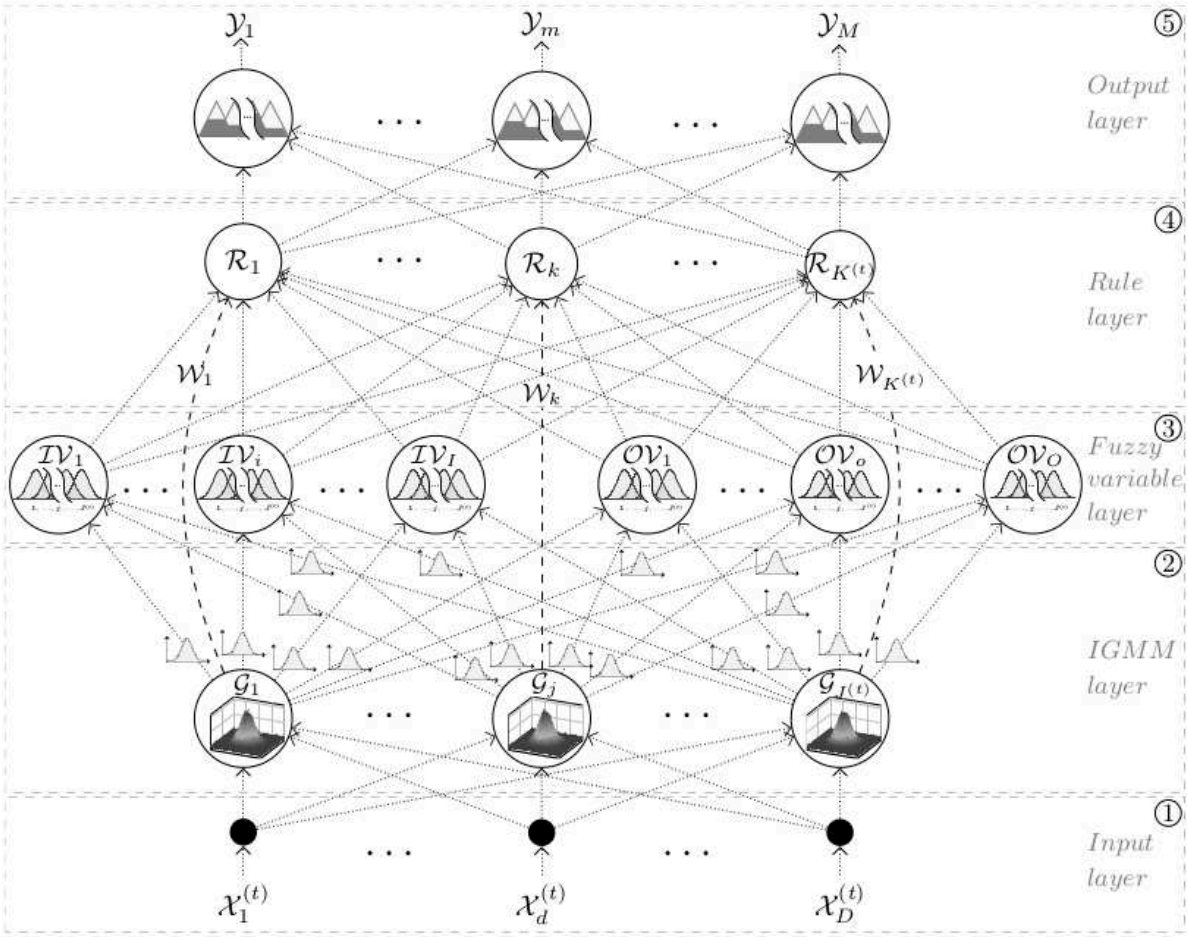
> "Corollary 1: If the input–output relationship of a system follows a Gaussian mixture pdf and if its inputs and output are mutually independent of each other, then the GFM (...) can be reduced to a ML model such that for the k-th rule, the regression function in the THEN-part is a constant, given by $f^k(x) = \lambda_{k,y}$, which is the Gaussian mean of the output. All parts (...), except part iii), remain unchanged."(GAN *et al.*, 2005).

A fórmula de defuzzificação de TS, representada na equação 4.20, é igual a de GFM, representada em 4.21, quando as proporções das componentes (probabilidades a priori) são iguais. Assim, para o caso especial de TS, sem violar as restrições das proporções de mistura, a condição 2 é alterada para $v_k = \frac{1}{K}$.

> "Corollary 2: If the input–output relationship of a system follows a Gaussian mixture pdf of equal priors, with each prior being 1/K, where K is the number of Gaussian components, then the GFM (...) can be reduced to a TS model with equal weights (or unweighted TS model). Barring part ii), all other parts (...) remain unchanged."(GAN *et al.*, 2005).

4.4 ARQUITETURA INFGMN

Figura 6 – Arquitetura da rede INFGMN.



Fonte – Mazzutti *et al.* (2018)

Mazzutti *et al.* (2018) descrevem que o *Incremental Neuro-Fuzzy Gaussian Mixture Network* (INFGMN) é um NFS que se utiliza da equivalência entre GMM e Mamdani-Larsen (ML), e do IGMM para fornecer a capacidade de aprendizado incremental. Ele possui 5 camadas de rede, e cada uma desempenha um papel bem específico. As notações na figura 6 são definidas como:

D é a dimensão do vetor de entrada;

M é a dimensão do vetor de saída;

X_d é a d-ésima componente do vetor de entrada;

Y_m é o valor de m-ésima dimensão na saída;

G_j é a j-ésima componente de mistura gaussiana na camada IGMM;

IV_i é a i-ésima variável linguística fuzzy de entrada na camada de variáveis fuzzy;

OV_i é a o-ésima variável linguística fuzzy de saída na camada de variáveis fuzzy;

R_k é a k-ésima regra fuzzy na camada de regras;

W_k é o k-ésimo peso de regra fuzzy; e

$X^{(t)}$, $J^{(t)}$ e $K^{(t)}$ representam o estado dos respectivos vetores no tempo t.

O INFGMN possui 2 modos de operação chamados *learning* e *recalling*, denotados respectivamente por f_L e f_R.

Na camada ① (Input layer) são apresentados as variáveis de entradas da rede (cada nó representa uma entrada), e estas são repassadas a camada ② sem alteração. (MAZZUTTI *et al.*, 2018)

$$f^{①}(X_d^{(t)}) = X_d^{(t)}, \forall d \in \{1, .., D\} \tag{4.29}$$

Na camada ② (IGMM layer) são mantidas as componentes gaussianas. Ela cria e ajusta os parâmetros de cada componente de acordo com as equações 4.8 a 4.12 do IGMM. O modo de operação *learning* pode ser representado por

$$f^{②}(G_j(X^{(t-1)}; p, \mu, C), X^{(t)}) = G_j(X^{(t)}; p^*, \mu^*, C^*), \forall j \in \{1..J^{(t)}\} \tag{4.30}$$

onde $G_j(X^{(t-1)}; p, \mu, C)$ é a j-ésima componente de mistura gaussiana, atualizada até o tempo $t-1$, e $G_j(X^{(t)}; p^*, \mu^*, C^*)$ é a componente atualizada após a apresentação do vetor de entrada $X^{(t)}$. As médias (μ^*) e as covariâncias (C^*) atualizadas são transmitidas para a camada ③ , e as probabilidades a priori (p^*) atualizadas são transmitidas diretamente a camada ④ como vetor de pesos W. Esta camada não é ativada no modo recalling. (MAZZUTTI *et al.*, 2018)

Na camada ③ (Fuzzy variable layer) são mantidas as variáveis linguísticas de entrada (IV_i) e saída (OV_o), e seus respectivos termos linguísticos $IV_{i(j)}$ e $OV_{o(j)}$, para $j = 1..J^{(t)}$. Cada termo linguístico, representado por $IV_{i(j)}(\mu_{i(j)}, C_{i(j)})$, ou $OV_{o(j)}(\mu_{o(j)}, C_{o(j)})$, possui função de pertinência gaussiana cujos parâmetros correspondem à PDF marginal da j-ésima componente na dimensão i ou o. Esta camada é ativada em ambos os modos de operação (learning e recalling). (MAZZUTTI *et al.*, 2018)

No modo learning, termos linguísticos podem ser criados, removidos e/ou atualizados, e seus parâmetros são ajustados de acordo com a média e covariância das componentes geradas na camada ② . (MAZZUTTI *et al.*, 2018)

$$f^{③}_{L(I)}(\mu, C) = \Big\{ \big\{ IV_{i(j)}(\mu_{i(j)}, C_{i(j)}) \mid j \in \{1..J^{(t)}\} \big\} \mid i \in \{1..I\} \Big\} \tag{4.31}$$

$$f^{③}_{L(O)}(\mu, C) = \Big\{ \big\{ OV_{o(j)}(\mu_{o(j)}, C_{o(j)}) \mid j \in \{1..J^{(t)}\} \big\} \mid o \in \{1..O\} \Big\} \tag{4.32}$$

No modo recalling, esta camada é responsável por fuzzificar as variáveis de entrada, onde $\lambda^{(IV_i)}$ é a função de pertinência da i-ésima variável linguística de entrada, $\phi^{(OV_o)}$ é a função de pertinência da i-ésima variável linguística de saída, e N^1 é a função gaussiana correspondente. (MAZZUTTI *et al.*, 2018)

$$f^{③}_{R(I)}(X, IV) = \Big\{ \lambda^{(IV_i)} : \big\{ N^1(X_i, \mu_{i(j)}, C_{i(j)}) \mid j \in \{1..J^{(t)}\} \big\} \mid i \in \{1..I\} \Big\} \tag{4.33}$$

$$f^{③}_{R(O)}(OV) = \Big\{ \phi^{(OV_o)} : \Big\{ \int_y N^1(y, \mu_{o(j)}, C_{o(j)}) dy \mid j \in \{1..J^{(t)}\} \Big\} \mid o \in \{1..O\} \Big\} \tag{4.34}$$

Na camada ④ (Rule layer) são mantidas as regras fuzzy do tipo Mamdani-Larsen. No modo learning, regras podem ser criadas, removidas e/ou atualizadas de acordo com as componentes gaussianas, onde o número de regras $K^{(t)}$ é igual ao número de componentes de mistura $J^{(t)}$. Cada regra R^k é moldada com antecedentes ($IV_{i(k)}$) e consequentes ($OV_{o(k)}$) cujos termos linguísticos, gerados na camada ③ , vieram da j-ésima componente gaussiana (onde $j = k$), cuja proporção, $p(j)$, será o peso da regra, W_k. (MAZZUTTI *et al.*, 2018)

$$\begin{aligned} f^{④}_{L}(IV, OV, W) = \Big\{ R^k : \ & \text{IF } \wedge_{i=1}^{I} X_i \text{ is } IV_{i(k)} \text{ THEN} \\ & \big[OV^{*m} : \vee_{o=1}^{O} Y_m \text{ is } OV_{o(k)} \big]_{m=1}^{M} (W_k) \\ & \mid k \in \{1..K^{(t)}\}, K^{(t)} = J^{(t)} \text{ e } M = O \Big\} \end{aligned} \tag{4.35}$$

No modo recalling, o grau de ativação de cada regra, α_k, que corresponde a seu antecedente, é computado como uma conjunção multiplicativa dos graus de pertinência de cada antecedente da regra, e então multiplicado pelo peso da regra, W_k. Com o grau de ativação, calcula-se a implicação multiplicativa para o consequente, e gera-se um conjunto fuzzy de saída para a regra. Os conjuntos fuzzy de saída são então agregados utilizando a agregação aditiva, gerando um único conjunto fuzzy para a

variável de saída. (MAZZUTTI *et al.*, 2018)

$$f_R^{④}\left(\lambda^{IV}, \phi^{OV*}\right) = \left\{ \alpha_k : W_k \cdot \prod_{i=1}^{I} \lambda^{IV_{i(k)}} \mid k \in \{1..K^{(t)}\} \right\}$$
$$\Rightarrow \left\{ \phi^{*m} : \sum_{k=1}^{K(t)} \alpha \cdot \phi^{OV^{*m}_{(k)}} \mid m \in \{1..M\}, K^{(t)} = J^{(t)} e M = O \right\} \quad (4.36)$$

A camada ⑤ (Output layer), ativada somente no modo recalling, é responsável por defuzzificar o conjunto fuzzy de saída utilizando-se do método centróide. (MAZZUTTI *et al.*, 2018)

$$f_R^{⑤}\left(\phi^*\right) = \left\{ \frac{\int_y y \phi^{*m}(y) dy}{\int_y \phi^{*m}(y)} dy \mid m \in \{1..M\} \right\} \quad (4.37)$$

5 PROPOSTA

Neste trabalho buscou-se expandir as funcionalidades do sistema INFGMN, proporcinando sua aplicação com modelos de Takagi-Sugeno e melhorando sua interpretabilidade via distinguibilidade nas partições fuzzy. Com isso, houveram mudanças arquiteturais nas camadas ③ , ④ e ⑤ .

5.1 ARQUITETURA

Camada ③ (Fuzzy Variable Layer): Esta camada é ativada tanto para o modo de operação learning quanto para o modo recalling. Esta camada é composta pelas variáveis linguísticas fuzzy de entrada IV_i, para $i = 1, ..., I$, e pelas variáveis de saída OV_o, para $o = 1, ..., O$, onde cada variável linguística IV_i é formada por um conjunto de $IV_{i_{(k_i)}}$ termos linguísticos fuzzy, para $k_i = 1, ..., K_i$, onde $K_i \leq \min(maxMFs, J)$, e cada variável OV_o é formada por um conjunto de $OV_{o_{(j)}}$ funções lineares, para $j = 1, ..., J^{(t)}$.

No modo de operação learning, os antecedentes e os consequentes de cada regra fuzzy (IF-THEN) são atualizados em cada etapa de treinamento t, após cada padrão de treino contido em $X^{(t)}$.

Os antecedentes, definidos pelos termos linguísticos fuzzy das variáveis de entrada, tem seu significado representado por conjuntos fuzzy com função de pertinência Gaussiana. Seus parâmetros $\mu_{i(k_i)}$ e $C_{i(k_i)}$, referenciados na equação 5.8, são definidos pelo processo de fusão e separação de conjuntos fuzzy, conforme as equações 5.1 a 5.5, onde $\alpha_{\text{-cut}(M)}$ é a união dos $\alpha_{\text{-cut}}$ dos conjuntos fuzzy fundidos, similar ao método de Tung e Quek (2009), e $\alpha_{\text{-cut}(M_S)}$ o intervalo resultante após o processo de separação dos conjuntos fuzzy, de forma a não se sobreporem aos núcleos dos demais conjuntos em um nível de pertinência maior que 0.5.

$$\alpha = 0.5 \tag{5.1}$$

$$\alpha_{\text{-cut}(M)} = \left[\min_{j \in M}\left(\mu_{i(j)} - C_{i(j)} \cdot \sqrt{-2\ln\alpha}\right), \max_{j \in M}\left(\mu_{i(j)} + C_{i(j)} \cdot \sqrt{-2\ln\alpha}\right)\right] \tag{5.2}$$

$$\alpha_{\text{-cut}(M_S)} = \left[\max\left(\min(\alpha_{\text{-cut}(M)}), \max_{j \in M-1}(\mu_{i(j)})\right), \min\left(\max(\alpha_{\text{-cut}(M)}), \min_{j \in M+1}(\mu_{i(j)})\right)\right] \tag{5.3}$$

$$\mu_{i(k_{i(M)})} = \frac{\max(\alpha_{\text{-cut}(M_S)}) + \min(\alpha_{\text{-cut}(M_S)})}{2} \tag{5.4}$$

$$C_{i(k_{i(M)})} = \frac{\max(\alpha_{\text{-cut}(M_S)}) - \min(\alpha_{\text{-cut}(M_S)})}{2 \cdot \sqrt{-2\ln\alpha}} \tag{5.5}$$

O processo de fusão começa por aqueles que forneçam a maior similaridade para a partição fuzzy do espaço de entrada, prosseguindo enquanto a fusão produzir uma partição mais similar que a separação, enquanto a similaridade da partição estiver acima de $simMerge$, e até que o número máximo de conjuntos fuzzy, $maxMFs$,

seja satisfeito. A similaridade da partição fuzzy do espaço de entrada $IV_{(i)}$ é calculada conforme a equação 5.6, onde $S(A, B)$ é a similaridade entre os conjuntos A e B, conforme as equações 3.3 a 3.9 de Hefny (2007), $\lambda_{i(j)}$ é o conjunto original da componente gaussiana, $\lambda_{i(k_{i(j)})}$ é seu conjunto aproximado pela fusão/separação, e W_j é o peso da componente.

$$S(IV_{(i)}) = \prod_{j=1}^{J^{(t)}} S(\lambda_{i(k_{i(j)})}, \lambda_{i(j)})^{W_j \cdot Z_j} \tag{5.6}$$

$$Z_j = \left(\frac{1}{S(\lambda_{i(k_{i(j)})}, \lambda_{i(j)}))} \right)^2 \tag{5.7}$$

Cada j-ésimo termo é ponderado pelo peso da respectiva componente, W_j, e pelo termo Z_j, que agrava os termos com similaridade menores.

Os consequentes são definidos por funções lineares de saída, e seus coeficientes serão aprendidos conforme a equação 5.9, onde $A_{o(j)}$ e $L_{o(j)}$ são os coeficientes angular e linear, respectivamente, e $C^{Io}_{(j)}$ e $C^{oo}_{(j)}$ são partições da inversa da matriz de covariância.

$$f^{③}_{L(I)}(\mu, C) = \Big\{ \Big\{ IV_{i(k_i)}(\mu_{i(k_{i(M)})}, C_{i(k_{i(M)})}) \\ \mid k_i \in \{1..K_i\} \wedge K_i \leq \min(maxMFs, J) \Big\} \mid i \in \{1..I\} \Big\} \tag{5.8}$$

$$f^{③}_{L(O)}(\mu, C) = \Big\{ \{ OV_{o(j)}([A^T_{o(j)}, L_{o(j)}]) \mid j \in \{1..J^{(t)}\} \\ \wedge A_{o(j)} = -\frac{C^{Io}_{(j)}}{C^{oo}_{(j)}} \wedge L_{o(j)} = \mu_{o(j)} + \mu^T_{I(j)} A_{o(j)} \Big\} \mid o \in \{1..O\} \Big\} \tag{5.9}$$

Para o modo de operação recalling, esta camada tem a responsabilidade de fuzzificar os dados do vetor de entrada X e calcular o resultado de cada função linear de saída. Esta etapa está representada nas Equações 5.10 e 5.11, onde $\lambda^{(IV_i)}$ é a função de pertinência fuzzy da i-ésima variável linguística de entrada, $\phi^{(OV_o)}$ é a função linear crisp da o-ésima variável de saída, e N^1 é uma distribuição normal 1-Dimensional,

$$f^{③}_{R(I)}(X, IV) = \Big\{ \lambda^{(IV_i)} : \Big\{ N^1(X_i, \mu_{i(k_i)}, C_{i(k_i)}) \\ \mid k_i \in \{1..K_i\} \wedge K_i \leq \min(maxMFs, J) \Big\} \mid i \in \{1..I\} \Big\} \tag{5.10}$$

$$f^{③}_{R(O)}(X, OV) = \big\{ \phi^{(OV_o)} : \big\{ f_{o(j)}(X) = L_{o(j)} + X \cdot A_{o(j)} \mid j \in \{1..J^{(t)}\} \big\} \mid o \in \{1..O\} \big\} \tag{5.11}$$

Camada ④ (Rule layer): Uma vez que o antecedente e consequente foram gerados na camada ③ , as regras fuzzy podem ser atualizadas. Esta camada é ativada para ambos os modos de operação, learning e recalling.

