Bruno Légora Souza da Silva

Fast Deep Stacked Network: Um algoritmo baseado em *Extreme Learning Machine* para treinamento rápido de uma arquitetura empilhada com pesos compartilhados

Brasil

2022

Bruno Légora Souza da Silva

Fast Deep Stacked Network: Um algoritmo baseado em *Extreme Learning Machine* para treinamento rápido de uma arquitetura empilhada com pesos compartilhados

Tese de doutorado apresentado ao Programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Doutor em Engenharia Elétrica.

Universidade Federal do Espírito Santo

Departamento de Engenharia Elétrica

Programa de Pós-Graduação

Orientador: Dr. Patrick Marques Ciarelli

Brasil

2022

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

Silva, Bruno Légora Souza da, 1991-
S586f Fast Deep Stacked Network: Um algoritmo baseado em Extreme Learning Machine para treinamento rápido de uma arquitetura empilhada com pesos compartilhados / Bruno Légora Souza da Silva. - 2022.
183 f. : il.

Orientador: Patrick Marques Ciarelli.
Tese (Doutorado em Engenharia Elétrica) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Análise de Regressão. 2. Classificação. 3. Aprendizado do computador. 4. Inteligência artificial. I. Ciarelli, Patrick Marques. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 621.3

Bruno Légora Souza da Silva

Fast Deep Stacked Network: Um algoritmo baseado em *Extreme Learning Machine* para treinamento rápido de uma arquitetura empilhada com pesos compartilhados

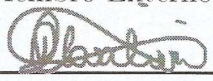
Tese de doutorado apresentado ao Programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Doutor em Engenharia Elétrica.

Trabalho aprovado. Brasil. 18 de março de 2022:

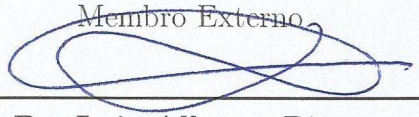
Dr. Patrick Marques Ciarelli
Universidade Federal do Espírito Santo
Orientador

gov.br Documento assinado digitalmente
Carmelo Jose Albanez Bastos Filho
Data: 22/03/2022 17:05:11-0300
Verifique em https://verificador.iti.br

Dr. Carmelo Jose Albanez Bastos Filho
Universidade de Pernambuco
Membro Externo



Dr. Daniel Cruz Cavalieri
Instituto Federal do Espírito Santo
Membro Externo



Dr. Luiz Alberto Pinto
Instituto Federal do Espírito Santo
Membro Externo

Dr. Thomas Walter Rauber
Universidade Federal do Espírito Santo
Membro Externo

Brasil
2022

Agradecimentos

Agradeço a Deus e a todos os meus familiares, especialmente meus pais, que sempre me apoiaram e incentivaram, mesmo nos momentos mais difíceis.

Ao professor Patrick Marques Ciarelli, meu orientador desde 2014. Ao Evandro e demais professores do CISNE, pelas diversas contribuições dadas ao longo destes anos como aluno do PPGEE.

Aos professores da Coordenadoria de Engenharia Elétrica do IFES - Campus Vitória, do Departamento de Engenharia Elétrica e do Departamento de Informática da UFES, que contribuíram para minha formação acadêmica. Aos demais professores que também contribuíram para a minha formação.

A todos os colegas que passaram pelo CISNE durante estes anos da minha vida acadêmica. Em especial à Thaís, pelo companheirismo diário e apoio dado nestes últimos anos.

Ao Programa de Pós-Graduação em Engenharia Elétrica da UFES e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pela bolsa concedida.

Aos professores doutores Carmelo Jose Albanez Bastos Filho, Daniel Cruz Cavalieri, Luiz Alberto Pinto e Thomas Walter Rauber, por aceitarem o convite para compor a banca avaliadora deste trabalho e pelas contribuições dadas nos momentos oportunos.

Resumo

O uso de Redes Neurais Artificiais (RNA) para resolução de problemas de classificação e regressão ganhou bastante popularidade, principalmente após a introdução do algoritmo *backpropagation* para treiná-las utilizando conjuntos de dados. Nos últimos anos, o grande volume de dados gerados e a capacidade de processamento de computadores e placas gráficas tornou possível treinar grandes arquiteturas (profundas) capazes de extrair e predizer informações sobre problemas complexos, usualmente usando grandes quantidades de tempo. Em contrapartida, algoritmos rápidos para o treinamento de redes simples, como a composta por apenas uma camada oculta, chamadas de *Single Layer Feedforward Network* (SLFN), mas capazes de aproximar qualquer função contínua, foram propostos. Um deles é o chamado *Extreme Learning Machine* (ELM), que possui solução rápida e fechada, sendo aplicado em diversas áreas do conhecimento e obtendo desempenhos superiores a outros métodos, como as próprias RNA treinadas com *backpropagation* e *Support Vector Machines* (SVM). Variantes do ELM foram propostas para resolver problemas de subajuste e sobreajuste, *outliers*, entre outros, mas ainda sofrem na presença de grandes volumes de dados e/ou quando é necessária uma arquitetura com mais neurônios para extrair mais informações. Nesse sentido, foi proposta uma versão empilhada, chamada *Stacked* ELM, que põe várias SLFN treinadas por ELM em cascata, aproveitando informações de um módulo em sua posterior, mas que possui limitação quanto ao consumo de memória, além de não ser adequada para lidar com problemas que envolvem uma única saída, como típicas tarefas de regressão. Outro método empilhado é chamado de *Deep Stacked Network* (DSN), que possui problemas quanto ao tempo de treinamento e uso de memória, mas sem apresentar a limitação de aplicação do *Stacked* ELM. Assim, este trabalho propõe combinar a arquitetura DSN com o algoritmo ELM e o *Kernel* ELM a fim de obter arquiteturas que empilham módulos pequenos, com treinamento rápido e utilizando pouca memória, capaz de atingir desempenhos equivalentes a modelos mais complexos. Também é proposta uma forma desta arquitetura lidar com dados que vão chegando aos poucos, chamado aprendizado incremental (ou *online*, no contexto de ELM). Vários experimentos foram conduzidos para avaliar os métodos propostos, tanto para problemas de classificação quanto regressão. No caso do método *online*, foram considerados apenas os problemas de regressão. Os resultados mostram que as técnicas são capazes de treinar arquiteturas empilhadas com desempenhos estatisticamente equivalentes às SLFN com muitos neurônios ou a métodos *online* propostos na literatura, quando as métricas acurácia e erro médio são avaliados. Quanto ao tempo de treinamento, os métodos se mostraram mais rápidos em diversos casos. Já quanto ao uso de memória, os métodos propostos se mostraram estatisticamente superiores, o que facilita o seu uso em ambientes com hardware restrito.

Palavras-chave: *Deep Stacked Network*. *Extreme Learning Machine*. Classificação. Regressão. Modelos Empilhados. Aprendizado incremental.

Abstract

Artificial Neural Networks have been applied to solve classification and regression problems, increasing their popularity, mainly since the proposal of the backpropagation algorithm for its training stage using datasets. In the past years, the volume of generated data and the increased processing power of computers and Graphical Processing Units (GPU) enabled the training of large (or deep) architectures, capable of extracting and predicting information from complex problems, which are usually computationally expensive. In contrast, fast algorithms to train small (shallow) architectures, such as the single hidden layer feedforward network (SLFN), but capable of approximate any continuous function, were proposed. One of them is called Extreme Learning Machine (ELM), which has a fast and closed solution and was applied in wide range of applications, obtaining better performances than other methods, such as backpropagation-trained neural networks and Support Vector Machines (SVMs). Variants of ELM were proposed to solve problems such as underfitting, overfitting and outliers, but they still suffer when used with large datasets and/or when more neurons are required to extract more information. Thus, Stacked ELM (S-ELM) was proposed, which stacks ELM-trained modules, using information from a module in the next one, improving the results using large datasets, but it has limitation regarding the memory consumption, furthermore, it is not adequate for handling problems that involve a single output, such as some regression tasks. Another stacked method is called Deep Stacked Network (DSN), which has problems with training time and memory usage, but without the application limitation of Stacked ELM. Therefore, this work proposes to combine the DSN architecture with the ELM and Kernel ELM algorithms in order to obtain an model composed of small modules, with fast training and with a reduced memory usage, but capable of obtain similar performances compared more complex models. We also propose a variation of this model which deals with data that arrives gradually, called incremental learning (or online in the ELM context). Extensive experiments were conducted to evaluate the proposed methods in regression and classification tasks. Regarding the online approach, only regression tasks were considered. The results show that the methods are capable of training stacked architectures with statistically equivalent performances to SLFN with a large amount of neurons or (or other online methods), when comparing an error or accuracy metric. Considering the training time, the proposed methods spent less time in many cases. When memory usage is considered, some of the proposed methods were considered statistically better, which favors its use in restricted environments.

Keywords: Deep Stacked Network. Extreme Learning Machine. Classification. Regression. Stacked Models. Online Sequential Learning.

Lista de ilustrações

Figura 1 – Um exemplo de arquitetura de rede neural do tipo SLFN. 40
Figura 2 – Um exemplo da arquitetura treinada utilizando ML-ELM. 52
Figura 3 – Um exemplo da arquitetura S-ELM 53
Figura 4 – Um exemplo da arquitetura DSN 55
Figura 5 – Representação gráfica do teste de Nemenyi para quadro métodos diferentes 83
Figura 6 – Acurácia de treino e teste na base de dados Fukunaga I-I de acordo com o número de amostras por classe 89
Figura 7 – Acurácia de treino e teste na base de dados Fukunaga I-4I de acordo com o número de amostras por classe 89
Figura 8 – Acurácia de treino e teste na base de dados Fukunaga I-Λ de acordo com o número de amostras por classe 90
Figura 9 – Acurácia de treino e teste na base de dados Fukunaga I-I-NULL de acordo com o número de amostras por classe 90
Figura 10 – A variação da acurácia dos métodos empilhados ao incluir novos módulos, considerando a base Segment 91
Figura 11 – O efeito da escolha de S e L na Acurácia de modelos DSN e FDSN na base de dados SatImage 92
Figura 12 – Acurácias obtidas nos problemas de classificação. Em (a), os métodos com os oito melhores *ranks*. Os demais estão em (b). 93
Figura 13 – Teste de Nemenyi utilizando a métrica Acurácia para os experimentos com problemas de classificação. 95
Figura 14 – Teste de Nemenyi utilizando o tempo de treinamento para os experimentos com problemas de classificação. 97
Figura 15 – Teste de Nemenyi utilizando o tamanho do modelo em bytes para os experimentos com problemas de classificação. 98
Figura 16 – Gráfico das métricas aRRMSE das técnicas com os oito melhores *ranks* obtidos nos problemas de regressão. 100
Figura 17 – Gráfico das métricas aRRMSE das técnicas com os oito piores *ranks* obtidos nos problemas de regressão. 101
Figura 18 – Teste de Nemenyi utilizando a métrica aRRMSE para os experimentos com problemas de regressão. 103
Figura 19 – Teste de Nemenyi utilizando o tempo de treinamento para os experimentos com problemas de regressão. 105
Figura 20 – Teste de Nemenyi utilizando o tamanho do modelo em bytes para os experimentos com problemas de regressão. 107