Durante o modo de operação learning, as regras fuzzy da INFGMN são atualizadas de acordo com a Equação 5.12, onde o argumento OV são funções lineares, o número de antecedentes da parte IF de cada regra é igual a I e o número de consequentes da parte THEN de cada regra é igual a O. Regras inconsistentes (com os mesmos antecendentes) serão fundidas, com o novo consequente definido como $OV_{o(k(M))}$ e o novo peso da regra como $W_{k(M)}$, onde $p(j)$ é a probabilidade à priori da j-ésima componente, e M contém os índices das respectivas componentes cujos antecendentes foram fundidos em $IV_{i(k_{i(M)})}$. O número $K^{(t)}$ de regras geradas, então, é igual ou menor ao número $J^{(t)}$ de componentes de mistura encontrados na camada ② (IGMM Layer) no tempo t.

$$f_L^{④}\left(IV, OV, W\right) = \Bigg\{ R^k : \text{ IF } \bigwedge_{i=1}^{I} X_i \text{ is } IV_{i(k(M))} \text{ THEN } \bigvee_{o=1}^{O} Y_o \text{ is } OV_{o(k(M))} \quad (W_{k(M)}) \mid k \in \{1..K^{(t)}\} \wedge K^{(t)} \leq J^{(t)} \wedge W_{k(M)} = \sum_{j \in M} p(j) \wedge OV_{o(k(M))} = \frac{\sum_{j \in M} p(j) \cdot OV_{o(j)}}{W_{k(M)}} \Bigg\} \quad (5.12)$$

Para o modo de operação recalling, a operação representada pela Equação 5.13, cujo argumento ϕ^{OV} são conjuntos *singleton*, é usada para avaliar as regras fuzzy da INFGMN da sequinte maneira: I) O conjunto de graus de ativação α_k,correspondente à parte antecedente da implicação $f_R^{④}$ de cada regra, é calculado através da aplicação de uma operação de conjunção fuzzy a cada antecedente (parte IF) da regra. Este grau de ativação é então multiplicado pelo peso da regra, W_k. II) A implicação fuzzy é calculada para a parte consequente da função $f_R^{④}$. III) A agregação da parte THEN de todas as regras que foram ativadas é feita através de uma agregação fuzzy aditiva, resultando no conjunto fuzzy ϕ^{*o}, contendo todos os conjuntos fuzzy agregados, para todas as saídas o=1,...,O, geradas pela parte consequente da função $f_R^{④}$ definida na Equação 5.13.

$$f_R^{④}\left(\lambda^{(IV)}, \phi^{(OV)}\right) = \left\{ \alpha_k : W_{k(M)} \cdot \prod_{i=1}^{I} \lambda^{(IV_{i(k(M))})} \mid k \in \{1..K^{(t)}\} \wedge K^{(t)} \leq J^{(t)} \right\} \Longrightarrow \left\{ \phi^{*o} : \sum_{k=1}^{K} \alpha_k \cdot \phi^{(OV_{o(k(M))})} \mid o \in \{1..O\} \right\} \quad (5.13)$$

Camada ⑤: Esta camada é ativada somente durante o modo de operação *recalling* e é responsável por *defuzzificar* a saída da INFGMN, conforme o método *Weighted Average* na equação $f_R^{⑤}$, onde:

α_k é o grau de ativação da k-ésima regra.

ϕ^{*o} é o conjunto fuzzy resultante após o processo de agregação na m-ésima variável de saída.

$$f_R^{⑤}(\phi^*, \alpha) = \left\{ \frac{\phi^{*o}}{\sum_{k=1}^{K} \alpha_k} \mid o \in \{1..O\} \right\} \tag{5.14}$$

5.2 MODO DE OPERAÇÃO LEARNING

No modo de operação learning, apenas as camadas de ① a ④ são ativadas. Ao final da operação é gerado um modelo FIS de Takagi-Sugeno com distinguibilidade na partição fuzzy (dependendo do valor do parâmetro $doMerge$). A C-INFGMN tem 8 parâmetros de configuração:

τ_{nov} - controla quando um novo X_d não é reconhecido como pertencente a qualquer componente de mistura j. Se sua probabilidade $p(X_d|j)$ é menor que τ_{nov}, então ele não é considerado um membro de j. Quando X_d é rejeitado por todas as componentes de densidade, uma nova componente é adicionada ao modelo, ajustando-se adequadamente seus parâmetros.

t_{max} e sp_{min} - para cada componente j é dado algum tempo t_{max} para que mostre sua utilidade/relevância para o modelo em termos da acumulação de sua probabilidade à posteriori sp_j. Uma componente j é removido sempre que $t_j > t_{max}$ e $sp_j < sp_{min}$.

δ - valor a ser usado como uma fração da variância total do domínio estimado a partir das características envolvidas no problema (variáveis) e passadas no vetor de entrada. Este parâmetro interfere no tamanho inicial (spread) das novas componentes de mistura Gaussiana adicionados ao modelo;

$doMerge$ - valor booleano que define se a INFGMN irá operar com distinguibilidade na partição.

$simMerge$ - define o nível de similaridade aceitável para a partição fuzzy. O processo de fusão prossegue enquanto a similaridade da partição fuzzy estiver acima desse valor.

$simRefit$ - partições subsequentes no tempo são normalmente similares e não precisam ser remodeladas, senão apenas atualizados os parâmetros dos conjuntos fundidos conforme as mudanças nos originais. Mas conforme novos dados são assimilados pelo modelo, a partição fuzzy distinguível pode perder similaridade com a partição original. Esse parâmetro define o momento em que a partição deve ser renovada, recomeçando o processo de fusão e separação do zero quando a similaridade da partição fuzzy cair para um valor menor que $simRefit$.

$maxMF$ - define o número máximo de conjuntos fuzzy por partição. Esse parâmetro apenas força a fusão de conjuntos até satisfazer um máximo, independente da similaridade. Pode ser útil para encontrar a melhor solução no espaço de busca, com esse parâmetro como restrição.

O algoritmo apresentado a seguir sumariza os passos executados pela rede C-INFGMN durante sua execução em modo de operação *learning*.

Algorithm 5.1 Pseudocódigo do Modo de Operação learning

function INFGMNLEARNING(X_D)
/* Computa a verossimilhança para todos os neurônios da camada ② */
  for all neuronio j **in** Camada ② **do**
    $p(X_D \mid j) = \frac{\exp\left\{-\frac{1}{2}(X_D-\mu_j)'C_j^{-1}(X_D-\mu_j)\right\}}{\sqrt{(2\pi)^D|C_j|}}$
  end for
/* Cria um novo neurônio j na Camada ② se necessário */
  if $J < 1$ **or** $p(X_D \mid j) < \frac{\tau_{nov}}{\sqrt{(2\pi)^D|C_j|}}$ **then**
    $J^* = J + 1; t_j = 0; sp_j = 1.0;$
    $\mu_j = X_D; C_j = C_{ini}I$
    $p(X_D \mid j) = \frac{\exp\left\{-\frac{1}{2}(X_D-\mu_j)'C_j^{-1}(X_D-\mu_j)\right\}}{\sqrt{(2\pi)^D|C_j|}}$
    $p(j) = sp_j / \sum_{q=1}^{J*} sp_q, \forall j$
  end if
/* Computa as probabilidades a posteriori na Camada ② */
  $p(j \mid X_D) = \frac{p(X_D|j)p(j)}{\sum_{q=1}^{J} p(X_D|q)p(q)}, \forall j$
/* Atualiza todos os neurônios da camada ② */
  for all neuronio j **in** Camada ② **do**
    if $p(X_D \mid j) < \frac{\tau_{nov}}{\sqrt{(2\pi)^D|C_j|}}$ **then**
      $sp_j^* = sp_j + p(j \mid X_D)$
      $w_j = p(j \mid X_D)/sp_j^*$
      $\mu_j^{k*} = \mu_j^k + w_j(X_D - \mu_j^k), \forall k$
      $C_j^{k*} = C_j^k - (\mu_j^{k*} - \mu_j^k)(\mu_j^{k*} - \mu_j^k)' + w_j[(x - \mu_j^{k*})(x - \mu_j^{k*})' - C_j^k], \forall k$
      $p(j)^* = \frac{sp_j}{\sum_{q=1}^{J*} sp_q}$

end if
end for
/* Reinicia os acumuladores e deleta todos os neurônios que são ruídos */
for all neuronio j **in** Camada ② **do**
if $t_j > t_{max}$ **and** $sp_j < sp_{min}$ **then**
delete neuronio j
else
$t_j = 0; sp_j = 1.0$
end if
end for
/* Atualiza os I neurônios (variáveis linguísticas fuzzy) da camada ③ */
for all neuronio i **in** Camada ③ **do**
if $S(IV_i) < simMerge \wedge S(IV_i) > simRefit$ **then**
continue;
end if
$IV_i = \left\{IV_{i(j)} = separa(\mu_{i(j)}, C_{i(j)}) \mid j \in J^{(t)}\right\}$
$IV_i = sort(IV_i)$
$index = \arg_{IDX=1,\ldots,|IV_i|-1} \max(S(fundeESepara(IV_{i(IDX)}, IV_{i(IDX+1)})))$
$novaSim = S(fundeESepara(IV_{i(index)}, IV_{i(index+1)})$
while $novaSim > S(IV_i) \vee \mid IV_i \mid > maxMF$ **do**
$IV_{i(index)} = fundeESepara(IV_{i(index)}, IV_{i(index+1)})$
delete $IV_{i(index+1)}$
$index = \arg_{IDX=1,\ldots,|IV_i|-1} \max(S(fundeESepara(IV_{i(IDX)}, IV_{i(IDX+1)})))$
$novaSim = S(fundeESepara(IV_{i(index)}, IV_{i(index+1)}))$
end while
end for
/* Atualiza os O neurônios (variáveis linguísticas fuzzy) da camada ③ */
for all neuronio o **in** Camada ③ **do**
$L_{o(j)} = \mu_{o(j)} + \mu^T_{I(j)} A_{o(j)}$
$A_{o(j)} = -\frac{C^{Io}_{(j)}}{C^{oo}_{(j)}}$
$OV_o = \left\{OV_{o(j)}\left([A^T_{o(j)}, L_{o(j)}]\right) \mid j \in \{1..J^{(t)}\}\right\}$
end for
/* Atualiza todos os $K^{(t)}$ neurônios (regras fuzzy) da camada ④ */
for all neuronio k **in** Camada ④ **do**
$R^k =$ IF $\bigwedge_{i=1}^{I} X_i$ is $IV_{i(k_{i(j)})}$ THEN $\bigvee_{o=1}^{O} X_{I+o}$ is $OV_{o(k_{o(j)})} \quad (W_k)$
$R = fundeRegrasInconsistentes(R)$
end for
end function

5.3 MODO DE OPERAÇÃO RECALLING

No modo de operação recalling, apenas as camadas de ③ a ⑤ são ativadas. Durante este modo de operação, não ocorrem alterações nos parâmetros da rede. Porém, os modos de operação *learning* e *recalling* não precisam ocorrer separadamente, isto é, aprendizado e utilização podem ser intercalados.

O algoritmo apresentado a seguir sumariza os passos executados pela rede C-INFGMN durante sua execução em modo de operação *recalling*.

Algorithm 5.2 Pseudocódigo do Modo de Operação recalling

function INFGMNRECALLING(X_I)
/* Fuzzifica as I entradas na camada ③ */
$\lambda^{(IV_i)} = \{N^1(X_i, \mu_{i(k_i)}, C_{i(k_i)}) \mid k_i \in K_i^{(t)}\}, \forall i \in I$
/* Calcula as funções de saída na camada ③ */
$\phi^{(OV_o)} = \left\{ f_{o(k_o)}(X_I) = L_{o(k_o)} + X_I \cdot A_{o(k_o)} \mid k_o \in K_o^{(t)} \right\}, \forall o \in O$
for all neuronio k **in** Camada ④ **do**
/* Computa o grau de ativação α_k, usando a conjunção multiplicativa */
$\alpha_k = W_k \cdot \prod_{i=1}^{I} \lambda^{(IV_{i(k_i)})}$
end for
for all neuronio o **in** Camada ⑤ **do**
/* Computa a implicação multiplicativa e a agregação aditiva */
$\phi^{*o} = \sum_{k=1}^{K} \alpha_k \cdot \phi^{(OV_{o(k_o)})}$
/* Defuzzifica a saída por média ponderada (wtaver)*/
$y_o = \frac{\phi^{*o}}{\sum_{k=1}^{K} \alpha_k}$
end for
end function

6 RESULTADOS

6.1 IDENTIFICAÇÃO ONLINE DE UM SISTEMA DINÂMICO NÃO-LINEAR COM PROPRIEDADES VARIANTES NO TEMPO

Este experimento foi utilizado por Mazzutti *et al.* (2018) para validar o INFGMN contra o *Evolving Neural-fuzzy Semantic Memory* (eFSM) num ambiente com propriedades que variam no tempo. O sistema não-linear é definido pela equação:

$$y(t+1) = \frac{y(t)}{1+y^2(t)} + u^3(t) + f(t) \tag{6.1}$$

onde $u(t) = sin(2\pi t/100)$, $y(t)$ e $f(t)$ são, respectivamente, entrada, saída e um disturbio que será introduzido no sistema. O disturbio $f(t)$ é definido como:

$$f(t) = \begin{cases} 1, & 1001 \leq t \leq 2000 \\ 0, & \text{caso contrário} \end{cases} \tag{6.2}$$

O *dataset* possui duração $t \in [1, 3000]$, e as condições iniciais $(u(0), y(0))$ são dadas como $(0, 0)$ para $u(t) \in [-1, 1]$ e $y(t) \in [-1.5, 1.5]$. O objetivo é aproximar $y(t+1)$ dado $(u(t), y(t))$.

Os parâmetros de configuração da INFGMN utilizados foram: $\delta = 0.0263$, $\tau_{nov} = 7.2583\text{e}{-05}$, $t_{max} = 100$, $sp_{min} = 0.2762$ e $doMerge = false$. Para a C-INFGMN, os parâmetros de configuração da IGMM foram os mesmos, e os parâmetros de distinguibilidade foram: $doMerge = true$, $simMerge = 0.8$, $simRefit = 0.7$, e $maxMF = 9$.

As figuras 7, 8, 9 e 10 apresentam, respectivamente, o desempenho dos sistemas eFSM, INFGMN usando o modelo de ML, INFGMN usando o modelo de TS, e *Constrained INFGMN* (C-INFGMN) usando o modelo de TS. Os resultados experimentais são mostrados na Tabela 7, e as partições fuzzy geradas nos tempos $t \in \{400, 1400, 2400\}$ pelos modelos de TS propostos neste trabalho (INFGMN e C-INFGMN) estão representadas nas figuras 11 e 12.

Tabela 7 – Resultados experimentais no Sistema Dinâmico Não-Linear

Método	Tipo	#regras	RMSE
eFSM	ML	21	0.1
INFGMN (MAZZUTTI *et al.*, 2018)	ML	15.5	0.06
INFGMN	TS	13.87	0.0538
C-INFGMN	TS	13.03	0.0564

A eFSM atingiu um *Root Mean Squared Error* (RMSE) de ± 0.1, e sua saída é aproximada à saída esperada do Sistema Dinâmico Não-Linear (figuras 7(a) e 7(b)). Ela teve picos de erro de até ± 3.0, e demora para aprender no início do experimento, obtendo um RMSE abaixo de 0.5 somente após 600 passos (figura 7(d)). O modelo

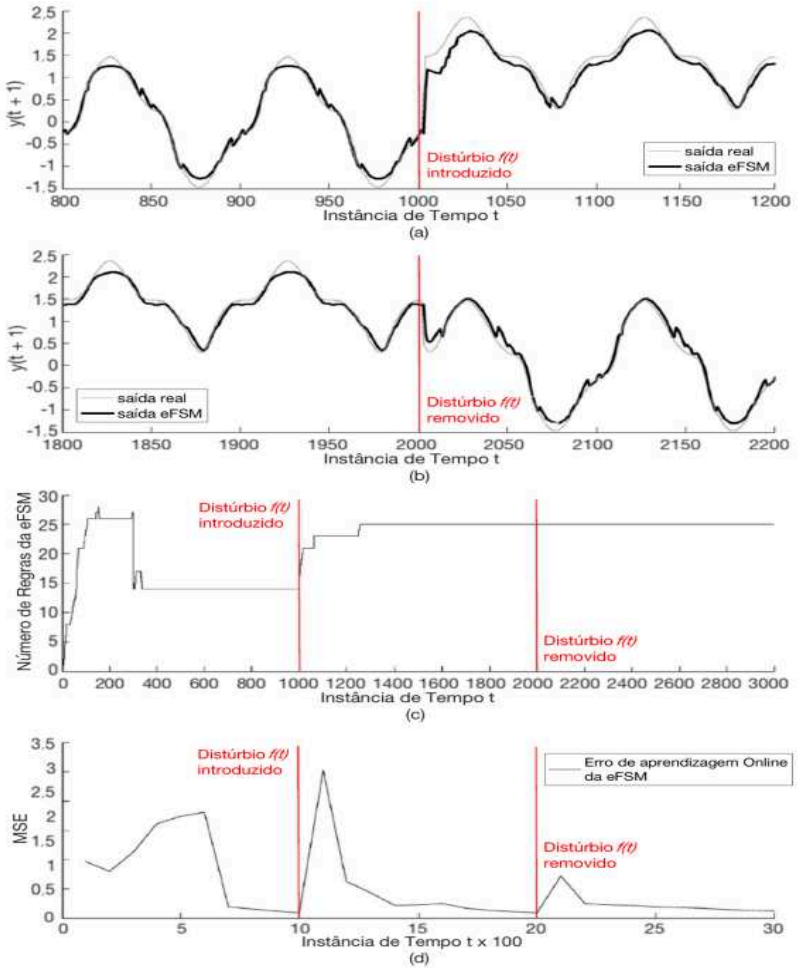


Figura 7 – Desempenho do modelo de ML no eFSM.

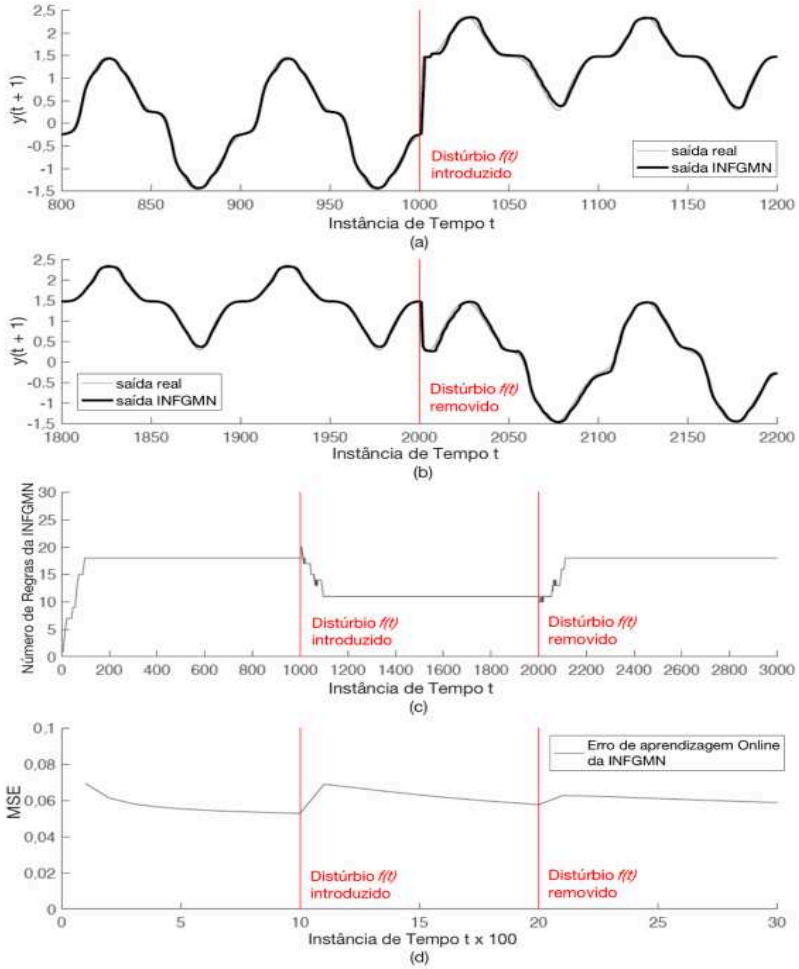


Figura 8 – Desempenho do modelo de ML no INFGMN.

(a) Saída quando o distúrbio f(t) é inserido em $t = 1000$. (b) Saída quando o distúrbio f(t) é removido em $t = 2000$. (c) Número de regras identificadas. (d) Erro de aprendizagem online.

Fonte – Mazzutti *et al.* (2018)

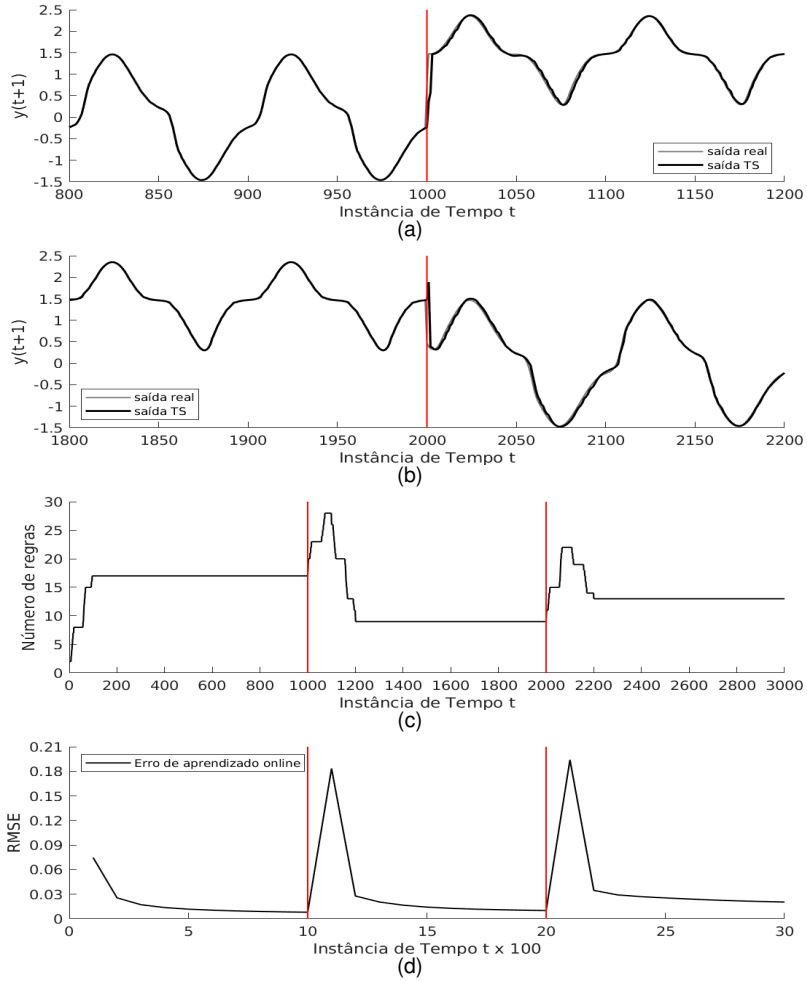


Figura 9 – Desempenho do modelo de TS no INFGMN.

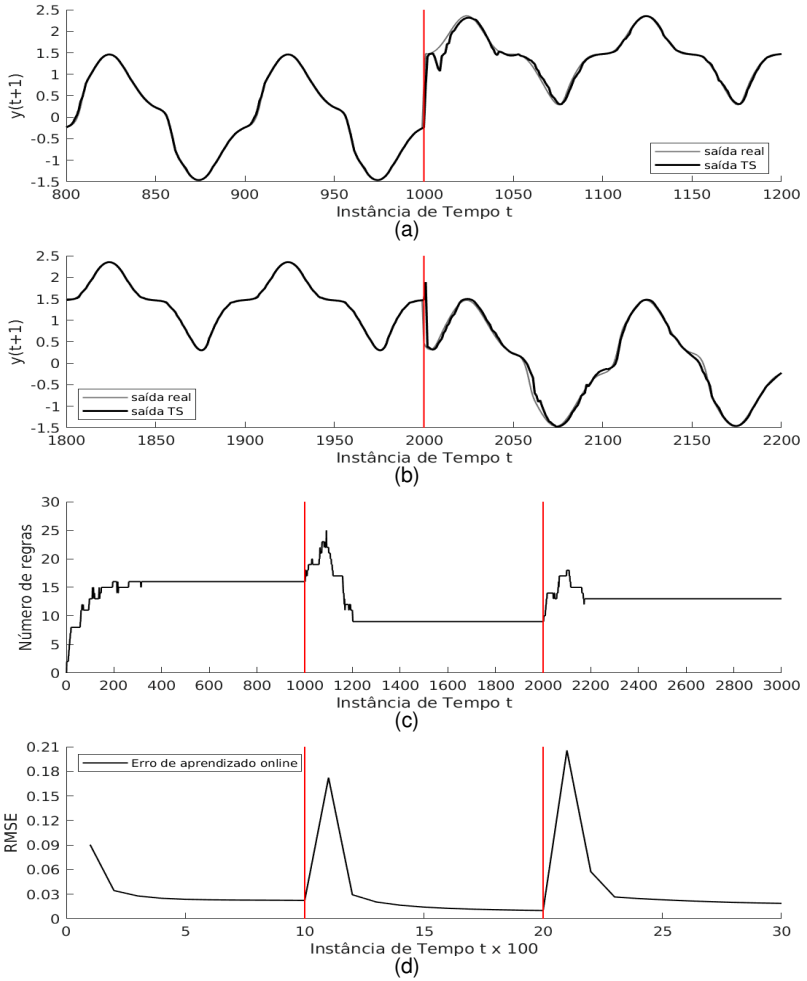
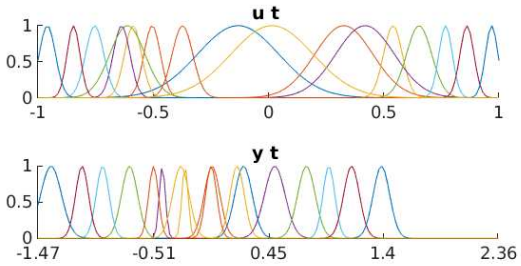


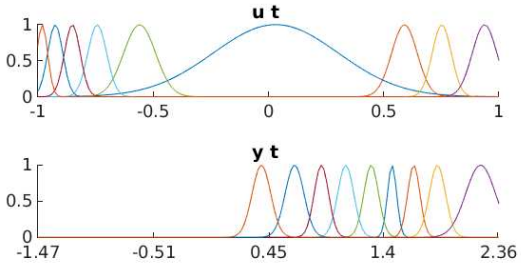
Figura 10 – Desempenho do modelo de TS no C-INFGMN.

(a) Saída quando o distúrbio f(t) é inserido em $t = 1000$. (b) Saída quando o distúrbio f(t) é removido em $t = 2000$. (c) Número de regras identificadas. (d) Erro de aprendizagem online.

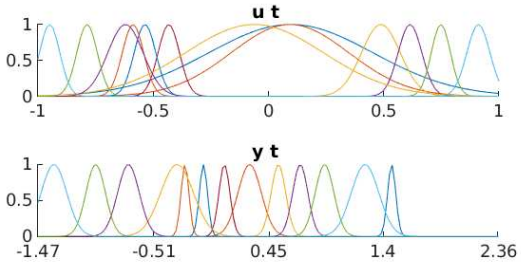
Fonte – Elaborado pelo autor (2020)



(a) t=400

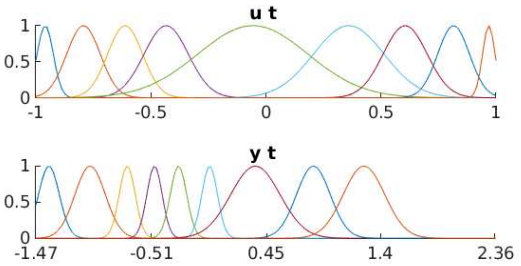


(b) t=1400

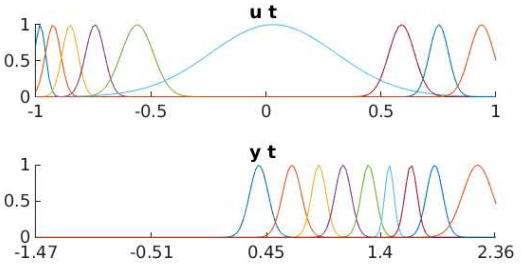


(c) t=2400

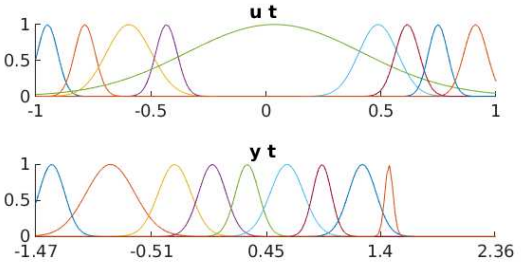
Figura 11 – Conjuntos fuzzy gerados pela INFGMN



(a) t=400



(b) t=1400



(c) t=2400

Figura 12 – Conjuntos fuzzy gerados pela C-INFGMN

Partições fuzzy geradas no Sistema Dinâmico Não-Linear.

Fonte – Elaborado pelo autor (2020)

de ML na INFGMN (MAZZUTTI *et al.*, 2018) atingiu um RMSE de ± 0.06, e sua saída corresponde à saída esperada (figuras 8(a) e 8(b)). Ela teve picos de erro de apenas ± 0.07, apresentando uma ótima estabilidade na presença de distúrbios, mas não progride muito com o tempo, mantendo o RMSE perto de 0.06 (figura 8(d)). O modelo de TS na INFGMN atingiu um RMSE de ± 0.054, e consegue manter uma correspondência com a saída esperada (figuras 9(a) e 9(b)), reafirmado pela figura 9(d), com picos de erro de ± 0.19, mas consegue manter um RMSE de ± 0.03 logo após 100 passos do início do experimento e da inserção e remoção do distúrbio. O mesmo vale para o modelo de TS na C-INFGMN que, embora demonstrou uma instabilidade ocasionada pelas restrições nos $\approx$ 25 passos seguintes à inserção do distúrbio, atingiu um RMSE de ± 0.057, e consegue alcançar uma correspondência com a saída esperada, (figuras

10(a) e 10(b)), reafirmado pela figura 9(d), com picos de erro de ± 0.2, mas sem deixar de entregar um RMSE competitivo de ± 0.06 após os primeiros 100 passos da remoção do distúrbio, e mantendo um RMSE de ± 0.03 logo após 200 passos do início do experimento e da inserção e remoção do distúrbio. Isso demonstra que a C-INFGMN não perdeu muita acurácia com o acréscimo de interpretabilidade advindo das restrições.

Considerando o número de regras, o modelo de TS na INFGMN obteve uma média de ± 13.9 regras, e o de C-INFGMN, com a fusão de regras inconsistentes, obteve uma média de ± 13 regras (figuras 9(c) e 10(c)), enquanto o modelo de ML na INFGMN (MAZZUTTI *et al.*, 2018) obteve uma média de ± 15.5 (figura 8(c)), e a eFSM de ± 21 regras (figura 7(c)).

O modelo de TS conseguiu então entregar uma acurácia e base de regras vagamente melhores, juntamente com partições fuzzy distinguíveis na C-INFGMN ao custo de aproximadamente 5% de acurácia. Porém, espera-se uma melhora de precisão substancial com o uso de modelos de TS ao custo da interpretabilidade no consequente das regras. Ao executar a INFGMN sob as mesmas configurações utilizadas para o modelo de TS, o modelo de ML obtém um RMSE de ± 0.1, observando-se então o ganho de acurácia esperado para o modelo de TS. Se eliminar os distúrbios (i.e., sem características variáveis no tempo), ou deixar o sistema rodar por mais tempo sem perturbação, ambos os modelo de TS poderiam entregar um RMSE que aproximar-se-ia de 0.03. Mas, para o problema em questão, o modelo de TS pode desapontar.

6.2 CONJUNTOS DE DADOS UCI

Para avaliar a capacidade da rede em problemas *offline*, foram utilizados 4 conjuntos de dados UCI, a saber: 1) *Abalone*; 2) *Boston Housing*; 3) *Concrete Compressive Strength*; e 4) *Concrete Slump Test*. Todos são problemas de regressão. As variáveis utilizadas foram as mesmas utilizadas por Tsekouras, Tsimikas *et al.* (2017). A tabela 8 informa as descrições básicas dos conjuntos de dados reais utilizados neste experimento. Originalmente, os conjuntos *Abalone* e *Boston Housing* têm 8 e 13 variáveis de entrada, respectivamente. Aqui foram desconsiderados a primeira variável de entrada do *Abalone* e a quarta do *Boston Housing* porque são variáveis categóricas. Para o conjunto *Concrete Slump Test*, o dado original tem três variáveis de saída, e aqui foi considerado apenas a primeira, relativa à predição da variável *slump (in cm)*.

Para validação do modelo, 60% dos dados foram utilizados para treinamento, e 40% para teste, assim como em (TSEKOURAS; TSIMIKAS *et al.*, 2017). Para aleatoriedade dos dados, foi realizada uma validação cruzada k-fold, com k=5, mas tomando 2 subconjuntos para teste e 3 para treinamento, assim obtendo $\frac{5!}{(5-2)!2!} = 10$ combinações diferentes. A rede C-INFGMN é comparada com os seguintes modelos: HFC (NIROS; TSEKOURAS, 2012), GON (PEDRYCZ; PARK *et al.*, 2008), o modelo de Pedrycz e

Tabela 8 – Descrição dos conjuntos de dados UCI.

Nome	#Entradas	#Instâncias
Abalone	7	4177
Boston Housing	12	506
Concrete Compressive Strength	8	1030
Concrete Slump Test	7	103

Kwak (2006), o modelo de Roh *et al.* (2011), o modelo de Tsekouras (2016), o modelo de Tsekouras e Tsimikas (2013) e Constrained PSO (TSEKOURAS; TSIMIKAS *et al.*, 2017). Alguns deles são modelos de redes de função de base radial (RBFN), os quais, sob certas restrições, são funcionalmente equivalentes a um modelo fuzzy de Sugeno. Os resultados para estes modelos foram extraídos de Tsekouras, Tsimikas *et al.* (2017). Também são comparados os modelos gerados pela INFGMN (sem restrições de distinguibilidade). As medidas de *benchmarking* são a acurácia das respostas dos modelos para os conjunto de treino e de teste, calculadas como RMSE, e o número de regras geradas.

Tabela 9 – Parâmetros de configuração da IGMM na INFGMN.

Nome	δ	τ_{nov}	t_{max}	sp_{min}
Abalone	0.2726	3.3717e-07	1253	5.6569
Boston Housing	0.4585	2.0530e-04	506	1
Concrete Compressive Strength	0.5946	0.0066	309	2.8284
Concrete Slump Test	0.7711	0.0526	103	1

Os parâmetros de configuração da INFGMN utilizados para os conjuntos de dados UCI são mostrados na tabela 9 (com $doMerge = false$). Para a C-INFGMN, os parâmetros de configuração da IGMM foram os mesmos, e os parâmetros de distinguibilidade $doMerge = true$ e $simMerge = 0.9$ para todos os conjuntos de dados UCI. $simRefit$ não foi necessário, pois a partição distinguível só foi criada após o processo de treinamento (mudando $doMerge$ de $false$ para $true$). Para o parâmetro $maxMF$, os valores utilizados foram: $maxMF = 5$ para *Abalone*, $maxMF = 7$ para *Boston Housing*, $maxMF = 5$ para *Concrete Compressive Strength*, e $maxMF = 7$ para *Concrete Slump Test*.

Os resultados experimentais para os conjuntos de dados UCI são mostrados na Tabela 10. O modelo proposto obteve a melhor acurácia nos conjuntos de testes para os conjuntos de dados *Abalone*, *Concrete Compressive Strenght* e *Concrete Slump Test*, e ficou em segundo para o conjunto de dados *Boston Housing*. Para os conjuntos de treinamento, a acurácia do modelo proposto foi competitiva nos conjuntos de dados *Abalone*, *Boston Housing* e *Concrete Compressive Strength*. Isso demonstra que a rede C-INFGMN tem uma boa capacidade de generalizar os dados de treinamento sem cair em overfitting. Em termos de números de regras, o modelo proposto foi

competitivo no experimento com o *Abalone*, onde o C-INFGMN chegou a diminuir o número de regras com a fusão de regras inconsistentes, e obteve a menor base de regras no experimento com o *Concrete Slump Test*, enquanto nos experimentos com o *Boston Housing* e *Concrete Compressive Strength*, o tamanho da base de regras foi consideravelmente grande.

Tabela 10 – Resultados experimentais para os conjuntos de dados UCI.

Método	#regras ($\mu \pm \sigma$)	Treino ($\mu \pm \sigma$)	Teste ($\mu \pm \sigma$)
Abalone			
HFC	10	2.449	2.685
	20	2.126	2.269
GON	10	2.315	2.391
	20	2.161	2.214
Tsekouras (2016)	10	2.056	2.224
	5	2.166	2.293
Constrained PSO (tipo TS)	6	2.072 ± 0.026	2.212 ± 0.022
INFGMN (tipo TS)	8.9 ± 2.514	2.129 ± 0.035	2.173 ± 0.045
C-INFGMN (tipo TS)	8 ± 2.108	2.146 ± 0.037	2.162 ± 0.040
Boston Housing			
Pedrycz e Kwak (2006)	25	5.21	6.14
	25	4.80	5.22
	25	4.12	5.32
Roh *et al.* (2011)	2	3.149	4.043
	4	4.306	4.726
	5	4.334	4.740
	7	3.992	4.589
Tsekouras (2016)	8	4.0254	5.190
	4	4.5826	5.068
Constrained PSO (tipo TS)	8	3.077 ± 0.204	3.596 ± 0.207
INFGMN (tipo TS)	14.7 ± 5.272	3.282 ± 0.730	4.127 ± 0.349
C-INFGMN (tipo TS)	14.7 ± 5.272	3.370 ± 0.756	3.965 ± 0.550
Concrete compressive strength			
Tsekouras e Tsimikas (2013)	4	13.01	17.01
	8	10.40	11.58
	12	9.886	11.28
Constrained PSO (tipo TS)	9	7.343 ± 0.510	8.427 ± 0.484
INFGMN (tipo TS)	12.9 ± 3.604	8.092 ± 0.429	8.485 ± 0.354
C-INFGMN (tipo TS)	12.9 ± 3.604	8.130 ± 0.464	8.417 ± 0.408
Concrete slump test			
Constrained PSO (tipo TS)	7	5.275 ± 0.269	8.018 ± 0.273
INFGMN (tipo TS)	4.3 ± 2.791	6.355 ± 0.792	7.574 ± 0.756
C-INFGMN (tipo TS)	4.3 ± 2.791	6.312 ± 0.868	7.416 ± 0.497

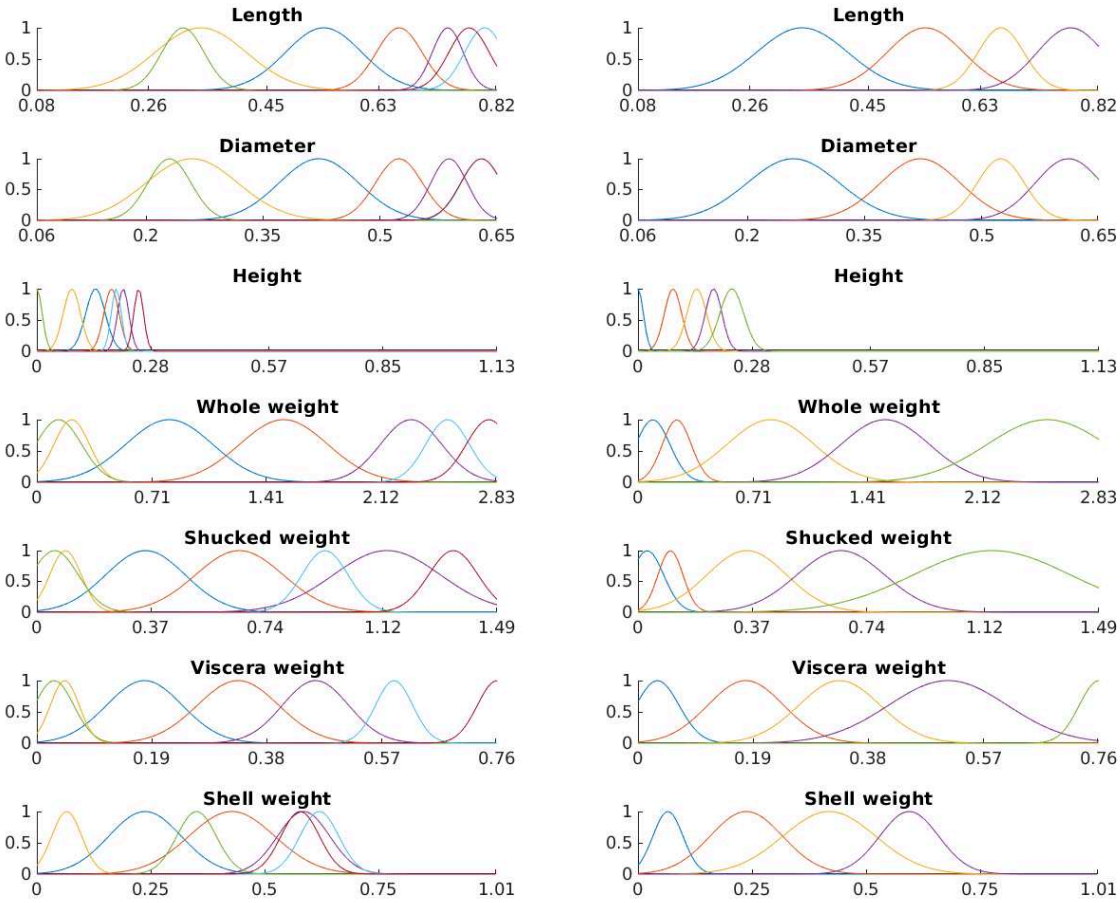
Ainda na tabela 10, ao comparar os resultados da C-INFGMN com a INFGMN, observa-se que a C-INFGMN conseguiu melhorar a interpretabilidade sem perder sua acurácia. Ela foi até capaz de, na média, reduzir o overfitting com as restrições de distinguibilidade, ao se observar um aumento na acurácia com os dados de teste e perda