Figura 21 – Análise de Componentes Principais nas Bases de Dados *Wind* e *Solar Energy*. 112
Figura 22 – Exemplo de uma Análise por Janela Deslizante. 113
Figura 23 – *Boxplots* das RMSE obtidas nos experimentos com a base de dados *Solar Energy* 116
Figura 24 – Gráfico da RMSE obtidas nas janelas deslizantes do experimento com a base de dados *Solar Energy* 117
Figura 25 – *Boxplots* das SMAPE obtidas nos experimentos com a base de dados *Solar Energy* 118
Figura 26 – *Boxplots* das RMSE obtidas nos experimentos com a base de dados *Wind Energy* 119
Figura 27 – Gráfico da RMSE obtidas nas janelas deslizantes do experimento com a base de dados *Solar Energy* 119
Figura 28 – *Boxplots* das SMAPE obtidas nos experimentos com a base de dados *Wind Energy* 120
Figura 29 – Exemplo de arquitetura do OSFDSN. 125
Figura 30 – Métrica RMSE obtida nos métodos propostos para diferentes tamanhos de lote. 131
Figura 31 – Análise de Componentes Principais nas bases de dados analisadas. . . . 132
Figura 32 – Tempo de treinamento dos métodos propostos para diferentes tamanhos de lote. 133
Figura 33 – Número de bytes para armazenar os métodos propostos para diferentes tamanhos de lote. 134
Figura 34 – RMSE obtidas nos experimentos com problemas de regressão *online* para bases de dados de até 8000 amostras. 136
Figura 35 – Teste de Nemenyi utilizando a métrica RMSE dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras. 138
Figura 36 – Teste de Nemenyi utilizando o tempo de treinamento dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras. 139
Figura 37 – Teste de Nemenyi utilizando o tamanho do modelo em bytes dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras. 141
Figura 38 – RMSE obtidas nos experimentos com problemas de regressão *online* para todas as bases de dados da Tabela 15. 142
Figura 39 – Teste de Nemenyi utilizando a métrica RMSE dos métodos testados em todas as bases de dados da Tabela 15. 144
Figura 40 – Teste de Nemenyi utilizando o tempo de treinamento dos métodos testados em todas as bases de dados da Tabela 15. 145
Figura 41 – Teste de Nemenyi utilizando o tamanho do modelo em bytes dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras. 147

Lista de quadros

Quadro 1 – Características dos métodos apresentados. 67
Quadro 2 – Resumo das complexidades dos métodos analisados, quando o tempo de treinamento é levado em consideração. 76
Quadro 3 – Resumo das complexidades dos métodos analisados, quando a memória necessária para treinamento é levada em consideração 78
Quadro 4 – Resumo das complexidades dos métodos analisados, quando o número de parâmetros é levado em consideração 79
Quadro 5 – Métodos considerados nos experimentos realizados. 85
Quadro 6 – Métodos considerados nos experimentos realizados. 128

Lista de tabelas

Tabela 1 – Conjuntos de dados de problemas de classificação utilizadas nos experimentos 80
Tabela 2 – Conjuntos de dados de problemas de regressão utilizadas nos experimentos 81
Tabela 3 – Características das bases de dados propostas por Fukunaga (1990). . . 88
Tabela 4 – Bases de dados utilizadas no experimento que avalia a arquitetura da FDSN. As colunas indicam o número de amostras de treino e de teste, número de características (d) e número de classes (m). 91
Tabela 5 – *Ranks* médios ($\overline{\text{Rank}}$) e métrica acurácia no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 1. Os métodos estão ordenados pelo *rank* médio, de forma crescente (do melhor para o pior). 94
Tabela 6 – *Ranks* médios ($\overline{\text{Rank}}$) e tempo de treinamento no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 1. Os métodos estão ordenados pelo *rank* médio, de forma crescente. . . . 96
Tabela 7 – *Ranks* médios ($\overline{\text{Rank}}$) e tamanho do modelo em kilobytes no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 1. Os métodos estão ordenados pelo *rank* médio, de forma crescente. 98
Tabela 8 – *Ranks* médios ($\overline{\text{Rank}}$) e métrica aRRMSE no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 2. Os métodos estão ordenados pelo *rank* médio, de forma crescente (do melhor para o pior). 99
Tabela 9 – *Ranks* médios ($\overline{\text{Rank}}$) e tempo de treinamento no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 2. Os métodos estão ordenados pelo *rank* médio, de forma crescente. . . . 104
Tabela 10 – *Ranks* médios ($\overline{\text{Rank}}$) e tamanho do modelo em kilobytes no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 2. Os métodos estão ordenados pelo *rank* médio, de forma crescente. 106
Tabela 11 – Detalhes da base de dados de Energia Solar. 110
Tabela 12 – Detalhes da base de dados de Energia Eólica. 110
Tabela 13 – Métricas obtidas nos testes realizados com o *framework* de Severiano et al. (2021) usando a base de dados *Solar Energy* 116
Tabela 14 – Métricas obtidas nos testes realizados com o *framework* de Severiano et al. (2021) usando a base de dados *Wind Energy* 118

Tabela 15 – Bases de dados de problemas de regressão utilizados nos experimentos *online*. 126
Tabela 16 – *Ranks* médios ($\overline{\text{Rank}}$) e métrica RMSE no formato "média (desvio padrão)" dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras. Os métodos estão ordenados pelo *rank* médio, de forma crescente. 137
Tabela 17 – *Ranks* médios ($\overline{\text{Rank}}$) e tempo de treinamento no formato "média (desvio padrão)" dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras. Os métodos estão ordenados pelo *rank* médio, de forma crescente. 139
Tabela 18 – *Ranks* médios ($\overline{\text{Rank}}$) e tamanho do modelo em kilobytes no formato "média (desvio padrão)" dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras. Os métodos estão ordenados pelo *rank* médio, de forma crescente. 140
Tabela 19 – *Ranks* médios ($\overline{\text{Rank}}$) e métrica RMSE no formato "média (desvio padrão)" dos métodos testados nas bases de dados da Tabela 15. Os métodos estão ordenados pelo *rank* médio, de forma crescente. 143
Tabela 20 – *Ranks* médios ($\overline{\text{Rank}}$) e tempo de treinamento no formato "média (desvio padrão)" dos métodos testados em todas as bases de dados da Tabela 15. Os métodos estão ordenados pelo *rank* médio, de forma crescente. 145
Tabela 21 – *Ranks* médios ($\overline{\text{Rank}}$) e tamanho do modelo em kilobytes no formato "média (desvio padrão)" dos métodos testados em todas as bases de dados da Tabela 15. Os métodos estão ordenados pelo *rank* médio, de forma crescente. 146
Tabela 22 – Acurácias (%) obtidas sobre o conjunto de teste para os experimentos da Seção 3.4.4.1 163
Tabela 23 – Tempos de treinamento dos experimentos da Seção 3.4.4.1. 163
Tabela 24 – Quantidade de kilobytes gastos em cada modelo dos experimentos da Seção 3.4.4.1. 164
Tabela 25 – Hiperparâmetros de regularização (C) obtidos após o 5-fold para os métodos R-ELM, DSN e FDSN, nos experimentos da Seção 3.4.4.1. . . 165
Tabela 26 – Hiperparâmetros obtidos após o 5-fold para os métodos GR-ELM e FDSN-GR, nos experimentos da Seção 3.4.4.1. 165
Tabela 27 – Hiperparâmetros C e γ obtidos após o 5-fold para o método FKDSN, nos experimentos da Seção 3.4.4.1. 166
Tabela 28 – Funções de ativação obtidas após o 5-fold nos experimentos da Seção 3.4.4.1. As funções sigmoide, *hard-limit*, gaussiana e multiquádrica são representadas pelas letras 's', 'h', 'r' e 'm', respectivamente. 166