Tabela 11 – Regras geradas no conjunto Abalone correspondente à partição fuzzy na figura 13a

R1.	If (Length is about 0.541) and (Diameter is about 0.421) and (Height is about 0.144) and (Whole weight is about 0.814) and (Shucked weight is about 0.351) and (Viscera weight is about 0.177) and (Shell weight is about 0.237) then (Rings is 0.065 -0.155 *Length +0.013 *Diameter +0.542 *Height +1.195 *Whole weight -1.208 *Shucked weight -0.293 *Viscera weight +0.465 *Shell weight) (0.60016)
R2.	If (Length is about 0.662) and (Diameter is about 0.524) and (Height is about 0.183) and (Whole weight is about 1.517) and (Shucked weight is about 0.655) and (Viscera weight is about 0.332) and (Shell weight is about 0.427) then (Rings is -0.130 -0.377 *Length +0.353 *Diameter +0.286 *Height +0.760 *Whole weight -0.743 *Shucked weight -0.232 *Viscera weight +0.242 *Shell weight) (0.18029)
R3.	If (Length is about 0.343) and (Diameter is about 0.258) and (Height is about 0.085) and (Whole weight is about 0.214) and (Shucked weight is about 0.092) and (Viscera weight is about 0.045) and (Shell weight is about 0.065) then (Rings is 0.135 +0.127 *Length +0.128 *Diameter +0.647 *Height +0.103 *Whole weight -0.975 *Shucked weight +0.130 *Viscera weight +0.856 *Shell weight) (0.21007)
R4.	If (Length is about 0.740) and (Diameter is about 0.588) and (Height is about 0.212) and (Whole weight is about 2.301) and (Shucked weight is about 1.133) and (Viscera weight is about 0.460) and (Shell weight is about 0.585) then (Rings is 0.112 -0.382 *Length +0.230 *Diameter +0.445 *Height +0.065 *Whole weight +0.007 *Shucked weight -0.074 *Viscera weight +0.001 *Shell weight) (0.0070892)
R5.	If (Length is about 0.315) and (Diameter is about 0.23) and (Height is about 0) and (Whole weight is about 0.134) and (Shucked weight is about 0.057) and (Viscera weight is about 0.028) and (Shell weight is about 0.350) then (Rings is -0.642) (0.00079894)
R6.	If (Length is about 0.8) and (Diameter is about 0.63) and (Height is about 0.195) and (Whole weight is about 2.526) and (Shucked weight is about 0.933) and (Viscera weight is about 0.59) and (Shell weight is about 0.62) then (Rings is 0.571) (0.00079894)
R7.	If (Length is about 0.775) and (Diameter is about 0.63) and (Height is about 0.25) and (Whole weight is about 2.779) and (Shucked weight is about 1.348) and (Viscera weight is about 0.76) and (Shell weight is about 0.578) then (Rings is -0.214) (0.00079894)

com os dados de treino, para todos os 4 conjuntos de dados UCI. Os conjuntos fuzzy gerados por elas, para uma das simulações de cada experimento, estão representados nas figuras 13, 14, 15 e 16. Observa-se nessas figuras que a C-INFGMN foi capaz de tornar distinguíveis as partições fuzzy correspondentes da INFGMN.

Na partição fuzzy da variável *Whole weight* do experimento *Abalone*, na figura 13, percebe-se que os 3 conjuntos à direita no espaço de entrada foram fundidos, enquanto os 2 conjuntos à esquerda foram separados. Isso é devido às componentes gaussianas correspondentes aos 3 conjuntos fuzzy à direita terem uma proporção menor na mistura (potencialmente de dados discrepantes), o que torna-os menos representativos para os dados subjacentes, e tendo portanto um menor peso no cál-



(a) INFGMN (b) C-INFGMN

Figura 13 – Conjuntos fuzzy para as variáveis de entrada no dataset UCI - Abalone

Fonte – Elaborado pelo Autor (2020)

culo para a similaridade da partição fuzzy. Do outro lado, os 2 conjuntos à esquerda, altamente sobrepostos, no cálculo de sua similaridade, foram julgados mais representativos separados do que fundidos. Isso pois, como pode se observar na tabela 11, o conjunto em 0.214, correspondente a regra 3, possui peso 0.21 e o conjunto em 0.134, correspondente a regra 5, possui peso $8\mathrm{e}{-4}$. Como o conjunto da regra 3 possui pertinência próxima a 0.5 no conjunto vizinho, ele sofreu poucas mudanças para garantia da distinguibilidade, mantendo assim alta similaridade ao conjunto original. Dessa forma, os conjuntos correspondentes as componentes discrepantes sofreram maiores mudanças em prol das componentes de altas proporções na mistura. O mesmo se aplica à variável de entrada *Shucked weight*, em que o conjunto da regra 3 (em 0.092, de peso 0.21) foi separado do conjunto da regra 5 (em 0.057, de peso $8\mathrm{e}{-4}$), e os conjuntos das regras 6 (em 0.933, de peso $8\mathrm{e}{-4}$), 4 (em 1.133, de peso $8\mathrm{e}{-4}$) e 7 (em

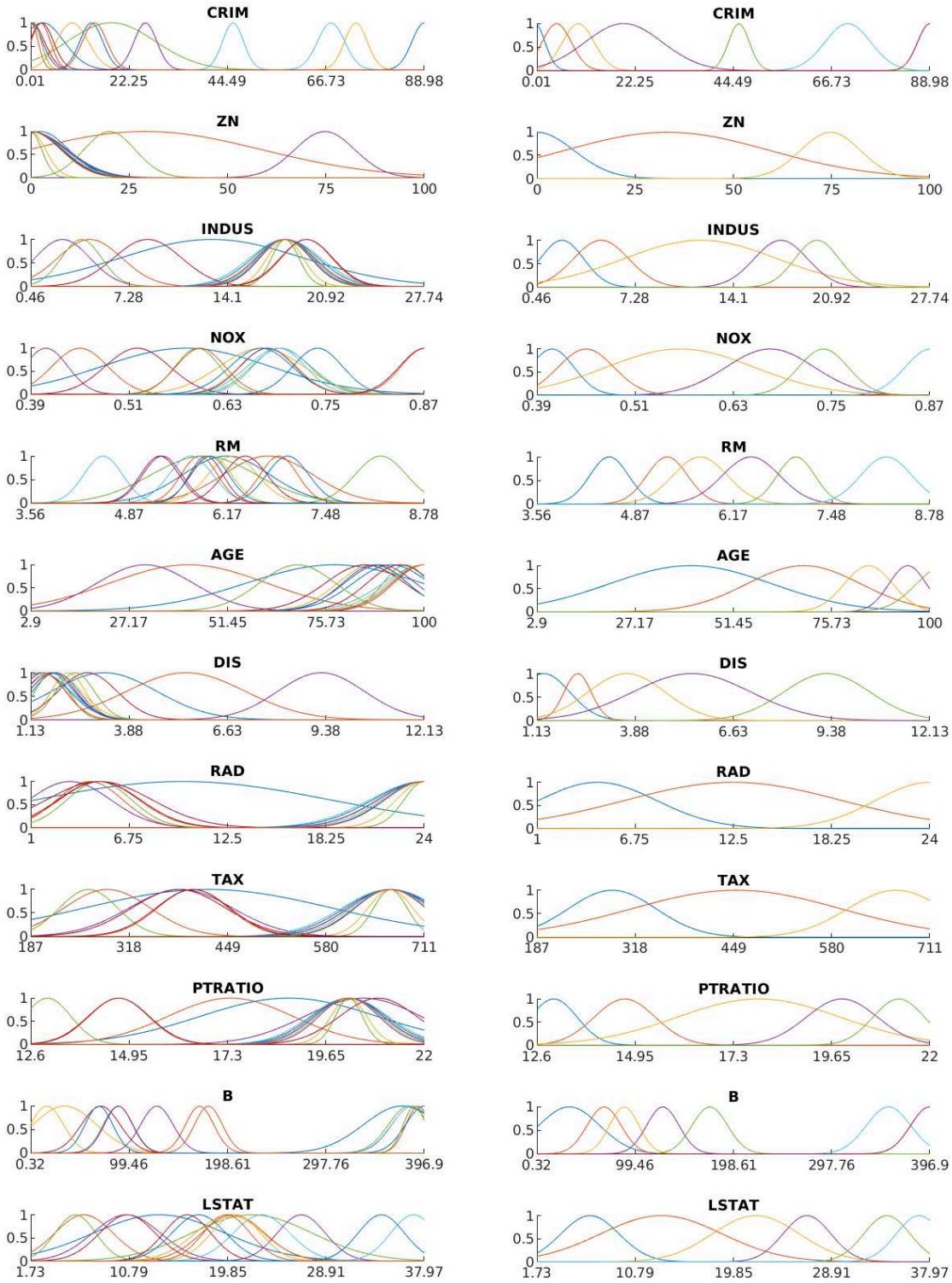
1.348, de peso $7\mathrm{e}{-3}$) foram fundidos.

6.3 DISCUSSÃO

Para validar o modelo proposto, foram realizados 2 experimentos. O primeiro foi a identificação online de um sistema dinâmico não linear com características que variam no tempo definido pela equação 6.1. Os resultados deste experimento demonstraram que o modelo de TS na INFGMN obteve um bom desempenho no número de regras, e a C-INFGMN garantiu distinguibilidade na partição fuzzy ao custo de aproximadamente 5% de sua acurácia. Embora a acurácia dos modelos de TS foi similar ao de ML em Mazzutti *et al.* (2018), enquanto esperava-se uma maior precisão com sistemas Takagi-Sugeno ao custo da interpretabilidade no consequente das regras, seu RMSE chegou a aproximadamente 50% do modelo de ML nos momentos em que adequou-se aos distúrbios, sugerindo ser uma boa alternativa para ambientes estáveis.

O segundo experimento foi para avaliar as habilidades da rede em problemas *offline*, utilizando 4 conjuntos de dados UCI com problemas de regressão, a saber: 1) *Abalone*; 2) *Boston Housing*; 3) *Concrete Compressive Strength*; e 4) *Concrete Slump Test*. Os resultados foram promissores, obtendo bons desempenhos em termos de acurácia, embora o número de regras tenha sido alto para 2 dos 4 conjuntos. A C-INFGMN foi capaz de gerar partições fuzzy distinguíveis e, ademais, entregou uma redução de *overfitting* em todos os 4 conjuntos de dados.

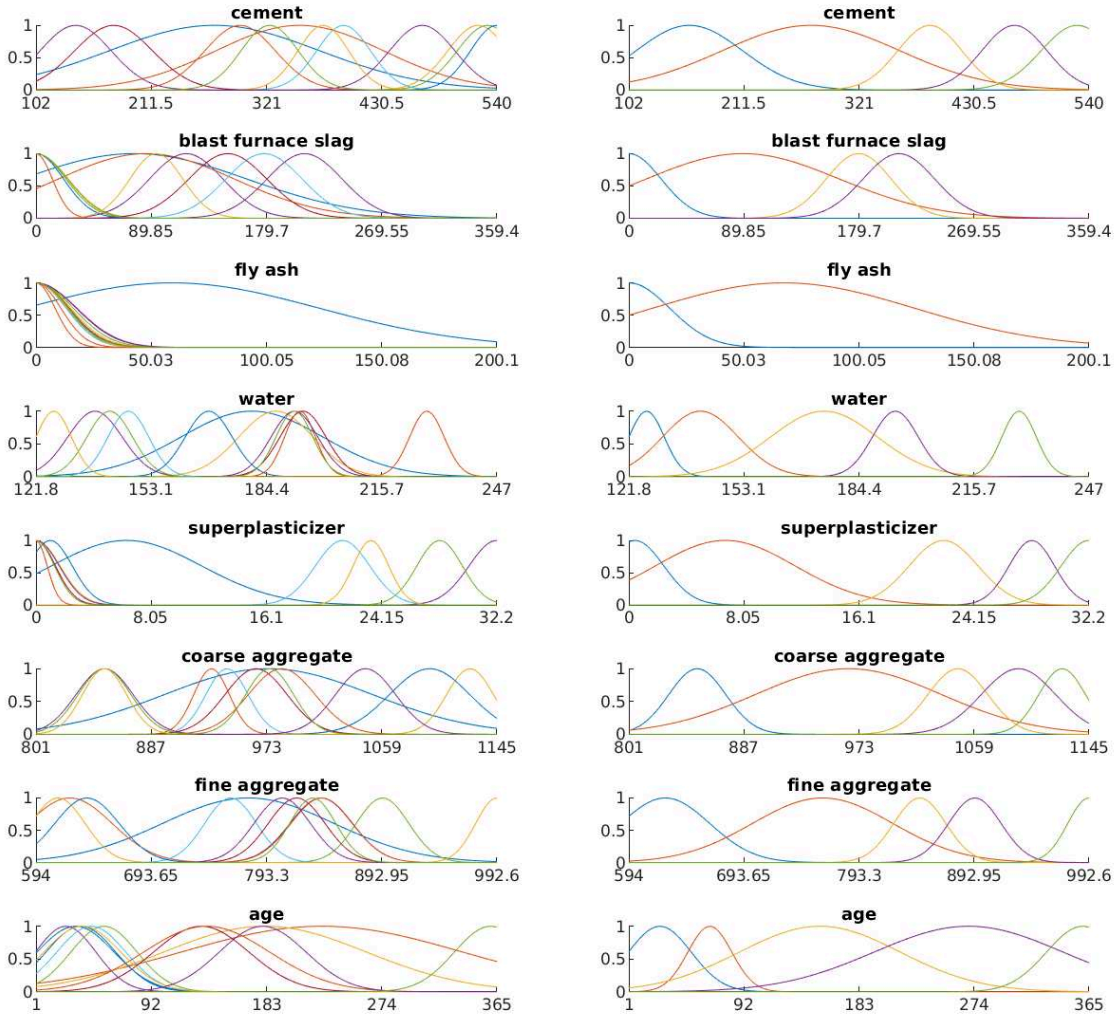
O método de similaridade utilizado, avaliando toda a partição fuzzy, e considerando o peso das regras para guiar a fusão ou separação dos conjuntos fuzzy, demonstrou-se bem eficaz para os problemas *offline* pelo que se observou na redução do overfitting. Quanto ao experimento *online*, ela gerou uma instabilidade nos momentos de inserção/remoção do distúrbio, provavelmente devido ao baixo peso das novas componentes na medida de similaridade, causando-lhes maiores modificações em prol das componentes já estabelecidas e que, em tal experimento, tornaram-se obsoletas num incremento do tempo. Para ambientes mais estáveis ou cujas mudanças sejam mais sutis no tempo, a C-INFGMN pode ser uma alternativa.



(a) INFGMN (b) C-INFGMN

Figura 14 – Conjuntos fuzzy para as variáveis de entrada no dataset UCI - Boston housing

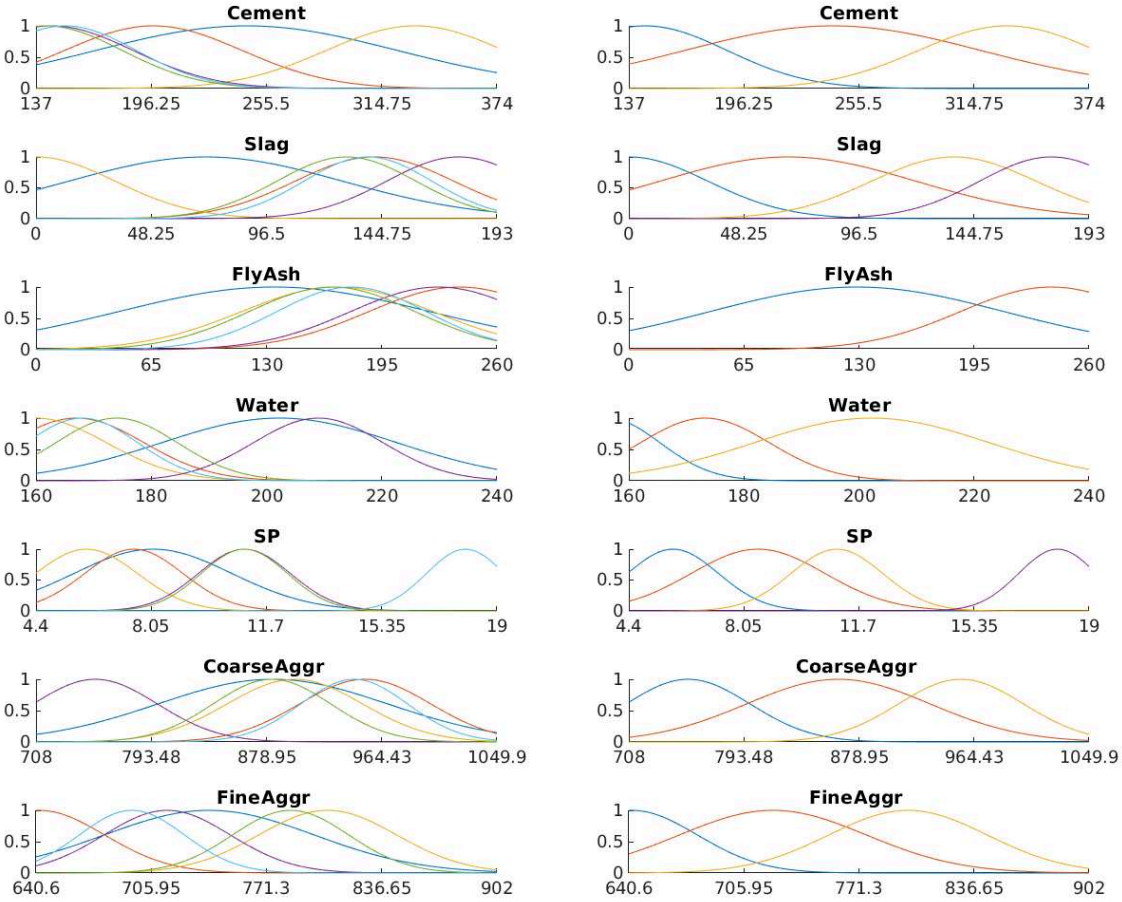
Fonte – Elaborado pelo Autor (2020)



(a) INFGMN (b) C-INFGMN

Figura 15 – Conjuntos fuzzy para as variáveis de entrada no dataset UCI - Concrete compressive strength

Fonte – Elaborado pelo Autor (2020)



(a) INFGMN

(b) C-INFGMN

Figura 16 – Conjuntos fuzzy para as variáveis de entrada no dataset UCI - Concrete slump test

Fonte – Elaborado pelo Autor (2020)

7 CONSIDERAÇÕES FINAIS

7.1 CONCLUSÕES

Este trabalho buscou expandir as funcionalidades do sistema INFGMN, proporcinando sua aplicação com modelos de Takagi-Sugeno e melhorando sua interpretabilidade via distinguibilidade nas partições fuzzy. Ele revisou conceitos de Sistemas de Inferência Fuzzy, cobrindo desde a lógica fuzzy até Redes Neuro-Fuzzy e interpretabilidade em sistemas fuzzy. Foram apresentados trabalhos relacionados que possuem distinguibilidade na partição fuzzy, e seus métodos para garantir tal critério. Também foram apresentados a INFGMN, sua arquitetura e trabalhos que fundamentaram sua proposta.

Este trabalho, então, conferiu à INFGMN a capacidade de manipular modelos de Takagi-Sugeno, garantir a distinguibilidade nas partições fuzzy, e obter um número moderado de conjuntos. Porém, devido à falta de um método de tuning para as partições fuzzy obtidas com as restrições, a capacidade de obter um número moderado de conjuntos na partição fuzzy deve ser utilizada com parcimônia.

Diferente dos trabalhos relacionados, que avaliavam a similaridade entre 2 conjuntos isolados para guiar o processo de fusão, neste trabalho foi elaborado um método de similaridade que avalia a partição fuzzy para guiar a fusão/separação dos conjuntos fuzzy, ponderado pelos pesos das regras para gerar menores alterações nos conjuntos cujas regras descrevam melhor o sistema e não sejam oriundas de dados discrepantes, dessa forma mantendo a acurácia no sistema fuzzy aproximado.

Para validar o modelo proposto, foram realizados 2 experimentos. O primeiro foi a identificação online de um sistema dinâmico não linear com características que variam no tempo definido pela equação 6.1. Os resultados deste experimento demonstraram que o modelo de TS na INFGMN obteve um bom desempenho no número de regras, e a C-INFGMN garantiu distinguibilidade na partição fuzzy ao custo de aproximadamente 5% de sua acurácia. Embora a acurácia dos modelos de TS foi similar ao de ML em Mazzutti *et al.* (2018), ela chegou a dobrar (tendo $\approx 50\%$ do RMSE) nos momentos em que adequou-se aos distúrbios, sugerindo ser uma boa alternativa para ambientes estáveis ou cujas mudanças sejam mais sutis no tempo.

O segundo experimento foi para avaliar as habilidades da rede em problemas *offline*, utilizando 4 conjuntos de dados UCI com problemas de regressão, a saber: 1) *Abalone*; 2) *Boston Housing*; 3) *Concrete Compressive Strength*; e 4) *Concrete Slump Test*. Embora o número de regras tenha sido alto para 2 dos 4 conjuntos, os resultados obtiveram bons desempenhos em termos de acurácia. A C-INFGMN foi capaz de gerar partições fuzzy distinguíveis e, ademais, entregou uma redução de *overfitting* em todos os 4 conjuntos de dados, demonstrando a eficácia do método de similaridade utilizado em problemas *offline*.

Os resultados e as conclusões deste trabalho são preliminares, e mais testes são necessários para validação do novo sistema.

7.2 TRABALHOS FUTUROS

Como trabalhos futuros, algumas áreas para melhoria são:

- Com a fusão de conjuntos no antecedente, pode-se gerar inconsistência de regras. Tal caso já foi trabalhado nos modelos de Takagi-Sugeno pela média ponderada das funções de saída. Porém, para modelos de Mamdani-Larsen, seus consequentes são conjuntos fuzzy que podem ser dissimilares para fusão, ou estarem sobrepostos por outros consequentes cujas regras não são inconsistêntes e talvez não devessem ser fundidos.
- Para um melhor tuning das regras do sistema fuzzy aproximado, poderia-se realizar a fusão de conjuntos fuzzy diretamente nas componentes do GMM, de forma que elas assimilem os dados futuros assumindo a partição fuzzy distinguível. Isso também poderia resolver a inconsistência de regras nos modelos de Mamdani-Larsen. Porém, isso poderia engessar as componentes, tirando-lhes sua autonomia para atualizar os conjuntos fuzzy resultantes, e possivelmente afetando a capacidade de Plasticidade da INFGMN.

REFERÊNCIAS

CASILLAS, Jorge; CORDÓN, Oscar; HERRERA, Francisco; MAGDALENA, Luis. Interpretability improvements to find the balance interpretability-accuracy in fuzzy modeling: an overview. *In:* INTERPRETABILITY issues in fuzzy modeling. [*S.l.*]: Springer, 2003. p. 3–22.

ENGEL, Paulo Martins; HEINEN, Milton Roberto. Incremental learning of multivariate gaussian mixture models. *In:* SPRINGER. BRAZILIAN Symposium on Artificial Intelligence. [*S.l.*: *s.n.*], 2010. p. 82–91.

GAN, Ming-Tao; HANMANDLU, Madasu; TAN, Ai Hui. From a Gaussian mixture model to additive fuzzy systems. **IEEE Transactions on Fuzzy Systems**, IEEE, v. 13, n. 3, p. 303–316, 2005.

HEFNY, Hesham A. Comments on “Distinguishability quantification of fuzzy sets”. **Information Sciences**, Elsevier, v. 177, n. 21, p. 4832–4839, 2007.

JANG, J-SR; SUN, Chuen-Tsai. Neuro-fuzzy modeling and control. **Proceedings of the IEEE**, IEEE, v. 83, n. 3, p. 378–406, 1995.

MAMDANI, Ebrahim H. Application of fuzzy algorithms for control of simple dynamic plant. *In:* IET, 12. PROCEEDINGS of the institution of electrical engineers. [*S.l.*: *s.n.*], 1974. p. 1585–1588.

MATHWORKS. **Fuzzy logic toolbox user’s guide**. 2019b. [*S.l.*], 2019. Disponível em: `https://www.mathworks.com/help/pdf_doc/fuzzy/index.html`.

MAZZUTTI, Tiago; ROISENBERG, Mauro; FREITAS FILHO, Paulo José de. INFGMN–Incremental Neuro-Fuzzy Gaussian mixture network. **Expert Systems with Applications**, Elsevier, v. 89, p. 160–178, 2017.

MAZZUTTI, Tiago; ROISENBERG, Mauro; FREITAS FILHO, Paulo José de. **MODELO INCREMENTAL NEURO-FUZZY GAUSSIAN MIXTURE NETWORK (INFGMN)**. 2018. f. 111. Doutorado em Ciências da Computação – Universidade Federal de Santa Catarina, UFSC, Florianópolis, SC.

MENCAR, Corrado; CASTELLANO, Giovanna; FANELLI, Anna M. Distinguishability quantification of fuzzy sets. **Information Sciences**, Elsevier, v. 177, n. 1, p. 130–149, 2007.

NIROS, Antonios D; TSEKOURAS, George E. A novel training algorithm for RBF neural network using a hybrid fuzzy clustering approach. **Fuzzy Sets and Systems**, Elsevier, v. 193, p. 62–84, 2012.

PEDRYCZ, Witold; KWAK, Keun-Chang. Linguistic models as a framework of user-centric system modeling. **IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans**, IEEE, v. 36, n. 4, p. 727–745, 2006.

PEDRYCZ, Witold; PARK, Ho-Sung; OH, Sung-Kwun. A granular-oriented development of functional radial basis function neural networks. **Neurocomputing**, Elsevier, v. 72, n. 1-3, p. 420–435, 2008.

PRAMOD, CP; PILLAI, GN. A Linguistically Interpretable ELANFIS for Classification Problems. *In:* IEEE. 2018 IEEE Symposium Series on Computational Intelligence (SSCI). [*S.l.*: *s.n.*], 2018. p. 720–727.

ROH, Seok-Beom; OH, Sung-Kwun; PEDRYCZ, Witold. Design of fuzzy radial basis function-based polynomial neural networks. **Fuzzy sets and systems**, Elsevier, v. 185, n. 1, p. 15–37, 2011.

TAKAGI, Tomohiro; SUGENO, Michio. Fuzzy identification of systems and its applications to modeling and control. **IEEE transactions on systems, man, and cybernetics**, IEEE, n. 1, p. 116–132, 1985.

TSEKOURAS, George E. Fuzzy rule base simplification using multidimensional scaling and constrained optimization. **Fuzzy sets and systems**, Elsevier, v. 297, p. 46–72, 2016.

TSEKOURAS, George E; TSIMIKAS, John. On training RBF neural networks using input–output fuzzy clustering and particle swarm optimization. **Fuzzy Sets and Systems**, Elsevier, v. 221, p. 65–89, 2013.

TSEKOURAS, George E; TSIMIKAS, John; KALLONIATIS, Christos; GRITZALIS, Stefanos. Interpretability constraints for fuzzy modeling implemented by constrained particle swarm optimization. **IEEE Transactions on Fuzzy Systems**, IEEE, v. 26, n. 4, p. 2348–2361, 2017.

TUNG, Whye Loon; QUEK, Chai. eFSM—A novel online neural-fuzzy semantic memory model. **IEEE Transactions on Neural Networks**, IEEE, v. 21, n. 1, p. 136–157, 2009.

ZADEH, Lotfi A. Fuzzy logic. **Computer**, IEEE, v. 21, n. 4, p. 83–93, 1988.

ZADEH, Lotfi A. Fuzzy sets. **Information and control**, Elsevier, v. 8, n. 3, p. 338–353, 1965.

ZADEH, Lotfi A. Outline of a new approach to the analysis of complex systems and decision processes. **IEEE Transactions on systems, Man, and Cybernetics**, IEEE, n. 1, p. 28–44, 1973.

ZHOU, Shang-Ming; GAN, John Q. Low-level interpretability and high-level interpretability: a unified view of data-driven interpretable fuzzy system modelling. **Fuzzy sets and systems**, Elsevier North-Holland, Inc., v. 159, n. 23, p. 3091–3131, 2008.

APÊNDICE A – IMPLEMENTAÇÃO DO MODELO C-INFGMN (MATLAB)

```
classdef INFGMN < handle

    methods

        %% Inicializacao dos parametros omitida ...

         %% Train the INFGMN with the data set X
        function self = train(self, X)

            validateattributes(X, {"dataset"}, {"nonempty", "ncols", ...
               size(self.ranges, 2)}, strcat(self.name, ":train"),""X"");
            X = double(X);

            if self.normalize
                X = mapminmax("apply", X", self.proportion)";
            end

            if any(any(isnan(X)))
                X = self.internal_imputation(X);
            end

            N = size(X,1);
            for i = 1:N
                didCreate = false;
                x = X(i,:);
                self.computeLikelihood(x);
                if (¬self.hasAcceptableDistribution())
                    self.createComponent(x);
                    didCreate = true;
                end
                self.computePosterior();
                self.updateComponents(x);
                self.sampleSize = self.sampleSize + 1;
                self.removeSpurious();
                if self.doMerge
                    self.updateFisVar(didCreate);
                end
            end

            % Force fuzzy layer update.
            self.needsFisUpdate = true;
        end