Tabela 29 – Valores de aRRMSE obtidas sobre o conjunto de teste para os experimentos da Seção 3.4.4.2 167
Tabela 30 – Tempos de treinamento dos experimentos da Seção 3.4.4.2. 168
Tabela 31 – Quantidade de kilobytes gastos em cada modelo dos experimentos da Seção 3.4.4.2. 170
Tabela 32 – Hiperparâmetros de regularização (C) obtidos após o 5-fold para os métodos R-ELM, DSN e FDSN, nos experimentos da Seção 3.4.4.2. . . 171
Tabela 33 – Hiperparâmetros C e γ obtidos após o 5-fold para o método FKDSN, nos experimentos da Seção 3.4.4.1. 172
Tabela 34 – Hiperparâmetros obtidos após o 5-fold para os métodos GOR-ELM e FDSN-GOR, nos experimentos da Seção 3.4.4.2. 173
Tabela 35 – Funções de ativação obtidas após o 5-fold nos experimentos da Seção 3.4.4.2. As funções sigmoide, *hard-limit*, gaussiana e multiquádrica são representadas pelas letras 's', 'h', 'r' e 'm', respectivamente. 174
Tabela 36 – Hiperparâmetros obtidos no estudo de caso feito no Capítulo 4. 175
Tabela 37 – Valores de RMSE obtidas nos experimentos da Seção **??**. 177
Tabela 38 – Tempos de treinamento dos testes da Seção 5.2.3.1. 177
Tabela 39 – Quantidade de kilobytes gastos em cada modelo dos experimentos da Seção 5.2.3.1. 178
Tabela 40 – Número médio de módulos dos métodos empilhados dos testes da Seção 5.2.3.1 e número de vizinhos mais próximos do método kNN. . . 178
Tabela 41 – Valores de RMSE obtidas nos experimentos da Seção 5.2.3.2. 179
Tabela 42 – Tempos de treinamento, em segundos, dos experimentos da Seção 5.2.3.2. 180
Tabela 43 – Quantidade de kilobytes usados em cada modelo dos experimentos da Seção 5.2.3.2. 180
Tabela 44 – Número médio de módulos dos métodos empilhados dos testes da Seção 5.2.3.2 e número de vizinhos mais próximos do método kNN. . . 181

Lista de abreviaturas e siglas

ADMM	*Alternating Direction Method of Multipliers*
ALM	*Augmented Lagrange Multiplier Method*
aRRMSE	*Average Relative Root Mean Square Error*
DC	Distância Crítica
DDM	*Drift Detection Method*
DNN	*Deep Neural Network*
DOER	*Dynamic Online Ensemble Regression*
DSN	*Deep Stacked Network*
ELM-AE	*Autoencoder Extreme Learning Machine*
ELM	*Extreme Learning Machine*
ePNN	*Evolving Probabilistic Neural Network*
FDSN	*Fast Deep Stacked Network*
FKDSN	*Fast Kernel Deep Stacked Network*
GOR-ELM	*Generalized Outlier Robust Extreme Learning Machine*
GPU	*Graphics Processing Unit*
GR-ELM	*Generalized Regularized Extreme Learning Machine*
ILSVRC	*ImageNet Large Scale Visual Recognition Competition*
K-ELM	*Kernel Extreme Learning Machine*
ML-ELM	*Multi-layer Extreme Learning Machine*
OEOA	*Online Ensemble using Ordered Aggregation*
OR-ELM	*Outlier Robust Extreme Learning Machine*
OS-ELM	*Online Sequential Extreme Learning Machine*
OSFDSN	*Online Sequential Fast Deep Stacked Network*

OSR-ELM	*Online Sequential Regularized Extreme Learning Machine*
OWE	*Online Weighted Ensemble*
PCA	*Principal Component Analysis*
R-ELM	*Regularized Extreme Learning Machine*
RK-ELM	*Reduced Kernel Extreme Learning Machine*
RMSE	*Root Mean Square Error*
RNA	Rede Neural Artificial
RRMSE	*Relative Root Mean Square Error*
S-ELM	*Stacked Extreme Learning Machine*
SLFN	*Single Layer Feedforward Neural Network*
SVM	*Support Vector Machine*
UWR-ELM	*Unweighted Regularized Extreme Learning Machine*
WR-ELM	*Weighted Regularized Extreme Learning Machine*

Lista de símbolos

$\mathcal{O}(\cdot)$	Notação *Big-O*
$\boldsymbol{X}$	Matriz de dados de entrada de uma base de dados
$\boldsymbol{T}$	Matriz de alvos (*targets*) correspondentes à entrada $\boldsymbol{X}$ da base de dados
$\boldsymbol{t}$	Vetor de alvos correspondentes à entrada $\boldsymbol{X}$ da base de dados
$\hat{\boldsymbol{T}}$	Matriz de alvos estimados correspondentes à uma entrada da base de dados
$\hat{\boldsymbol{t}}$	Vetor de alvos estimados correspondentes à uma entrada da base de dados
$\boldsymbol{H}$	Matriz de saída da camada oculta em relação à entrada $\boldsymbol{X}$
$\boldsymbol{B}$	Matriz de pesos entre a camada oculta e a camada de saída
$\boldsymbol{\beta}$	Vetor de pesos entre a camada oculta e a saída única de uma SLFN
$\tilde{\boldsymbol{V}}$	Matriz de projeção da camada oculta do método S-ELM
$\boldsymbol{W}$	Matriz de pesos entre a camada de entrada e oculta de uma SLFN
$\boldsymbol{\nu}$	Vetor de vieses da camada oculta da SLFN
$h(\cdot)$	Função de ativação da camada oculta
C	Parâmetro de regularização do algoritmo R-ELM e GR-ELM
α	Parâmetro que controla o *trade-off* entre rede esparsa e densa dos algoritmos R-ELM, GR-ELM e GOR-ELM
λ	Parâmetro de regularização dos algoritmos OR-ELM e GOR-ELM
$\boldsymbol{D}$	Matriz diagonal contendo os pesos das amostras de uma base de dados, no treinamento do WR-ELM
$L(\cdot)$	Função Lagrangiana de algum problema de otimização
ρ	Parâmetro de regularização do ADMM
$\boldsymbol{Y}$	Matriz de multiplicadores de Lagrange
$\langle \cdot, \cdot \rangle$	Produto interno entre duas matrizes quaisquer