```

```
43          function updateFisVar(self, thereIsANewComponent)
44              for i_vn = 1:length(self.varNames) % para cada feature
45                  compMFs = [ squeeze(self.covs(i_vn, i_vn, :)) .^ ...
                        self.spread, self.means(:, i_vn) ];
46                  self.FPstruct.(self.varNames{i_vn}).updateSystem( ...
                        compMFs, self.priors, thereIsANewComponent );
47              end
48          end
49
50          function refitFP(self)
51              for i_vn = 1:length(self.varNames) % para cada feature
52                  compMFs = [ squeeze(self.covs(i_vn, i_vn, :)) .^ ...
                        self.spread, self.means(:, i_vn) ];
53                  self.FPstruct.(self.varNames{i_vn}) = FuzzyPartition( ...
54                      self.maxMFs, self.simMerge, self.simRefit, compMFs, ...
                            self.priors );
55              end
56          end
57
58
59          %% Compute the log probability density of the multivariate ...
                normal distribution
60          % evaluated for each INFGMN gaussian component.
61          function computeLikelihood(self, x)
62              if self.nc > 0
63                  [self.loglike, self.mahalaD] = self.logmvnpdf(x, ...
                        self.means, self.covs);
64              end
65          end
66
67          %% Compute the log probability density of a multivariate normal ...
                distribution
68          function [loglike, mahalaD] = logmvnpdf(self, x, means, covs)
69              [n,d] = size(x);
70              k = size(means,1);
71              loglike = zeros(n, k);
72              mahalaD = zeros(n, k);
73              logSqrtDetCov = zeros(1, k);
74
75              for j = 1:k
76                  Xcentered = bsxfun(@minus, x, means(j,:));
77                  if self.ignoreCovariance
78                      L = sqrt(diag(covs(:,:,j))"); % a row vector
79                      if  any(L < eps(max(L)) * d)
80                          error("Ill Conditioned Covariance.");
81                      end
82                      xRinv = bsxfun(@times, Xcentered , (1 ./ L));
```

```
83                      logSqrtDetCov(j) = sum(log(L));
84                  else
85                      [L,err] = cholcov(covs(:,:,j), 0); % a matrix
86                      if err ≠ 0
87                          error("Ill Conditioned Covariance.");
88                      end
89                      xRinv = Xcentered / L;
90                      logSqrtDetCov(j) = sum(log(diag(L)));
91                  end
92
93                  mahalaD(:,j) = sum(xRinv.^2, 2);
94                  loglike(:,j) = - 0.5 * (mahalaD(:,j) + ...
                        2*logSqrtDetCov(j) + d * log(2 * pi));
95              end
96          end
97
98
99          %% Compute the posterior probability for each INFGMN gaussian ...
                component.
100         function computePosterior(self)
101             logPrior = log(self.priors);
102             self.post = bsxfun(@plus, self.loglike, logPrior);
103             maxll = max(self.post, [], 2);
104             % minus maxll to avoid underflow
105             self.post = exp(bsxfun(@minus, self.post, maxll));
106             density = sum(self.post, 2);
107             % normalize
108             self.post = bsxfun(@rdivide, self.post, density);
109             logpdf = log(density) + maxll;
110             self.dataLikelihood = self.dataLikelihood + sum(logpdf);
111         end
112
113         %% Check the INFGMN novelty criterion.
114         function h = hasAcceptableDistribution(self)
115             for j = 1:self.nc
116                 if (self.mahalaD(j) ≤ self.maxDist)
117                     h = true;
118                     return;
119                 end
120             end
121             h = false;
122         end
123
124         %% Create a gaussian component when a data point x matches the ...
                novelty criterion.
125         function createComponent(self, x)
126             self.nc = self.nc + 1;
```

```
127             self.means(self.nc,:) = x;
128             if self.nc > 1
129                 self.covs(:,:,self.nc) = ...
                        diag(mean(self.jdiag(self.covs), 3));
130             else
131                 self.covs(:,:,self.nc) = diag(self.sigma);
132             end
133             self.sps(1,self.nc) = 1;
134             self.st(1,self.nc) = 0;
135             self.sp(1,self.nc) = 0;
136             self.updatePriors();
137             [self.loglike(1,self.nc), self.mahalaD(1, self.nc)] = ...
138                 self.logmvnpdf(x, self.means(self.nc,:), ...
                        self.covs(:,:,self.nc));
139         end
140
141         %% Update the INFGMN gaussians priors.
142         function updatePriors(self)
143             if ¬self.uniform
144                 self.priors = self.sps ./ sum(self.sps);
145              else
146                 self.priors = ones(1,self.nc) ./ self.nc;
147             end
148         end
149
150         %% Update the INFGMN components parameters.
151         function updateComponents(self, x)
152             self.sps = self.sps + self.post;
153             self.st = self.st + 1.0;
154             self.sp = self.sp + self.post;
155             w = (self.post ./ self.sps) * self.beta;
156             for j = 1:self.nc
157                 Xcentered = x - self.means(j,:);
158                 ΔMU = w(j) * Xcentered;
159                 self.means(j,:) = self.means(j,:) + ΔMU;
160                 XcenteredNEW = x - self.means(j,:);
161                 self.covs(:,:,j) = self.covs(:,:,j) - (ΔMU")*(ΔMU) + ...
162                     w(j) * ((XcenteredNEW")*(XcenteredNEW) - ...
                            self.covs(:,:,j)) + self.regValue;
163                 for d = 1:length(self.varNames)
164                     if self.covs(d,d,j) < 0
165                         self.covs(d,d,j) = 1e-7;
166                     end
167                 end
168             end
169 %             self.covs(self.covs < 0) = 0.0000001;
170             % normalize the priors
```

```
171             self.updatePriors();
172         end
173 
174         %% Remove spurious gaussian components.
175         function removeSpurious(self)
176             for i = self.nc:-1:1
177                 if (self.st(i) ≥ self.tmax)
178                     if self.sp(i) < self.spmin
179                         self.nc = self.nc - 1;
180                         self.priors(i) = [];
181                         self.means(i,:) = [];
182                         self.covs(:,:,i) = [];
183                         self.st(i) = [];
184                         self.sps(i) = [];
185                         self.sp(i) = [];
186                         self.post(i) = [];
187                         self.mahalaD(i) = [];
188                         self.loglike(i) = [];
189                         if self.doMerge
190                             self.removeFromMerged(i);
191                         end
192                     else
193                         self.st(i) = 0;
194                         self.sp(i) = 0;
195                     end
196                 end
197             end
198         end
199 
200         function removeFromMerged(self, i_nc)
201             for i_vn = 1:length(self.varNames)
202                 self.FPstruct.(self.varNames{i_vn}).purge(i_nc);
203             end
204         end
205 
206         %% INFGMN recalling algorithm
207         function y = recall(self, x)
208             validateattributes(x, {"dataset"}, {"nonempty", }, ...
                    strcat(self.name, ":recall"),""x"");
209             varNames = x.Properties.VarNames; %#ok<PROPLC>
210             inputIndexes = cellfun( @(var)find(strcmp(self.varNames, ...
                    var)), varNames, "UniformOutput", 1); %#ok<PROPLC>
211             outputIndexes = 1:length(self.varNames);
212             outputIndexes(inputIndexes) = [];
213             self.updateFuzzyLayer(inputIndexes, outputIndexes);
214             if self.normalize
215                 y = ones(size(x, 1), length(self.varNames));
```

```
216                 y(:, inputIndexes) = double(x);
217                 y = mapminmax("apply", y", self.proportion)";
218                 if any(any(isnan(y(:, inputIndexes))))
219                     x_imp = nan(size(x, 1), length(self.varNames));
220                     x_imp(:, inputIndexes) = y(:, inputIndexes);
221                     x_imp = self.internal_imputation(x_imp);
222                     y(:, inputIndexes) = x_imp(:, inputIndexes);
223                 end
224                 y(:, outputIndexes) = evalfis(y(:, inputIndexes), ...
                        self.fis);
225                 y = mapminmax("reverse", y", self.proportion)";
226                 y = y(:, outputIndexes);
227             else
228                 y = double(x);
229                 if any(any(isnan(y)))
230                     x_imp = nan(size(x, 1), length(self.varNames));
231                     x_imp(:, inputIndexes) = y;
232                     x_imp = self.internal_imputation(x_imp);
233                     y = x_imp(:, inputIndexes);
234                 end
235                 y = evalfis(y, self.fis);
236             end
237             y = mat2dataset(y , "VarNames", self.varNames(outputIndexes));
238         end
239
240         % INFGMN internal_imputation method
241         function x = internal_imputation(self, x)
242             indexes = isnan(x);
243             if all(indexes)
244                 error("Each trainning input pattern must have at least ...
                        1 non nan column.");
245             end;
246             if any(any(indexes))
247                 N = size(x, 1);
248                 for j = 1:N
249                     pajs = zeros(self.nc, 1);
250                     curIndexes = indexes(j, :);
251                     xm = zeros(self.nc, sum(curIndexes));
252
253                     for i = 1:self.nc
254                         meanA = self.means(i, ¬curIndexes);
255                         meanB = self.means(i, curIndexes);
256                         covA = self.covs(¬curIndexes, ¬curIndexes, i);
257                         xm(i,:) = meanB;
258                         ll = self.logmvnpdf(x(j, ¬curIndexes), meanA, ...
                                covA);
259                         pajs(i) = exp(ll) * self.priors(i);
```

```
260                     end
261                     pajs = pajs ./ sum(pajs);
262                     x(j, curIndexes) = sum(bsxfun(@times, xm, pajs));
263                     x(j, x(j, :) > self.ranges(2, :)) = self.ranges(2, ...
                            x(j, :) > self.ranges(2, :));
264                     x(j, x(j, :) < self.ranges(1, :)) = self.ranges(1, ...
                            x(j, :) < self.ranges(1, :));
265                 end
266             end
267         end
268
269         % INFGMN imputation method
270         function y = imputation(self, x)
271             validateattributes(x, {"dataset"}, {"nonempty", "nrows", ...
                    1}, strcat(self.name, ":imputation"),""x"");
272             values = double(x);
273             indexes = isnan(values);
274             if all(indexes)
275                 error("Each trainning input pattern must have at least ...
                        1 non nan column.");
276             end;
277             if any(any(indexes))
278                 y = ones(1, length(self.varNames));
279                 y(¬indexes) = values(¬indexes);
280                 if self.normalize
281                     y = mapminmax("apply", y", self.proportion)";
282                 end
283                 pajs = zeros(self.nc, 1);
284                 xm = zeros(self.nc, sum(indexes));
285
286                 for i = 1:self.nc
287                     meanA = self.means(i, ¬indexes);
288                     meanB = self.means(i, indexes);
289                     covA = self.covs(¬indexes, ¬indexes, i);
290                     xm(i,:) = meanB;
291                     ll = self.logmvnpdf(y(¬indexes), meanA, covA);
292                     pajs(i) = exp(ll) * self.priors(i);
293                 end
294                 pajs = pajs ./ sum(pajs);
295                 y(indexes) = sum(bsxfun(@times, xm, pajs));
296                 if self.normalize
297                     y = mapminmax("reverse", y", self.proportion)";
298                 end
299                 y = mat2dataset(y(indexes) , "VarNames", ...
                        self.varNames(indexes));
300             end
301         end
```

```
302
303         % Calculate the indexes assignments for the data points in X for
304         % cluster analysis.
305         function idx = cluster(self, X)
306             N = size(X,1);
307             idx = zeros(N,1);
308             for i=1:N
309                 x = X(i,:);
310                 self.computeLikelihood(x);
311                 self.computePosterior();
312                 [¬,idx(i)] = max(self.post);
313             end
314         end
315
316
317         %% Accessor methods.
318         function ms = modelSize(self)
319             ms = size(self.fis.Rules, 2);
320         end
321
322         function covs = modelCovariances(self)
323             covs = self.covs;
324         end
325
326         function means = modelMeans(self)
327             means = self.means;
328         end
329
330         function weights = modelWeights(self)
331             weights = self.priors;
332         end
333
334         function spreads = modelSpreads(self)
335             if size(self.covs, 3) > 1
336                 spreads = squeeze(self.jdiag(self.covs))" .^ self.spread;
337             else
338                 spreads = squeeze(self.jdiag(self.covs)) .^ self.spread;
339             end
340         end
341
342         function fis = toFIS(self)
343             fis = self.fis;
344         end
345
346         function mfTypes = getMfTypes(self)
347             mfTypes = {self.inMfType, self.outMfType};
348         end
```

```
349
350         function diagcovs = jdiag(¬, covs)
351             diagcovs = zeros(1, size(covs,2), size(covs,3));
352             for j = 1:size(covs,3)
353                 diagcovs(:,:,j) = diag(covs(:,:,j))";
354             end
355         end
356
357         function setMergeStats(self, doMerge, newSimMerge, newSimRefit, ...
                newMaxMFs)
358             self.doMerge = doMerge;
359             self.simMerge = newSimMerge;
360             self.simRefit = newSimRefit;
361             self.maxMFs = newMaxMFs;
362             if self.doMerge
363                 self.refitFP();
364             end
365             self.needsFisUpdate = true;
366         end
367
368         function setFisType(self, newFisType)
369             if strcmp(newFisType, "mamdani")
370                 self.outMfType = "gaussmf";
371                 defuzzMethod = "centroid";
372             elseif strcmp(newFisType, "sugeno")
373                 self.outMfType = "linear";
374                 defuzzMethod = "wtaver";
375             else
376                 throw(MException(["MATLAB:" self.name ...
                        ":incorrectFisType"], ...
377                     ""newFisType" must be "mamdani" or "sugeno""));
378             end
379             self.fis = newfis(self.name, newFisType, "prod", "max", ...
                    "prod", "sum", defuzzMethod);
380             self.needsFisUpdate = true;
381         end
382
383         function setSpread(self, newSpread)
384             self.spread = newSpread;
385             self.needsFisUpdate = true;
386         end
387
388     end %end methods
389
390     methods (Access = private)
391
392         function updateFuzzyLayer(self, inputIndexes, outputIndexes)
```

```
393             self.needsFisUpdate = self.needsFisUpdate || ...
394                 isempty(self.fis.output) || ...
395                 ¬isempty(setdiff([self.fis.output(:).name], ...
                        self.varNames(outputIndexes))) || ...
396                 ¬isempty(setdiff(self.varNames(outputIndexes), ...
                        [self.fis.output(:).name]));
397             if (self.needsFisUpdate)
398                 self.cleanFis();
399                 self.createFuzzyVariables(inputIndexes, outputIndexes);
400                 self.createRules(inputIndexes);
401             end
402             self.needsFisUpdate = false;
403         end
404
405         function createFuzzyVariables(self, inputIndexes, outputIndexes)
406             for i = 1:length(inputIndexes)
407                 range = self.ranges(:, inputIndexes(i));
408                 varName = self.varNames{inputIndexes(i)};
409                 if  self.doMerge
410                     till = size(self.FPstruct.(varName).mergedMFs, 1);
411                 else
412                     till = self.nc;
413                     mus = self.means(:, inputIndexes(i));
414                     spreads = self.jdiag(self.covs(inputIndexes(i), ...
                            inputIndexes(i),:)) .^ self.spread;
415                 end
416                 self.fis = addvar(self.fis, "input", ...
417                     varName, [range(1), range(2)]);
418                 for j = 1:till
419                     if self.doMerge
420                         spd = self.FPstruct.(varName).mergedMFs(j, 1);
421                         mu  = self.FPstruct.(varName).mergedMFs(j, 2);
422                     else
423                         mu = mus(j);
424                         spd = spreads(j);
425                     end
426                     params = self.toMfParams("trimf", mu, spd, ...
                            self.inMfType);
427                     if self.fis.input(i).range(1) >  params(1)
428                         self.fis.input(i).range(1) = params(1) - 0.3;
429                     else
430                         if self.fis.input(i).range(2) <  params(3)
431                             self.fis.input(i).range(2) = params(3) + 0.3;
432                         end
433                     end
434                     if self.doMerge
435                         muy = ones(1, length(self.varNames));
```

```
436                         muy(inputIndexes(i)) = double(mu);
437                         muy = mapminmax("reverse", muy", self.proportion)";
438                         self.addInputMf(i, mu, spd, ["MF" num2str(j) " ...
                                about " num2str(muy(inputIndexes(i))) ])
439                     else
440                         self.addInputMf(i, mu, spd, ["MF" num2str(j)])
441                     end
442                 end
443             end
444             for i = 1:length(outputIndexes)
445                 range = self.ranges(:, outputIndexes(i));
446                 self.fis = addvar(self.fis, "output", ...
447                     self.varNames{outputIndexes(i)}, [range(1), range(2)]);
448             end
449             if strcmp(self.fis.type, "mamdani")
450                 for i = 1:length(outputIndexes)
451                     mus = self.means(:, outputIndexes(i));
452                     spreads = self.jdiag(self.covs(outputIndexes(i), ...
                            outputIndexes(i),:)) .^ self.spread;
453                     for j = 1:self.nc
454                         mu = mus(j);
455                         spd = spreads(j);
456                         params = self.toMfParams("trimf", mu, spd, ...
                                self.outMfType);
457                         if self.fis.output(i).range(1) >  params(1)
458                             self.fis.output(i).range(1) = params(1) - 0.3;
459                         else
460                             if self.fis.output(i).range(2) <  params(3)
461                                 self.fis.output(i).range(2) = params(3) ...
                                        + 0.3;
462                             end
463                         end
464                         self.addOutputMf(i, mu, spd, ["MF" num2str(j)])
465                     end
466                 end
467             else % sugeno
468                 if ¬strcmp(self.outMfType, "linear")
469                     throw(MException(["MATLAB:" self.name ...
                            ":errorOnSugenoOutputType"], ...
470                         "outputMF type must be linear (constant type ...
                                not supported yet)"));
471                 end
472                 for j = 1:self.nc
473                     muj = self.means(j,:);
474                     invcovj = pinv( self.covs(:,:,j) );
475                     for i = 1:length(outputIndexes)
476                         o = outputIndexes(i);
```

```
477                         CAngular = -invcovj(inputIndexes, o) / ...
                                invcovj(o, o);
478                         CLinear = muj(o) - muj(inputIndexes) * CAngular;
479                         mfName = ["MF" num2str(j) " " ...
480                             self.getEquationText(CLinear, CAngular, ...
                                    inputIndexes)];
481                         self.addOutputFunction(i, CLinear, CAngular, ...
                                mfName);
482                     end
483                 end
484             end
485         end
486
487         function createRules(self, inputIndexes)
488             numVars = length(self.varNames);
489             mfsTemplate = ones(1, numVars);
490             ruleList = ones(self.nc, numVars + 2);
491             for i = 1:self.nc
492                 ruleList(i, :) = [(mfsTemplate .* i) self.priors(i) 1];
493             end
494             if self.doMerge
495                 for inp = 1:length(self.fis.Inputs)
496                     varName = self.varNames{inputIndexes(inp)};
497                     for i_mf = 1:length(self.FPstruct.(varName).mergedIDXs)
498                         ruleList(self.FPstruct.(varName).mergedIDXs{i_mf}, ...
                                inp) = i_mf;
499                     end
500                 end
501                 [premisses,¬,idxs] = unique(ruleList(:, ...
                    1:length(self.fis.Inputs)), "rows");
502                 if size(premisses,1) < size(ruleList, 1)
503                     conclusions = zeros(size(premisses,1), ...
                        size(ruleList,2)-size(premisses,2)-2);
504                     weights = zeros(size(premisses,1), 1);
505                     for jj = 1:max(idxs)
506                         rows = (idxs == jj);
507                         jths_conclusions = ruleList(rows, ...
                            length(self.fis.Inputs)+1:end-2);
508                         jths_weights = ruleList(rows, end-1);
509                         weights(jj) = sum(jths_weights);
510                         conclusions(jj) = find(rows, 1, "first");
511
512                         oldParams = ...
                            vertcat(self.fis.Outputs.MembershipFunctions( ...
                            jths_conclusions).Parameters);
513                         newParams = sum(bsxfun(@times, jths_weights, ...
                            oldParams), 1) / weights(jj);
```

```
514                         self.fis.Outputs.MembershipFunctions( ...
                                conclusions(jj)).Parameters = newParams;
515                         self.fis.Outputs.MembershipFunctions( ...
                                conclusions(jj)).Name = ...
516                              ["MF" num2str(conclusions(jj)) " " ...
517                              self.getEquationText(newParams(end), ...
                                    newParams(1:end-1), inputIndexes)];
518                     end
519                     ruleList = [premisses conclusions weights ...
                            ones(size(weights))];
520                 end
521             end
522             self.fis = addrule(self.fis, ruleList);
523         end
524
525         function cleanFis(self)
526             self.fis.rule = [];
527             self.fis.output = [];
528             self.fis.input = [];
529         end
530
531         function addInputMf(self, index, mean, spread, mfName)
532             params = self.toMfParams(self.inMfType, mean, spread, ...
                        self.defaultFISOptions.inMfType);
533             self.fis = addmf(self.fis, "input", index, mfName, ...
                        self.inMfType, params);
534         end
535
536         function addOutputMf(self, index, mean, spread, mfName)
537             params = self.toMfParams(self.outMfType, mean, spread, ...
                        self.defaultFISOptions.outMfType);
538             self.fis = addmf(self.fis, "output", index, mfName, ...
                        self.outMfType, params);
539         end
540
541         function params = toMfParams(¬, mfType, mean, spread, ...
                    defaultMfType)
542             params = [spread mean];
543             if ¬strcmp(mfType, defaultMfType)
544                 params = mf2mf(params, defaultMfType, mfType);
545             end
546         end
547
548         function addOutputFunction(self, index, CLinear, CAngular, mfName)
549             params = [CAngular" CLinear];
550             self.fis = addmf(self.fis, "output", index, mfName, ...
                        self.outMfType, params);
```

```
551         end
552
553         function eqnTxt = getEquationText(self, CLinear, CAngular, ...
                inputIndexes)
554             eqnTxt = num2str(CLinear);
555             for i_coef = 1:length(CAngular)
556                 eqnTxt = [eqnTxt " " num2str(CAngular(i_coef), "%+.5g") ...
557                     "*" self.varNames{inputIndexes(i_coef)} ];
558             end
559         end
560
561     end
562
563 end % end class
```

```
1  classdef FuzzyPartition < handle
2
3      properties(Constant)
4          ALPHA = 0.5;
5          ALPHA_AUX = sqrt(-2 * log(FuzzyPartition.ALPHA));
6          I_SPD = 1;
7          I_MU = 2;
8      end
9
10     properties
11         age          (1,1) {mustBeReal, mustBeInteger, ...
               mustBeNonnegative} = 0;
12         simRefit     (1,1) {mustBeReal, mustBePositive, ...
               mustBeLessThanOrEqual(simRefit,1)} = 0.7^(1/0.7^2);
13         log2Smerge   (1,1) {mustBeReal, mustBeNonpositive} = ...
               log2(0.8)/0.8^2;
14         maxMFs       (1,1) {mustBeReal, mustBeInteger, mustBePositive} = 25;
15         mergedIDXs   (:,1) cell = cell(0,1); % {mustBeInteger, ...
               mustBePositive};
16         mergedLgSim  (:,1) {mustBeReal, mustBeNonpositive} = zeros(0,1);
17         mergedMFs    (:,2) {mustBeReal} = zeros(0,2);
18     end
19
20     methods(Access = private)
21
22         function log2Sim = log2PartitionSim(self)
23             log2Sim = sum(self.mergedLgSim);
24         end
25
26         function sim = partitionSim(self)
27             sim = 2^self.log2PartitionSim();
```

```
28          end
29
30          function len = totalMFs(self)
31              len = length(self.mergedIDXs);
32          end
33
34          function bool = time2Refit(self, NC)
35              bool = self.age > 5*(log2(NC)+1) ...
36                  || self.partitionSim() < self.simRefit;
37          end
38
39      end
40
41      methods
42
43          function self = FuzzyPartition(maxMFs, simMerge, simRefit, ...
                compMFs, weights)
44              self.maxMFs = maxMFs;
45              self.log2Smerge = log2(simMerge)/simMerge^2;
46              self.simRefit = simRefit^(1/simRefit^2);
47              components = self.calcAlphaSupport(compMFs, weights);
48              self.refitPartition(components);
49          end
50
51          function updateSystem(self, compMFs, weights, thereIsANewComponent)
52              components = FuzzyPartition.calcAlphaSupport(compMFs, weights);
53              minAfter = Inf;
54              for i_merged = length(self.mergedIDXs)-1:-1:1
55                  minAfter = min(minAfter, ...
                        min(components.MF(self.mergedIDXs{i_merged+1}, ...
                        self.I_MU)));
56                  if minAfter ≤ ...
                        max(components.MF(self.mergedIDXs{i_merged}, self.I_MU))
57                      self.refitPartition(components);
58                      return;
59                  end
60              end
61              maxBefore = -Inf;
62              for i_merged = 2:length(self.mergedIDXs)
63                  maxBefore = max(maxBefore, ...
                        max(components.MF(self.mergedIDXs{i_merged-1}, ...
                        self.I_MU)));
64                  if min(components.MF(self.mergedIDXs{i_merged}, ...
                        self.I_MU)) ≤ maxBefore
65                      self.refitPartition(components);
66                      return;
67                  end
```

```
68              end
69              self.updateAllMergedMFs(components);
70              self.updateAllmergedLgSim(components);
71              self.age = self.age +1;
72              if thereIsANewComponent
73                  self.age = self.age +1; % double-aging
74                  if self.time2Refit(length(components.weights))
75                      self.refitPartition(components);
76                      return;
77                  end
78                  self.fitNewComponent(components);
79              end
80              if self.time2Refit(length(components.weights))
81                  self.refitPartition(components);
82              end
83          end
84
85          function purge(self, idx)
86              for idxMerged = length(self.mergedIDXs):-1:1
87                  self.mergedIDXs{idxMerged} = ...
                        setdiff(self.mergedIDXs{idxMerged}, idx);
88                  if isempty(self.mergedIDXs{idxMerged})
89                      self.popMergedPropsAt(idxMerged);
90                  else
91                      filtro = self.mergedIDXs{idxMerged} > idx;
92                      self.mergedIDXs{idxMerged}(filtro) = ...
93                          self.mergedIDXs{idxMerged}(filtro) - 1;
94                  end
95              end
96          end
97
98      end
99
100     methods(Access = private)
101
102         function fitNewComponent(self, components)
103             newMF = components.MF(end, :);
104             idxNewMF = size(components.MF, 1);
105             for i_merged = 1:self.totalMFs()+1 % para cada MF da feature
106                 if i_merged ≤ self.totalMFs() && ...
107                         self.mergedMFs(i_merged, self.I_MU) < ...
                            newMF(self.I_MU)% se estiver antes do MF
108                     continue;
109                 end
110                 if i_merged ≤ self.totalMFs() && ...
111                         min(components.MF(self.mergedIDXs{i_merged}, ...
                                self.I_MU)) ≤ newMF(self.I_MU)
```

```
112                     idxNewMergedIDX = i_merged;
113                     self.mergedIDXs{idxNewMergedIDX}(end+1,1) = idxNewMF;
114                     self.updateIdxMergedMFs(components, idxNewMergedIDX);
115                     self.updateIdxmergedLgSim(components, idxNewMergedIDX);
116                 elseif i_merged > 1 && ...
117                         newMF(self.I_MU) ≤ ...
                            max(components.MF(self.mergedIDXs{i_merged-1}, ...
                            self.I_MU))
118                     idxNewMergedIDX = i_merged-1;
119                     self.mergedIDXs{idxNewMergedIDX}(end+1,1) = idxNewMF;
120                     self.updateIdxMergedMFs(components, idxNewMergedIDX);
121                     self.updateIdxmergedLgSim(components, idxNewMergedIDX);
122                 else % it is between
123                     self.putNewMergedIDXAt({idxNewMF}, i_merged, ...
                        components);
124                     self.tryMergeMFs(components);
125                 end
126                 break;
127             end
128         end
129
130         function tryMergeMFs(self, components)
131             if self.totalMFs() == 1
132                 return;
133             end
134             log2SimOnMerge = zeros(length(self.mergedIDXs)-1, 1);
135             for ii = 1:length(log2SimOnMerge) % para cada par de ...
                    componentes vizinhos
136                 log2SimOnMerge(ii) = self.log2SimilarityOnMergeIdx([ii, ...
                        ii+1], components);
137             end
138             log2SimImprov = log2SimOnMerge - (self.mergedLgSim(1:end-1) ...
                    + self.mergedLgSim(2:end));
139             [improv, idxMerged] = max(log2SimImprov);
140             while ¬isempty(log2SimImprov) && ...
141                     (   improv ≥ 0 || self.totalMFs() > self.maxMFs ...
142                     ||  self.log2PartitionSim() + improv > ...
                            self.log2Smerge )
143                 self.mergedIDXs{idxMerged} = ...
                        [self.mergedIDXs{idxMerged}; ...
                        self.mergedIDXs{idxMerged+1}];
144                 self.popMergedPropsAt(idxMerged+1);
145                 self.updateIdxMergedMFs(components, idxMerged);
146                 self.mergedLgSim(idxMerged) = log2SimOnMerge(idxMerged);
147                 log2SimOnMerge(idxMerged) = [];
148                 log2SimImprov(idxMerged) = [];
149                 if idxMerged > 1
```

```
150                     log2SimOnMerge(idxMerged-1) = ...
                            self.log2SimilarityOnMergeIdx([idxMerged-1, ...
                            idxMerged], components);
151                     log2SimImprov(idxMerged-1) = ...
                            log2SimOnMerge(idxMerged-1) ...
152                          - (self.mergedLgSim(idxMerged-1) + ...
                                 self.mergedLgSim(idxMerged));
153                 end
154                 if idxMerged < length(self.mergedIDXs)
155                     log2SimOnMerge(idxMerged) = ...
                            self.log2SimilarityOnMergeIdx([idxMerged, ...
                            idxMerged+1], components);
156                     log2SimImprov(idxMerged) = ...
                            log2SimOnMerge(idxMerged) ...
157                          - (self.mergedLgSim(idxMerged) + ...
                                 self.mergedLgSim(idxMerged+1));
158                 end
159                 [improv, idxMerged] = max(log2SimImprov);
160             end
161         end
162
163         function refitPartition(self, components)
164             [¬, sortedMu] = sort(components.MF(:, self.I_MU));
165             self.mergedIDXs = num2cell(sortedMu);
166             self.updateAllMergedMFs(components);
167             self.updateAllmergedLgSim(components);
168             self.tryMergeMFs(components);
169             self.age = 0;
170         end
171
172         function putNewMergedIDXAt(self, newMergedIDX, idxNewMergedIDX, ...
                components)
173             self.mergedIDXs = [ self.mergedIDXs(1:idxNewMergedIDX-1); ...
174                 newMergedIDX;   self.mergedIDXs(idxNewMergedIDX:end) ];
175             self.mergedMFs(idxNewMergedIDX+1:end+1, :) = ...
                    self.mergedMFs(idxNewMergedIDX:end, :);
176             self.mergedLgSim(idxNewMergedIDX+1:end+1, :) = ...
                    self.mergedLgSim(idxNewMergedIDX:end, :);
177             adjIdxNewMergedIDX = max(idxNewMergedIDX-1, ...
                    1):min(self.totalMFs(), idxNewMergedIDX+1);
178             self.updateIdxMergedMFs(components, adjIdxNewMergedIDX);
179             self.updateIdxmergedLgSim(components, adjIdxNewMergedIDX);
180         end
181
182         function popMergedPropsAt(self, idxMerged)
183             self.mergedIDXs(idxMerged) = [];
184             self.mergedLgSim(idxMerged) = [];
```

```
185             self.mergedMFs(idxMerged,:) = [];
186         end
187
188         function updateAllmergedLgSim(self, components)
189             self.mergedLgSim = zeros(size(self.mergedMFs,1), 1);
190             self.updateIdxmergedLgSim(components, ...
                    1:size(self.mergedMFs,1));
191         end
192
193         function updateIdxmergedLgSim(self, components, idxMergeds)
194             for idx = idxMergeds
195                 sim = self.similarity( self.mergedMFs(idx,:), ...
196                     components.MF(self.mergedIDXs{idx}, :) );
197                 weight = components.weights(self.mergedIDXs{idx});
198                 self.mergedLgSim(idx) = sum(weight .* log2(sim) ./ sim.^2);
199             end
200         end
201
202         function updateAllMergedMFs(self, components)
203             self.mergedMFs = zeros(length(self.mergedIDXs), 2);
204             self.updateIdxMergedMFs(components, 1:length(self.mergedIDXs));
205         end
206
207         function updateIdxMergedMFs(self, components, idxMergeds)
208             for idx = idxMergeds
209                 minXroot = min(components.alphaSupport( ...
                        self.mergedIDXs{idx}, 1 ));
210                 if idx > 1
211                     minXroot = max( minXroot, ...
212                         max(components.MF(self.mergedIDXs{idx-1}, ...
                                self.I_MU)) );
213                 end
214                 maxXroot = max(components.alphaSupport( ...
                        self.mergedIDXs{idx}, 2 ));
215                 if idx < length(self.mergedIDXs)
216                     maxXroot = min( maxXroot, ...
217                         min(components.MF(self.mergedIDXs{idx+1}, ...
                                self.I_MU)) );
218                 end
219                 self.mergedMFs(idx, :) = self.mergeAlphaSupport( [ ...
                        minXroot maxXroot ] );
220             end
221         end
222
223         function [log2sim, merged] = log2SimilarityOnMergeIdx(self, ...
                idxMergeds, components)
224             idxs = vertcat(self.mergedIDXs{idxMergeds});
```

```
225             minXroot = min(components.alphaSupport( idxs, 1 ));
226             if min(idxMergeds) > 1
227                 minXroot = max( minXroot, ...
228                     max(components.MF(self.mergedIDXs{min(idxMergeds)-1}, ...
                            self.I_MU)) );
229             end
230             maxXroot = max(components.alphaSupport( idxs, 2 ));
231             if max(idxMergeds) < length(self.mergedIDXs)
232                 maxXroot = min( maxXroot, ...
233                     min(components.MF(self.mergedIDXs{max(idxMergeds)+1}, ...
                            self.I_MU)) );
234             end
235             merged = FuzzyPartition.mergeAlphaSupport( [ minXroot ...
                    maxXroot ] );
236             sim = self.similarity( merged, components.MF(idxs, :) );
237             weight = components.weights(idxs);
238             log2sim = sum(weight .* log2(sim) ./ sim.^2);
239         end
240
241         function sim = similarity(self, mergedOne, subMFs)
242             mudiff = abs(mergedOne(self.I_MU) - subMFs(:, self.I_MU));
243             spdaux = [ mergedOne(self.I_SPD) + zeros(size(subMFs, 1), ...
                        1), ...
244                  subMFs(:, self.I_SPD) ];
245             spdmin = min( spdaux, [], 2 );
246             spdmax = max( spdaux, [], 2 );
247             spddiff = spdmax - spdmin;
248             spdsum  = spdmax + spdmin;
249             sqrtneglnw = mudiff ./ spddiff;
250             sqrtneglnv = mudiff ./ spdsum; % v = possibility
251             ohm = spddiff .* erf(sqrtneglnw) ...
252                 - spdsum  .* erf(sqrtneglnv);
253             sim  = (2 * spdmin + ohm) ...
254                 ./ (2 * spdmax - ohm);
255             sim(mudiff==0 & spddiff==0) = 1;
256             assert(¬any(isnan(sim)));
257             lim_inf = 1e-15;
258             sim(-1e-15 < sim & sim < lim_inf) = lim_inf;
259             assert(all(sim > 0));
260         end
261
262     end
263
264     methods(Static, Access = private)
265
266         function components = calcAlphaSupport(compMFs, weights)
267             components.MF = compMFs;
```

```
268             components.weights = weights(:);
269             aux = components.MF(:, FuzzyPartition.I_SPD) * ...
                    FuzzyPartition.ALPHA_AUX;
270             components.alphaSupport = [ ...
271                 ( components.MF(:, FuzzyPartition.I_MU) - aux ), ...
272                 ( components.MF(:, FuzzyPartition.I_MU) + aux ) ];
273         end
274
275         function merged = mergeAlphaSupport(alphaSupports)
276             minXroot = min(alphaSupports(:, 1));
277             maxXroot = max(alphaSupports(:, 2));
278             newMu = (minXroot + maxXroot) / 2;
279             newSigma = (maxXroot - newMu) / FuzzyPartition.ALPHA_AUX;
280             newSigma = max(newSigma, 1e-7);
281             merged = [newSigma, newMu];
282         end
283
284     end
285 end
```