$\boldsymbol{U}$	Variável dual escalonada do ADMM
$\boldsymbol{Z}$	Variável incluída na reescrita de problemas de otimização
$\boldsymbol{z}_{i,\cdot}$	Linha da matriz $\boldsymbol{Z}$
$S_\kappa(\cdot)$	Operador *block-soft thresholding* com argumento κ
$\boldsymbol{R}$	Resíduo primal do ADMM
$\boldsymbol{S}$	Resíduo dual do ADMM
$\|\cdot\|_F$	Norma matricial Frobenius
$\|\cdot\|_{2,1}$	Norma matricial $\ell_{2,1}$
$\|\cdot\|_2$	Norma vetorial ℓ_2
$\|\cdot\|_1$	Norma vetorial ℓ_1
$\|\cdot\|_0$	Norma vetorial ℓ_0
τ	Parâmetro de regularização do OR-ELM e GOR-ELM
ϵ	Variável incluída na reescrita de problemas de otimização
$\widetilde{\boldsymbol{A}}$	Matriz construída na otimização do GOR-ELM
$\widetilde{\boldsymbol{B}}$	Matriz construída na otimização do GOR-ELM
$\widetilde{\boldsymbol{C}}$	Matriz construída na otimização do GOR-ELM
$\widetilde{\boldsymbol{D}}$	Matriz construída na otimização do GOR-ELM
$\mathcal{S}_\kappa(\cdot)$	Operador que aplica o *block-soft thresholding* em todas as linhas da matriz de seu argumento
ϵ^{abs}	Tolerância absoluta do critério de parada do ADMM
ϵ^{rel}	Tolerância relativa do critério de parada do ADMM
$\boldsymbol{I}$	Matriz identidade
η	Parâmetro de regularização do GOR-ELM
S	Número de módulos de um modelo empilhado ou número máximo de membros de um *ensemble*
$\boldsymbol{K}$	Matriz acumuladora de informação do método OS-ELM, sendo atualizada em cada iteração

$\boldsymbol{P}$	Inversa da matriz $\boldsymbol{K}$, isto é, $\boldsymbol{K}^{-1}$
$\boldsymbol{W}^{\mathrm{RO}}$	Matriz de pesos retornada pelo método de inicialização ortogonal
$\boldsymbol{W}^{\mathrm{pca}}$	Matriz de pesos obtida após o PCA nos dados de entrada $\boldsymbol{X}$ no método de inicialização ortogonal
$\boldsymbol{W}^{\mathrm{orth}}$	Base ortogonal gerada a partir de uma matriz de pesos
$\boldsymbol{G}(\cdot)$	Operador que aplica uma função de ativação $h(\cdot)$ em todos os elementos de uma matriz
$\boldsymbol{\Pi}$	Matriz $\boldsymbol{H}$ antes da aplicação da função de ativação em todos os seus elementos
$\boldsymbol{N}$	Matriz onde todas as linhas são iguais ao vetor $\boldsymbol{\nu}$
L	Número de neurônios na camada oculta de uma SLFN
m	Dimensão das amostras de saída (alvos) de uma base de dados
n	Dimensão das amostras de entrada (atributos) de uma base de dados
N	Número de amostras de uma base de dados
Q	Estatística do teste de Friedman
D	Número de bases de dados do teste de Friedman
ψ	Número de algoritmos usados no teste de Friedman
q_γ	Número de algoritmos usados no teste de Friedman para um nivel de significância γ
$\mathcal{P}(\cdot)$	Probabilidade de ocorrência de um evento
$\chi^2_{(\psi-1)}$	Distribuição χ^2 com $\psi - 1$ graus de liberdade
$\boldsymbol{\Omega}_{ELM}$	Matriz de *kernel* dos métodos K-ELM, RK-ELM e FKDSN
$K(u, v)$	Função de *Kernel* escolhida nos métodos K-ELM, RK-ELM e FKDSN
Δ	Subconjunto de amostras de treino utilizado como "vetores suporte" nas técnicas RK-ELM e FKDSN
δ_j	Uma amostra de treino que pertence a Δ
$\mathcal{D}^t$	t-ésimo lote de amostras de uma base de dados *online*

MSE_j^t	Erro quadrático médio do j-ésimo módulo com relação ao t-ésimo lote de amostras
e_j^t	Erro quadrático do j-ésimo membro com respeito a t-ésima amostra

Sumário

1 INTRODUÇÃO 29
1.1 Hipótese 33
1.2 Objetivos 34
1.3 Contribuições 34
1.4 Organização do Trabalho 35
1.5 Notações utilizadas 35
1.6 Publicações 37

2 REFERENCIAL TEÓRICO 39
2.1 Técnicas de Aprendizado de Máquina Offline 39
2.1.1 *Extreme Learning Machine* 39
2.1.2 *Regularized Extreme Learning Machine* 41
2.1.3 *Generalized Regularized Extreme Learning Machine* 42
2.1.4 *Outlier Robust Extreme Learning Machine* 45
2.1.5 *Generalized Outlier Robust Extreme Learning Machine* 46
2.1.6 Kernel Extreme Learning Machine 48
2.1.7 *Multi-layer Extreme Learning Machine* 50
2.1.8 *Stacked Extreme Learning Machine* 52
2.1.9 *Deep Stacked Network* 54
2.2 Técnicas de Aprendizado de Máquina Online 55
2.2.1 *Online Sequential Extreme Learning Machine* 57
2.2.2 *Online Sequential Regularized Extreme Learning Machine* 59
2.2.3 Learn^{++}.NSE 60
2.2.4 *Online Weighted Ensemble* 62
2.2.5 *Dynamic And Online Ensemble Regression* 63
2.2.6 *Online Ensemble Using Ordered Aggregation* 64
2.3 Métodos de Inicialização de Pesos de ELM e suas variantes 65
2.3.1 Inicialização ortogonal 65
2.3.2 *Autoencoder* 66
2.4 Considerações Finais 66

3 PROPOSTAS CONSIDERANDO MÉTODOS *OFFLINE* 69
3.1 Problemas Comuns de Métodos Offline 69
3.2 Métodos Offline Propostos 70
3.2.1 *Fast Deep Stacked Network* 70
3.2.2 *Fast Kernel Deep Stacked Network* 73

3.2.3 Resumo das Técnicas Propostas . . . 75
3.3 Complexidade Computacional dos algoritmos FDSN e FKDSN . . . 75
3.3.1 Análisa Assintótica do Tempo de Treinamento . . . 76
3.3.2 Análisa Assintótica da Memória Usada no Treino . . . 76
3.3.3 Número de Parâmetros dos Modelos . . . 77
3.4 Experimentos Considerando FDSN e FKDSN . . . 79
3.4.1 Metodologia dos Experimentos . . . 79
3.4.1.1 Conjuntos de dados utilizados . . . 80
3.4.1.2 Métricas e Testes Estatísticos . . . 80
3.4.1.3 Detalhes da Implementação e Ambiente de Execução . . . 84
3.4.1.4 Especificações dos Hiperparâmetros . . . 86
3.4.2 Análise Considerando o Tamanho da Base da Dados . . . 87
3.4.3 Análise Considerando Diferentes Arquiteturas . . . 89
3.4.4 Análise por Teste de Hipótese . . . 92
3.4.4.1 Resultados em Bases de Dados de Classificação . . . 92
3.4.4.2 Resultados em Bases de Dados de Regressão . . . 99
3.5 Considerações Finais . . . 106

4 ESTUDO DE CASO DE MÉTODOS OFFLINE . . . 109
4.1 Metodologia . . . 110
4.1.1 Bases de dados e Métricas Utilizadas . . . 110
4.1.2 Métodos Utilizados e Detalhes de Implementação . . . 111
4.1.3 Especificação dos Hiperparâmetros . . . 113
4.2 Experimentos e Resultados . . . 114
4.2.1 Base de Dados *Solar Energy* . . . 114
4.2.2 Base de Dados *Wind Energy* . . . 115
4.3 Considerações Finais . . . 119

5 PROPOSTAS CONSIDERANDO MÉTODOS *ONLINE* . . . 121
5.1 *Online Sequential Fast Deep Stacked Network* . . . 121
5.2 Experimentos Considerando Métodos *Online* . . . 125
5.2.1 Metodologia dos Experimentos . . . 125
5.2.1.1 Detalhes da Implementação e Ambiente de Execução . . . 127
5.2.1.2 Especificação dos Hiperparâmetros . . . 128
5.2.2 Análise considerando tamanhos de lotes distintos . . . 129
5.2.3 Análise por Teste de Hipótese . . . 135
5.2.3.1 Considerando Bases Menores e os métodos DOER e OEOA . . . 135
5.2.3.2 Considerando Todas as Bases Levantadas . . . 141
5.3 Considerações Finais . . . 147

6 CONCLUSÃO E TRABALHOS FUTUROS 149

REFERÊNCIAS 153

APÊNDICES 161

APÊNDICE A – TABELAS DE RESULTADOS - MÉTODOS OFFLINE - PROBLEMAS DE CLASSIFICAÇÃO . . 163

APÊNDICE B – TABELAS DE RESULTADOS - MÉTODOS OFFLINE - PROBLEMAS DE REGRESSÃO 167

APÊNDICE C – TABELAS DE RESULTADOS - MÉTODOS OFFLINE - ESTUDO DE CASO 175

APÊNDICE D – TABELAS DE RESULTADOS - MÉTODOS ONLINE - BASES DE DADOS *ONLINE* ATÉ 8000 AMOSTRAS 177

APÊNDICE E – TABELAS DE RESULTADOS - MÉTODOS ONLINE - BASES DE DADOS *ONLINE* 179

1 Introdução

No século XX, estudos reconheceram que o cérebro humano trabalha de uma forma diferente do que os computadores digitais convencionais. O cérebro é altamente complexo, não linear e processa as informações de forma paralela, possuindo a capacidade de organizar seus componentes, chamados de neurônios, em estruturas capazes de realizar operações complexas rapidamente (HAYKIN, 2009). Isso incentivou estudos e trabalhos em Redes Neurais Artificiais (RNA), comumente chamadas apenas de "Redes Neurais".

Um dos principais diferenciais do cérebro humano é a sua capacidade de generalização, sendo capaz de reconhecer objetos diferentes, mas que possuem alguma similaridade com algum conhecimento já adquirido (HAYKIN, 2009). O cérebro humano também possui a capacidade de se adaptar a novos ambientes e/ou novos conhecimentos, à medida que eles vão sendo disponibilizados. Humanos, por exemplo, nascem com pouca informação e têm seu conhecimento adquirido ao longo de seu crescimento. Ou seja, a rede neural biológica aprende com os dados que ela recebe ao longo do tempo.

As RNA também aprendem a partir de dados apresentados, e possuem capacidade de aprendizado de modelos complexos, como os não-lineares, de produzir uma saída confiável dada uma entrada e a de lidar com vários tipos de dados, por ser uma espécie de "processador universal de informações" (HAYKIN, 2009).

Assim, as RNA funcionam de uma forma simplificada e similar ao funcionamento de uma rede neural biológica: dados os seus neurônios artificiais organizados em camadas e um conjunto de dados, seja ele de qualquer origem, um algoritmo de treinamento é utilizado para transformar tais dados em informações armazenadas na estrutura neural.