APÊNDICE B – ARTIGO

Método para aumento de interpretabilidade em modelos de Takagi-Sugeno no sistema INFGMN

César Eduardo Corrêa[1], **Mauro Roisenberg**[1]

[1]Departamento de Informática e Estatística - Universidade Federal de Santa Catarina (UFSC) – Florianópolis - SC – Brasil

cesarecorrea94@gmail.com, mauro.roisenberg@ufsc.br

Abstract. Interpretability and accuracy are conflicting objectives in machine learning models, where improvement in one usually worsens the other. The interpretability of the generated models is important so that it can be validated by experts in the field and pass reliability to address the problem. This work seeks to expand the functionalities of the INFGMN, a neuro-fuzzy system, providing its application with Takagi-Sugeno models, and improving its interpretability via distinguishability in fuzzy partitions, achieving a good trade-off between accuracy and interpretability. In order to guarantee distinguishability, a fuzzy partition similarity method was developed to guide the merging and separation of fuzzy sets, weighted by the weights of the rules to cause less changes in the sets whose rules better describe the system and are not derived from outliers. The results presented for *offline* problems are promising.

Resumo. Interpretabilidade e acurácia são objetivos conflitantes em modelos de aprendizado de máquina, onde a melhora de um geralmente causa a piora do outro. A interpretabilidade dos modelos gerados é importante para que possa ser validado por especialistas do domínio e passar confiabilidade para tratar o problema. Neste trabalho busca-se expandir as funcionalidades do INFGMN, um sistema neuro-fuzzy, proporcinando sua aplicação com modelos de Takagi-Sugeno, e melhorando sua interpretabilidade via distinguibilidade nas partições fuzzy, obtendo um bom equilíbrio entre acurácia e interpretabilidade. Para garantia de distinguibilidade, foi elaborado um método de similaridade de partição fuzzy para guiar a fusão e separação dos conjuntos fuzzy, ponderado pelos pesos das regras para gerar menores alterações nos conjuntos cujas regras descrevam melhor o sistema e não sejam oriundas de dados discrepantes. Os resultados apresentados para problemas *offline* são promissores.

1 Introdução

Muitas técnicas de aprendizado de máquina geram modelos do tipo black-box, que não explicam o mapeamento entre entrada e saída. A interpretabilidade dos modelos gerados é importante para que possa ser validado por especialistas do domínio e passar confiabilidade para tratar o problema. Pesquisadores então trabalharam na construção de modelos que aprendiam com dados de treinamento para aquisição de conhecimento. [MAZZUTTI et al., 2017]

Sistemas de Inferência Fuzzy (FIS) utilizam da lógica fuzzy para modelar sistemas através de regras que manipulam variáveis linguísticas, cujos valores são palavras ao invés de números. Embora palavras sejam imprecisas, isso ajuda a lidar com a imprecisão do mundo e torna o sistema interpretável por imitar o pensamento humano. [MAZZUTTI et al., 2017; MATHWORKS, 2019]

Redes NeuroFuzzy incorporam a capacidade de aprendizado automático das redes neurais com a capacidade de representação de conhecimento dos sistemas fuzzy através de regras. INFGMN é um sistema neurofuzzy que utiliza da equivalência entre um Modelo de Mistura de Gaussianas (GMM) e um Sistema de Inferência Fuzzy (FIS) para extração de conhecimento na forma de regras de Mamdani-Larsen. [MAZZUTTI et al., 2018]

Modelos de Mamdani-Larsen (ML) utilizam regras fuzzy cujas partes do antecedente e consequente fazem uso somente de termos linguísticos (palavras), o que torna-os mais interpretáveis. Já em modelos de Takagi-Sugeno (TS), o consequente é uma função das variáveis de entradas, tornando-os mais precisos. [MATHWORKS, 2019; CASILLAS et al., 2003]

Interpretabilidade e acurácia são objetivos conflitantes, i.e, a melhora em um geralmente causa a piora do outro, gerando o dilema Acurácia-Interpretabilidade. Uma abordagem comum para obter o melhor equilíbrio entre os dois é definir um objetivo principal e buscar formas de suprir o que lhe falta. Na Modelagem Fuzzy Linguística (LFM), como modelos de Mamdani-Larsen, o objetivo é obter modelos fuzzy com boa interpretabilidade, enquanto na Modelagem Fuzzy Precisa (PFM), como modelos de Takagi-Sugeno, o objetivo é obter modelos fuzzy com boa acurácia. [CASILLAS et al., 2003; ZHOU; GAN, 2008]

Zhou e Gan (2008) definem uma taxonomia para a interpretabilidade em sistemas fuzzy, separando nos conceitos de interpretabilidade de baixo-nível e de alto-nível. A de baixo-nível é alcançada no nível de conjuntos fuzzy, otimizando as funções de pertinência, enquanto a de alto-nível é alcançada no nível de regras fuzzy, conduzindo uma redução na complexidade geral, como exemplo, a redução do número de regras.

Dentre os critérios semânticos para alcançar interpretabilidade de baixo-nível está a distinguibilidade, em que para cada espaço de entrada, os conjuntos fuzzy devem definir

claramente seus intervalos de pertinência e serem suficientemente distintos um do outro para poderem ser representados por um termo linguístico com um significado semântico claro. Uma forma de alcançar distinguibilidade se dá pela fusão de conjuntos fuzzy, sendo guiado, por exemplo, por medidas de similaridade entre conjuntos fuzzy. [ZHOU; GAN, 2008]

O objetivo do trabalho a que se refere este artigo é expandir as funcionalidades do sistema INFGMN, proporcinando sua aplicação com modelos de Takagi-Sugeno e melhorando sua interpretabilidade via distinguibilidade nas partições fuzzy. Isso concederá ao INFGMN uma nova abordagem pelo uso de uma Modelagem Fuzzy Precisa e a melhora de sua interpretabilidade, obtendo um bom equilíbrio entre acurácia e interpretabilidade.

2 Similaridade entre Gaussianas

Segundo Mencar, Castellano e Fanelli (2007), a distinguibilidade pode ser quantificada por meio de medidas de similaridade e, na análise de interpretabilidade, a mais comumente adotada é a da equação 1, onde frequentemente o mínimo e máximo são usados como T-norma e T-conorma, permitindo ser reescrita como a equação 2. Uma medida para calcular a distinguibilidade seria então definida como o complemento da similaridade: $\Delta(A,B) = 1 - S(A,B)$.

$$S(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

$$S(A,B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{2}$$

Hefny (2007), utilizando-se da medida da equação 2, elabora o cálculo para funções gaussianas, chegando as equações 3 a 9.

$$S(A,B) = \frac{\Psi}{2 - \Psi} \tag{3}$$

$$\Psi = \beta + (1-\beta) erf\left[\left(\frac{1}{1-\beta}\right)\sqrt{-ln(v)}\right] - erf\left(\sqrt{(}-ln(v))\right) \tag{4}$$

$$v = \Pi(A,B) = exp\left[-\left(\frac{\mu_A - \mu_B}{\sigma_A + \sigma_B}\right)^2\right] \tag{5}$$

$$\beta = \frac{2\sigma_{min}}{\sigma_{max} - \sigma_{min}} \tag{6}$$

$$\sigma_{max} = max(\sigma_A, \sigma_B) \tag{7}$$

$$\sigma_{min} = min(\sigma_A, \sigma_B) \tag{8}$$

$$erf(x) = \frac{2}{\sqrt{(}\pi)} \int_0^x e^{-t^2} dt \tag{9}$$

3 GMM (Modelo de Mistura de Gaussianas)

Mazzutti, Roisenberg e Filho (2018) mencionam que um GMM é uma PDF (Função Densidade de Probabilidade) representada pela soma ponderada de suas componentes gaussianas, e sua equação é:

$$G(x; p, \mu, C) = \sum_{j=1}^{J} p(j) N^D(x; \mu_j, C_j) \tag{10}$$

tal que:

$$0 \leq p(j) \leq 1, \forall j \tag{11}$$

$$\sum_{j=1}^{J} p(j) = 1 \tag{12}$$

onde D é a dimensão do modelo, J é o número de componentes gaussianas no modelo, x é o vetor D-dimensional de entrada, p é o vetor J-dimensional de proporções, $p(j)$ é a proporção da j-ésima componente gaussiana, μ_j é o vetor D-dimensional de médias da j-ésima componente, μ é a matriz $(D \times J)$ dos vetores de médias concatenados, C_j é a matriz $(D \times D)$ de covariância da j-ésima componente, C é a matriz $(D \times DJ)$ das matrizes de covariância concatenados, e $N^D(x; \mu_j, C_j)$ é a função densidade normal multivariada (D-dimensional) da j-ésima componente.

4 IGMM (Modelo de Mistura de Gaussianas Incremental)

Segundo Engel e Heinen (2010), o IGMM foi projetado para aprender um GMM a partir de um fluxo de dados de maneira incremental e não-supervisionada. Ele consegue lidar com o dilema estabilidade-plasticidade, i.e, se um novo dado deve ser assimilado pelo modelo, ou causar uma mudança estrutural para acomodar a nova informação. Com isso, novas componentes são adicionadas à mistura sob demanda, através de um critério de novidade.

O IGMM contém 4 parâmetros: σ_{ini} é um parâmetro de configuração da variância inicial de cada componente; τ_{nov} é o critério de novidade, e define o quão distante um vetor de entrada tem de estar de cada componente para a criação de uma nova; t_{max} e sp_{min} definem quando uma componente não é mais necessária para o modelo, ocorrendo sua remoção quando $t_j > t_{max}$ e $sp_j < sp_{min}$, sendo sp_j o número de instâncias assimiladas pela componente j, e t_j o tempo que ela levou para isso. [ENGEL; HEINEN, 2010; MAZZUTTI et al., 2018]

As probabilidades a priori, $p(j)$ (proporção da componente), são ajustadas e normalizadas para satisfazer as equações 11 e 12 de uma GMM, e adicionalmente a equação 13,

onde $p(x|j)$ é a probabilidade de se observar um vetor x pertencente à componente j, e é computada como a PDF de j, i.e., $p(x|j) = N^D(x;\mu_j,C_j)$ [MAZZUTTI et al., 2018]

$$\int_{-\infty}^{+\infty} p(x|j)dx = 1 \tag{13}$$

Quando um vetor x é reconhecido como não pertencente a nenhuma componente j, conforme a equação 14, uma nova componente com média $\mu_j^* = x$ e matriz de covariância $C_j^* = \sigma_{ini}$ é adicionada à mistura.

$$p(x|j) < \frac{\tau_{nov}}{(2\pi)^D \mid C_j \mid}, \forall j \tag{14}$$

Para a assimilação de novos dados, o IGMM segue uma versão incremental do processo iterativo de 2 passos do algoritmo EM (Estimation/Maximization). No passo E, estima-se a probabilidade de pertinência de um vetor x a cada uma das componentes, conforme o Teorema de Bayes, pela equação 15. No passo M, os parâmetros de cada componente são ajustados para assimilarem o novo dado, conforme as equações 16 a 19, onde $sp_j^*, \mu_j^*, C_j^*, p(j)^*$ são os novos valores de $sp_j, \mu_j, C_j, p(j)$, respectivamente.

$$p(j|x) = \frac{p(x|j)p(j)}{\sum_{j=1}^{J} p(x|j)p(j)}, \forall j \tag{15}$$

$$sp_j^* = sp_j + p(j|x) \tag{16}$$

$$\mu_j^* = \mu_j + \frac{p(j|x)}{sp_j^*}(x - \mu_j) \tag{17}$$

$$\begin{aligned} C_j^* =& C_j - (\mu_j^* - \mu_j)(\mu_j^* - \mu_j)^T \\ &+ \frac{p(j|x)}{sp_j^*}\left[(x - \mu_j^*)(x - \mu_j^*)^T - C_j\right] \end{aligned} \tag{18}$$

$$p(j)^* = \frac{sp_j^*}{\sum_{q=1}^{J} sp_q^*} \tag{19}$$

5 Equivalência GMM e FIS

Gan, Hanmandlu e Tan (2005) descrevem um GFM (Modelo Fuzzy Generalizado) da forma:

$$R^k: \text{ IF } x_1 \text{ is } A_1^k \wedge \cdots \wedge x_D \text{ is } A_D^k \text{ THEN } y \text{ is } B^k(f^k(x), v_k) \tag{20}$$

onde A_i^k é um subconjunto fuzzy da i-ésima variável de entrada x_i na k-ésima regra R^k de peso v_k, e B^k é um subconjunto fuzzy de saída com centróide variante numa função $f^k(x)$.

Para modelos fuzzy com agregação aditiva, e conjunção e implicação multiplicativas (condições para a equivalência), a fórmula para obter a saída defuzzificada é: [GAN et al., 2005]

$$y^o = \sum_{k=1}^{K} \frac{\lambda^k(x) v_k}{\sum_{k'=1}^{K} \lambda^{k'}(x) v_{k'}} f^k(x) \tag{21}$$

Considerando um caso MISO (Multiple-Input Single-Output) com vetor de entrada x e saída y, Gan, Hanmandlu e Tan (2005) derivaram de um GMM a esperança de y condicionada a x como:

$$E[y|x] = \frac{\sum_{j=1}^{J} p(j) \prod_{d=1}^{D} N^D(x_d; \mu_{j,d}, \sigma_{j,dd}) \left(\mu_{j,y} - \frac{[x-\mu_{j,x}]' \sigma_j^{xy}}{\sigma_j^{yy}} \right)}{\sum_{j'=1}^{J} p(j') \prod_{d'=1}^{D} N^D(x_d; \mu_{j',d}, \sigma_{j',dd})} \tag{22}$$

onde $\sigma_{j,xx}$ é uma partição da matriz de covariância, C, enquanto σ_j^{xy} e σ_j^{yy} são partições de sua inversa.

Assim, Gan, Hanmandlu e Tan (2005) demonstraram que as equações 21 e 22 são equivalentes sob as seguintes condições: (i) O número de regras, K, na base de regras do GFM é igual ao número de componentes, J, no GMM. (ii) O peso de cada regra, v_k, é dado pela correspondente probabilidade a priori, $p(j)$, do modelo de mistura. (iii) A função de regressão no consequente das regras do GFM é linear, e é uma função de todas as variáveis de entrada, dada pela equação 23. (iv) A função de pertinência de cada uma das variáveis do antecedente das regras é uma função gaussiana. (v) O sistema fuzzy é aditivo, com conjunção e implicação multiplicativas.