Uma das possíveis arquiteturas de redes neurais supervisionadas é a chamada *single hidden layer feedforward neural network* (SLFN), que possui apenas uma camada (oculta) de neurônios entre a entrada e saída da rede. O termo *feedforward* indica que a informação é propagada apenas em uma direção (i.e., não há realimentação). Tal arquitetura tem seus neurônios conectados aos das camadas vizinhas através de dois conjuntos de pesos (da entrada para a camada oculta e da camada oculta para a saída), que são parâmetros que devem ser adaptados utilizando um algoritmo de treinamento.

Um dos algoritmos mais utilizados para o treinamento de alguns tipos de RNA é chamado de *backpropagation*, que foi proposto por Werbos (1974) e possibilitou a aplicação de RNA em diversas áreas do conhecimento.

O *backpropagation* é uma aplicação de descida de gradiente para otimização de parâmetros, que têm convergência lenta e problemas com mínimos locais (HUANG et al.,

2015). Embora seja possível a utilização de outros métodos de otimização em conjunto com esse algoritmo, de forma a reduzir o tempo de convergência e/ou melhorar a sua capacidade de generalização, ainda não é possível garantir que o resultado encontrado será satisfatório (HUANG et al., 2015).

O uso de uma arquitetura SLFN, embora aparentemente simples, é respaldado pelo Teorema da Aproximação Universal (CYBENKO, 1989), que afirma que uma rede neural com apenas uma camada oculta, com um número adequado de neurônios nesta camada, é suficiente para aproximar qualquer função contínua, a partir de um conjunto de treinamento com N pares entrada-saída. Isto é, para um valor de erro $\xi > 0$ desejado, existirá uma SLFN com L neurônios tal que o módulo do erro entre a aproximação e o valor real para qualquer amostra é menor que ξ. O teorema, porém, não diz nada quanto à generalização dos dados ou tempo de treinamento desta rede.

Huang, Zhu e Siew (2004) mostraram que uma SLFN com L neurônios na camada oculta, mesmo tendo seus pesos de entrada (entre a entrada e camada oculta) amostrados aleatoriamente, é capaz de aprender exatamente L observações distintas. Assim, Huang, Zhu e Siew (2006) propuseram um algoritmo rápido de treinamento para esta arquitetura, denominado *Extreme Learning Machine* (ELM). Alguns autores utilizam a notação "rede ELM" (RAMZAN et al., 2018; ZYARAH; KUDITHIPUDI, 2017; SANTOS; SILVA FILHO; SANTOS, 2019) ou ainda "modelo ELM" (ZHOU; CHEN; ZHU, 2018; SOARES; ARAÚJO, 2016) para se referirem à uma SLFN treinada com ELM, embora essa sigla tenha sido definida como um algoritmo de treinamento para a arquitetura citada.

O ELM, devido a sua rapidez e simplicidade de implementação, se tornou bastante popular e foi utilizado em diversas aplicações com sucesso, como visão computacional, processamento de imagens, análise de séries temporais, entre outras, muitas vezes atingindo resultados superiores até a métodos como o *Support Vector Machine* (SVM) e SLFNs similares treinadas utilizando o algoritmo *backpropagation* (HUANG et al., 2015).

A ideia original do ELM é gerar os pesos de entrada aleatoriamente, encontrar a matriz de saída da camada oculta utilizando tais pesos e um conjunto de dados e, finalmente, calcular os pesos de saída considerando uma máquina linear, que obtém tais valores de forma determinística, utilizando o conceito de pseudoinversa da matriz de saída da camada oculta (também chamada de inversa de Moore–Penrose) (HUANG et al., 2012).

Modificações no algoritmo ELM original foram propostas com o objetivo de melhorar o desempenho da rede resultante. Como exemplo, a técnica *Regularized* ELM (R-ELM) permite controlar um *trade-off* entre redes mais esparsas e densas, porém com a desvantagem de tratar apenas redes com um neurônio de saída. Também foram feitas modificações para o algoritmo ser robusto à *outliers*, resultando na técnica *Outlier Robust* ELM (OR-ELM) (ZHANG; LUO, 2015). Embora definido para apenas saídas unidimensionais, o R-ELM pode ser facilmente adaptado para múltiplas saídas, calculando cada uma separadamente

em problemas de regressão, ou a estratégia *one-against-all* em problemas de classificação. O R-ELM posteriormente foi estendido para considerar múltiplas saídas simultaneamente, utilizando o conceito de normas matriciais e otimização através do algoritmo *Alternating Direction Method of Multipliers* (ADMM) para problemas de classificação, resultando na técnica *Generalized Regularized* ELM (GR-ELM) (INABA et al., 2018) e regressão robusta à *outliers*, resultando na técnica *Generalized Outlier Robust* ELM (GOR-ELM) (SILVA et al., 2020b). Outra modificação no algoritmo ELM se refere ao uso de *kernels*, que podem substituir a natureza aleatória do algoritmo, dando origem ao *Kernel* ELM.

Embora muito rápido e capaz de obter boas generalizações, algoritmos baseados em ELM tendem a sofrer quando são utilizados com conjuntos de dados com muitas amostras, principalmente se o número L for grande (um processo de inversão de matriz é necessário). Assim, algoritmos baseados em ELM podem sofrer com problemas de limitação de memória, no processo de treinamento e armazenamento do modelo, e custo computacional intenso nas operações matriciais (WANG et al., 2016).

Com o advento da era do *big-data*, problemas na aquisição de dados com erro humano ou de instrumentos começaram a se tornar mais frequentes (INABA, 2018), bem como grandes conjuntos de dados, que podem dificultar uma boa exploração de uma SLFN treinada com ELM. Segundo Zhou et al. (2014), há diversas bases de dados com muitas amostras que necessitam de uma camada oculta suficientemente grande para que as informações sejam bem extraídas. Isto é, desempenhos melhores podem ser obtidos apenas aumentando o número de neurônios ocultos na rede e/ou aumentar a quantidade de camadas (ou "estágios de processamento"). Porém, treinar SLFN com muitos neurônios e/ou redes neurais com muitas camadas utilizando largos conjuntos de dados requer computadores com grande quantidade de memória e capacidade de processamento. Todavia, o acesso a tais infraestruturas nem sempre é possível, mesmo facilitado através de serviços disponíveis na internet, a depender de questões financeiras.