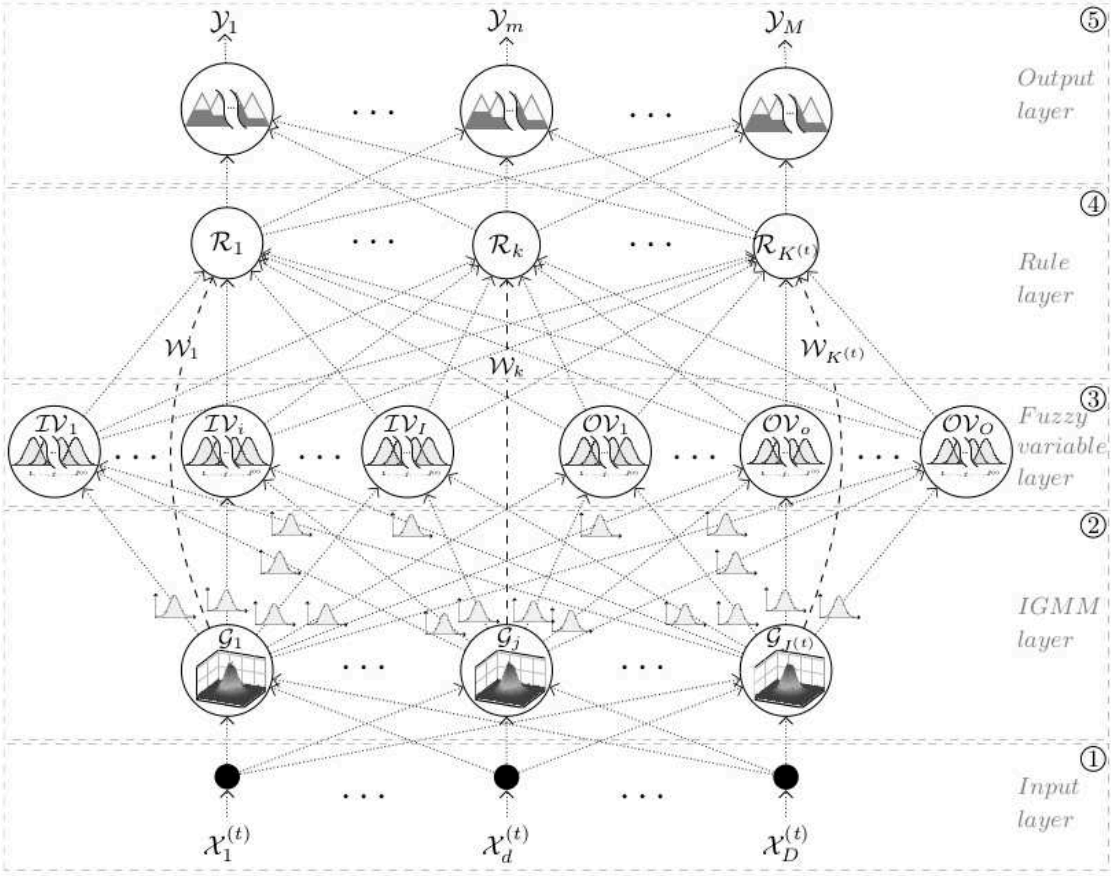
$$f^k(x) = \mu_{j,y} - \frac{[x-\mu_{j,x}]^t \sigma_j^{xy}}{\sigma_j^{yy}} \tag{23}$$

6 INFGMN (Incremental Neuro-Fuzzy Gaussian Mixture Network)

Mazzutti, Roisenberg e Filho (2018) descrevem que o INFGMN é um NFS (Sistema Neuro-Fuzzy) que se utiliza da equivalência entre GMM e ML para geração de um FIS, e do IGMM para fornecer a capacidade de aprendizado incremental. O INFGMN possui 2 modos de operação, *learning* e *recalling*, que não precisam ocorrer separadamente, isto é, aprendizado e utilização podem ser intercalados. Sua arquitetura, representada na figura 1, possui 5 camadas de rede, e cada uma desempenha um papel bem específico.

Na camada 1 (Input layer) é apresentado o vetor D-dimensional das variáveis de entradas da rede, onde cada nó representa uma entrada, e estas são repassadas a camada 2 sem alteração.

Figura 1 – Arquitetura da rede INFGMN.



Fonte – Mazzutti, Roisenberg e Filho (2018)

Na camada 2 (IGMM layer) são mantidas as J componentes gaussianas, G_j, que são criadas e ajustadas de acordo com as equações recursivas do IGMM. As PDFs marginais atualizadas são transmitidas para a camada 3, e as novas proporções das componentes, p, são transmitidas diretamente a camada 4 como vetor de pesos, W. Esta camada não é ativada no modo recalling.

Na camada 3 (Fuzzy variable layer) são mantidas as variáveis linguísticas de entrada e saída, IV_i e OV_o, e seus respectivos termos linguísticos. Cada termo linguístico possui função de pertinência gaussiana cujos parâmetros correspondem à PDF marginal da j-ésima componente na dimensão i ou o. No modo learning, termos linguísticos podem ser criados, removidos e/ou atualizados, e seus parâmetros são ajustados de acordo com a média e covariância das componentes geradas na camada 2. No modo recalling, esta camada é responsável por fuzzificar as variáveis de entrada, calculando suas pertinências a cada termo linguístico das partições de entrada.

Na camada 4 (Rule layer) são mantidas as regras fuzzy do tipo Mamdani-Larsen. No modo learning, regras podem ser criadas, removidas e/ou atualizadas de acordo com as componentes gaussianas, onde o número de regras $K^{(t)}$ é igual ao número de componentes de mistura $J^{(t)}$, no tempo t. Cada regra R^k é moldada com antecedentes e consequentes cujos termos linguísticos, gerados na camada 3, vieram da j-ésima componente gaussiana (onde $j = k$), cuja proporção, $p(j)$, será o peso da regra, W_k.

No modo recalling, o grau de ativação de cada regra, α_k, é computado como uma conjunção multiplicativa dos graus de pertinência de cada termo antecedente da regra, e então multiplicado pelo peso da regra, W_k. Com o grau de ativação, calcula-se a implicação multiplicativa para o consequente, e gera-se um conjunto fuzzy de saída para a regra. Os conjuntos fuzzy de saída são então agregados utilizando a agregação aditiva, gerando um único conjunto fuzzy para a variável de saída.

A camada 5 (Output layer), ativada somente no modo recalling, é responsável por defuzzificar o conjunto fuzzy de saída utilizando-se do método centróide.

7 Proposta

Para expandir as funcionalidades do sistema INFGMN, proporcinando sua aplicação com modelos de Takagi-Sugeno e melhorando sua interpretabilidade via distinguibilidade nas partições fuzzy, houveram mudanças arquiteturais nas camadas 3, 4 e 5.

A Camada 3 (Fuzzy Variable Layer) é composta pelas variáveis linguísticas fuzzy de entrada com termos linguísticos fuzzy, pelas variáveis de saída com funções lineares, e é ativada tanto para o modo de operação learning quanto para o modo recalling.

No modo de operação learning, os antecedentes e os consequentes de cada regra fuzzy são atualizados após cada padrão de treino em cada etapa de treinamento. Os antecedentes, definidos pelos termos linguísticos fuzzy das variáveis de entrada, tem seu significado representado por conjuntos fuzzy com função de pertinência Gaussiana cujos parâmetros $\mu_{i(k_i)}$ e $\sigma_{i(k_i)}$ são definidos pelo processo de fusão e separação de conjuntos fuzzy, conforme as equações 24 a 28, onde $\alpha_{\text{-cut}(M)}$ é a união dos $\alpha_{\text{-cut}}$ dos conjuntos fuzzy fundidos, similar ao método utilizado por Tung e Quek (2009), e $\alpha_{\text{-cut}(M_S)}$ o intervalo resultante após o processo de separação dos conjuntos fuzzy, de forma a não se sobreporem

aos núcleos dos demais conjuntos em um nível de pertinência maior que 0.5.

$$\alpha = 0.5 \tag{24}$$

$$\alpha_{\text{-cut}(M)} = \left[\min_{j \in M} \left(\mu_{i(j)} - \sigma_{i(j)} \cdot \sqrt{-2 \ln \alpha} \right), \max_{j \in M} \left(\mu_{i(j)} + \sigma_{i(j)} \cdot \sqrt{-2 \ln \alpha} \right) \right] \tag{25}$$

$$\alpha_{\text{-cut}(M_S)} = \left[\max \left(\min(\alpha_{\text{-cut}(M)}), \max_{j \in M-1} (\mu_{i(j)}) \right), \min \left(\max(\alpha_{\text{-cut}(M)}), \min_{j \in M+1} (\mu_{i(j)}) \right) \right] \tag{26}$$

$$\mu_{i(k_{i(M)})} = \frac{\max(\alpha_{\text{-cut}(M_S)}) + \min(\alpha_{\text{-cut}(M_S)})}{2} \tag{27}$$

$$\sigma_{i(k_{i(M)})} = \frac{\max(\alpha_{\text{-cut}(M_S)}) - \min(\alpha_{\text{-cut}(M_S)})}{2 \cdot \sqrt{-2 \ln \alpha}} \tag{28}$$

O processo de fusão começa por aqueles que forneçam a maior similaridade para a partição fuzzy do espaço de entrada, prosseguindo enquanto a fusão produzir uma partição mais similar que a separação, enquanto a similaridade da partição estiver acima de $simMerge$, e até que o número máximo de conjuntos fuzzy, $maxMFs$, seja satisfeito. A similaridade da partição fuzzy do espaço de entrada $IV_{(i)}$ é calculada conforme a equação 29, onde $S(A,B)$ é a similaridade entre os conjuntos A e B, conforme as equações 3 a 9 de Hefny (2007), $\lambda_{i(j)}$ é o conjunto original da componente gaussiana, $\lambda_{i(k_{i(j)})}$ é seu conjunto aproximado pela fusão/separação, e W_j é o peso da componente. Cada j-ésimo termo é ponderado pelo peso da respectiva componente, W_j, e pelo termo Z_j, que agrava os termos com similaridade menores.

$$S(IV_{(i)}) = \prod_{j=1}^{J^{(t)}} S(\lambda_{i(k_{i(j)})}, \lambda_{i(j)})^{W_j \cdot Z_j} \tag{29}$$

$$Z_j = \left(\frac{1}{S(\lambda_{i(k_{i(j)})}, \lambda_{i(j)}))} \right)^2 \tag{30}$$

Os consequentes são definidos por funções lineares de saída, cujos coeficientes angulares são definidos por $A_{o(j)} = -C_{(j)}^{Io} \div C_{(j)}^{oo}$, e o coeficiente linear por $L_{o(j)} = \mu_{o(j)} + \mu_{I(j)}^T A_{o(j)}$, onde $C_{(j)}^{Io}$ e $C_{(j)}^{oo}$ são partições da inversa da matriz de covariância da j-ésima componente, conforme as condições de Gan, Hanmandlu e Tan (2005).

Para o modo de operação recalling, esta camada tem a responsabilidade de fuzzificar os dados do vetor de entrada e calcular o resultado de cada função linear de saída.

A camada 4 (Rule layer) é composta pelas regras fuzzy da INFGMN, e é ativada para ambos os modos de operação, learning e recalling.

Durante o modo de operação learning, as regras fuzzy são atualizadas a partir dos novos antecedentes e consequentes gerados na camada 3. Regras inconsistentes (com

os mesmos antecendentes) serão fundidas, com o novo peso de regra definido como $W_{k(M)} = \sum_{j \in M} p(j)$, e o novo consequente como $OV_{o(k(M))} = \frac{\sum_{j \in M} p(j) \cdot OV_{o(j)}}{W_{k(M)}}$, onde $p(j)$ é a probabilidade à priori da j-ésima componente, e M contém os índices das respectivas regras fundidas. O número de regras geradas pode então ser igual ou menor que o número de componentes de mistura na camada 2 (IGMM Layer).

No modo de operação recalling, esta camada tem a função de avaliar as regras fuzzy da INFGMN. O conjunto de graus de ativação α_k é calculado através da aplicação de uma conjunção fuzzy multiplicativa a cada antecedente das regras, e então multiplicado pelo peso da regra, W_k. Com α_k, calcula-se a implicação fuzzy multiplicativa, alterando a pertinência da saída de cada regra. E então, a saída das regras são agregadas através de uma agregação fuzzy aditiva.

A Camada 5 é ativada somente durante o modo de operação *recalling* e é responsável por *defuzzificar* a saída da INFGMN conforme o método *Weighted Average*.

8 Resultados

Para validar o modelo proposto, foram realizados 2 experimentos. O primeiro foi a identificação online de um sistema dinâmico não linear com características que variam no tempo. Os resultados deste experimento, mostrados na Tabela 1, demonstraram que o modelo de TS na INFGMN obteve um bom desempenho no número de regras, e a C-INFGMN garantiu distinguibilidade na partição fuzzy ao custo de aproximadamente 5% de sua acurácia. Embora a acurácia dos modelos de TS foi similar ao de ML em Mazzutti, Roisenberg e Filho (2018), enquanto esperava-se uma maior precisão com sistemas Takagi-Sugeno ao custo da interpretabilidade no consequente das regras, seu RMSE chegou a aproximadamente 50% do modelo de ML nos momentos em que adequou-se aos distúrbios, sugerindo ser uma boa alternativa para ambientes estáveis.

O segundo experimento foi para avaliar as habilidades da rede em problemas *offline*, utilizando 4 conjuntos de dados UCI com problemas de regressão, a saber: 1) *Abalone*; 2) *Boston Housing*; 3) *Concrete Compressive Strength*; e 4) *Concrete Slump Test*. Os resultados foram promissores, obtendo bons desempenhos em termos de acurácia, embora o número de regras tenha sido alto para 2 dos 4 conjuntos. A C-INFGMN foi capaz de

Tabela 1 – Resultados experimentais no Sistema Dinâmico Não-Linear

Método	Tipo	#regras	RMSE
eFSM	ML	21	0.1
INFGMN [MAZZUTTI et al., 2018]	ML	15.5	0.06
INFGMN	TS	13.87	0.0538
C-INFGMN	TS	13.03	0.0564

Tabela 2 – Resultados experimentais para os conjuntos de dados UCI.

Método	#regras ($\mu \pm \sigma$)	Treino ($\mu \pm \sigma$)	Teste ($\mu \pm \sigma$)
Abalone			
HFC	10	2.449	2.685
	20	2.126	2.269
GON	10	2.315	2.391
	20	2.161	2.214
Tsekouras (2016)	10	2.056	2.224
	5	2.166	2.293
Constrained PSO (tipo TS)	6	2.072 ± 0.026	2.212 ± 0.022
INFGMN (tipo TS)	8.9 ± 2.514	2.129 ± 0.035	2.173 ± 0.045
C-INFGMN (tipo TS)	8 ± 2.108	2.146 ± 0.037	2.162 ± 0.040

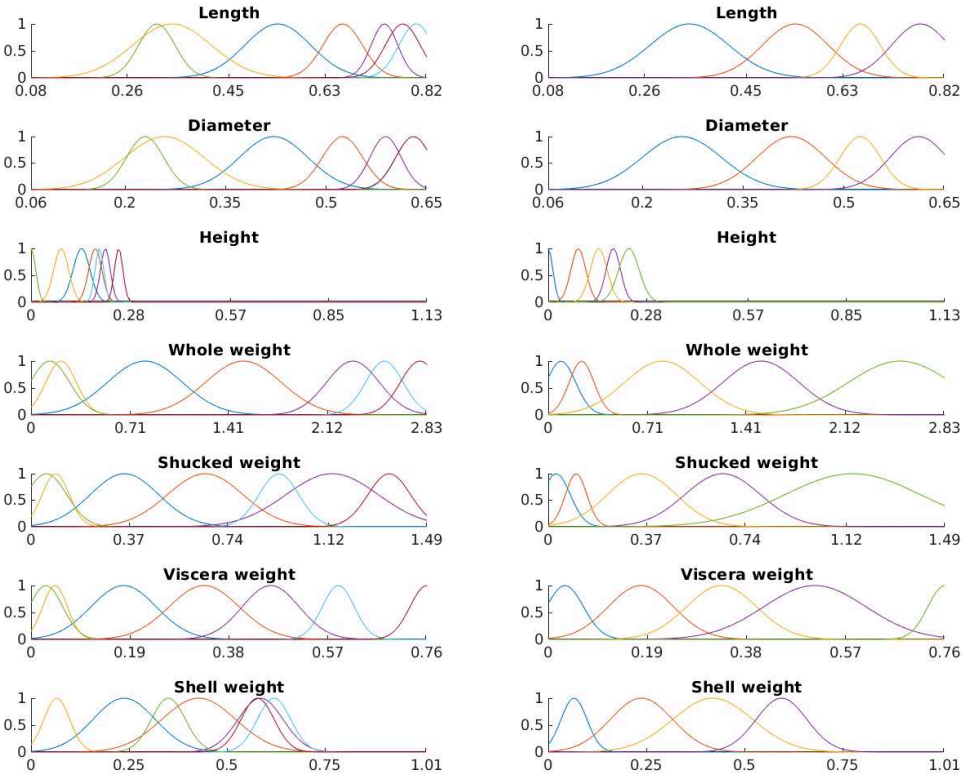
gerar partições fuzzy distinguíveis e, ademais, entregou uma redução de *overfitting* em todos os 4 conjuntos de dados. O desempenho no conjunto *Abalone* está na Tabela 2, e os conjuntos fuzzy gerados numa das simulações com o *Abalone* estão representados na figura 2.

O método de similaridade utilizado, avaliando toda a partição fuzzy, e considerando o peso das regras para guiar a fusão ou separação dos conjuntos fuzzy, demonstrou-se bem eficaz para os problemas *offline* pelo que se observou na redução do overfitting. Quanto ao experimento *online*, ela gerou uma instabilidade nos momentos de inserção/remoção do distúrbio, provavelmente devido ao baixo peso das novas componentes na medida de similaridade, causando-lhes maiores modificações em prol das componentes já estabelecidas e que, em tal experimento, tornaram-se obsoletas num incremento do tempo. Para ambientes mais estáveis ou cujas mudanças sejam mais sutis no tempo, a C-INFGMN pode ser uma alternativa.

9 Considerações finais

Este trabalho conferiu à INFGMN a capacidade de manipular modelos de Takagi-Sugeno, garantir a distinguibilidade nas partições fuzzy, e obter um número moderado de conjuntos. Porém, devido à falta de um método de tuning para as partições fuzzy obtidas com as restrições, a capacidade de obter um número moderado de conjuntos na partição fuzzy deve ser utilizada com parcimônia.

Diferente dos trabalhos relacionados, que avaliavam a similaridade entre 2 conjuntos isolados para guiar o processo de fusão, neste trabalho foi elaborado um método de similaridade que avalia a partição fuzzy para guiar a fusão/separação dos conjuntos fuzzy, ponderado pelos pesos das regras para gerar menores alterações nos conjuntos cujas regras descrevam melhor o sistema e não sejam oriundas de dados discrepantes, dessa forma mantendo a acurácia no sistema fuzzy aproximado.



(a) INFGMN (b) C-INFGMN

Figura 2 – Conjuntos fuzzy para as variáveis de entrada no dataset UCI - Abalone

Fonte – Corrêa e Roisenberg (2020)

Os resultados e as conclusões deste trabalho são preliminares, e mais testes são necessários para validação do novo sistema.

Referências

CASILLAS, J. et al. Interpretability improvements to find the balance interpretability-accuracy in fuzzy modeling: an overview. In: **Interpretability issues in fuzzy modeling**. [S.l.]: Springer, 2003. p. 3–22.

CORRÊA, C. E.; ROISENBERG, M. **Método para aumento de interpretabilidade em modelos de Takagi-Sugeno no sistema INFGMN**. Monografia (Graduação em Ciências da Computação) — Universidade Federal de Santa Catarina, UFSC, Florianópolis, SC, 2020.

ENGEL, P. M.; HEINEN, M. R. Incremental learning of multivariate gaussian mixture models. In: SPRINGER. **Brazilian Symposium on Artificial Intelligence**. [S.l.], 2010. p. 82–91.

GAN, M.-T.; HANMANDLU, M.; TAN, A. H. From a gaussian mixture model to additive fuzzy systems. **IEEE Transactions on Fuzzy Systems**, IEEE, v. 13, n. 3, p. 303–316, 2005.

HEFNY, H. A. Comments on "distinguishability quantification of fuzzy sets". **Information Sciences**, Elsevier, v. 177, n. 21, p. 4832–4839, 2007.

MATHWORKS. **Fuzzy logic toolbox user's guide**. 2019b. ed. [S.l.], 2019. Disponível em: <https://www.mathworks.com/help/pdf_doc/fuzzy/index.html>.

MAZZUTTI, T.; ROISENBERG, M.; FILHO, P. J. de F. Infgmn–incremental neuro-fuzzy gaussian mixture network. **Expert Systems with Applications**, Elsevier, v. 89, p. 160–178, 2017.

MAZZUTTI, T.; ROISENBERG, M.; FILHO, P. J. de F. **MODELO INCREMENTAL NEURO-FUZZY GAUSSIAN MIXTURE NETWORK (INFGMN)**. 111 p. Tese (Doutorado em Ciências da Computação) — Universidade Federal de Santa Catarina, UFSC, Florianópolis, SC, 2018.

MENCAR, C.; CASTELLANO, G.; FANELLI, A. M. Distinguishability quantification of fuzzy sets. **Information Sciences**, Elsevier, v. 177, n. 1, p. 130–149, 2007.

TSEKOURAS, G. E. Fuzzy rule base simplification using multidimensional scaling and constrained optimization. **Fuzzy sets and systems**, Elsevier, v. 297, p. 46–72, 2016.

TUNG, W. L.; QUEK, C. efsm—a novel online neural-fuzzy semantic memory model. **IEEE Transactions on Neural Networks**, IEEE, v. 21, n. 1, p. 136–157, 2009.

ZHOU, S.-M.; GAN, J. Q. Low-level interpretability and high-level interpretability: a unified view of data-driven interpretable fuzzy system modelling. **Fuzzy sets and systems**, Elsevier North-Holland, Inc., v. 159, n. 23, p. 3091–3131, 2008.