Uma nova classe de técnicas de aprendizado de máquina, chamada de *deep learning*, foi proposta com o objetivo de extrair informações complexas e construir representações de grande quantidades de dados. Estas técnicas consistem tipicamente na utilização de um grande número de estágios de processamento da informação, onde usualmente são utilizadas redes neurais com um grande número de camadas ocultas (DENG, 2014).

Como resultado da grande quantidade de camadas, tais redes possuem um elevado número de parâmetros, o que dificulta o seu treinamento se comparadas às redes neurais "rasas", que possuem poucos parâmetros. Há a possibilidade de utilizar camadas convolucionais, que utilizam filtros que implementam a operação matemática de convolução com um reduzido número de parâmetros (YOUNG et al., 2018). Devido à natureza convolucional, esta abordagem é adequada para lidar com certos tipos de dados, como séries temporais, áudios ou vídeos. Já as redes "rasas" usualmente não têm a mesma capacidade de lidar

com características temporais e/ou espaciais.

Tais redes convolucionais vêm chamando a atenção pelos bons resultados atingidos em diversas aplicações, como reconhecimento de fala (DENG, 2016) e localização de textos em imagens (ZHU; ZANIBBI, 2016), processamento de linguagem natural (YOUNG et al., 2018) ou ainda aplicações de visão computacional (SANTOS; SAMATELO; VASSALLO, 2020). Porém, o fato destes sistemas utilizarem um grande número de estágios de processamento e a utilização de uma grande quantidade de dados de treino implica em um intenso custo computacional, sendo necessária a utilização de unidades com alta capacidade de processamento para seu treino (DENG, 2014). Embora o treinamento de tais redes possa ser feito de forma *off-line*, esperar esta etapa pode não ser factível, a depender da aplicação, que já pode apresentar resultados satisfatórios com técnicas mais simples.

Muitas vezes, redes neurais profundas são estabelecidas com arquiteturas estáticas e demasiadamente grandes, o que pode implicar em um compromisso inadequado entre acurácia (desempenho) e custo computacional (tempo de processamento), e que podem não ser capazes de se adaptarem a novos dados. Como exemplo, a arquitetura vencedora da *ImageNet Large Scale Visual Recognition Competition* (ILSVRC) 2013, chamada de ZFNet, possui 8 camadas e taxa de erro de 11,2%, enquanto a arquitetura vencedora da edição de 2014, chamada de GoogLeNet, possui 22 camadas e taxa de erro de 6,7% (RUSSAKOVSKY et al., 2015), ou seja, uma arquitetura quase três vezes maior foi necessária para reduzir o erro em 4,5%.

Em compensação, autores como Wolpert (1992) propõem a combinação de diversos modelos pequenos e treinados separadamente, porém organizados de forma empilhada para que melhores desempenhos sejam atingidos, evitando assim o problema do custo computacional causado pela combinação de grande quantidade de dados e de parâmetros no modelo, que é chamado de problema de escalabilidade (DENG; YU, 2011).

Uma das arquiteturas empilhadas propostas na literatura é chamada de *Deep Stacked Network* (DSN), em que módulos pequenos são treinados independentemente. Embora independentes, a entrada de um módulo empilhado é dada pelos dados de entrada concatenados com a saída do módulo anterior e, portanto, não é possível treinar diversos módulos ao mesmo tempo (DENG, 2014). Essa rede, embora lide com modelos menores, ainda possui problemas computacionais de memória que muitas vezes podem limitar seu uso em ambientes restritos. No contexto de ELM, também foi proposto uma arquitetura chamada *Stacked* ELM (ZHOU et al., 2014) (S-ELM), que empilha módulos pequenos, utilizando *Principal Component Analysis* (PCA) para reduzir dimensão de um módulo e aproveitar este conhecimento para treinar o módulo seguinte, melhorando o desempenho da rede geral.

Os métodos citados sempre consideram que o seu treinamento irá lidar com todo o conjunto de dados disponível para o problema a ser resolvido. Contudo, existem algoritmos

que lidam com conjuntos que vão sendo disponibilizados ao longo do tempo, e que são chamados de incrementais (POLIKAR et al., 2001; MINKU, 2011; CIARELLI; OLIVEIRA; SALLES, 2012), ou ainda de "online" ou "sequenciais", principalmente no contexto de ELM (LIANG et al., 2006; SHAO; ER, 2016)[1]. Uma das vantagens deste tipo de algoritmo é a sua melhor capacidade de lidar com dados que vão mudando suas características ao longo do tempo, fenômeno denominado *concept drift* (MINKU, 2011).

Uma classe ainda mais específica entre os métodos incrementais são aqueles que têm a capacidade de evoluir (adaptar) sua arquitetura de acordo com os novos dados recebidos, se adaptando ao problema de aprendizado de máquina de modo controlado, sem perder o controle do seu crescimento. Exemplos deste tipo de rede, que são chamadas de evolutivas, são a *Evolving Fuzzy Neural Network* (EFuNN) (WATTS, 2009) e a *Evolving Probabilistic Neural Network* (ePNN) (CIARELLI; OLIVEIRA; SALLES, 2012).

Os métodos DSN e S-ELM, que empilham módulos com a finalidade de melhorar seus desempenhos, possuem desvantagens em relação ao tempo de treinamento e à memória necessária para armazená-los. Com a finalidade de mitigar tais problemas, este trabalho propõe o *Fast* DSN (FDSN) e o *Fast Kernel* DSN (FKDSN), nos quais em ambas as abordagens os módulos são construídos usando cálculos recursivos e são utilizados algoritmos de treinamento baseados no ELM e no *Kernel* ELM, respectivamente, acelerando o treinamento e reduzindo a necessidade de memória para armazenar os modelos treinados. Nestas técnicas, alguns pesos das redes neurais são reutilizados, o que reduz o consumo de memória. Porém, o desempenho das mesmas pode ser afetado devido a esta característica.

Uma versão sequencial do FDSN também é proposta. Chamada de *Online Sequential Fast DSN* (OSFDSN), esta técnica combina as vantagens de reuso de pesos do FDSN com a capacidade de atualização de modelos sequenciais, alterando suas características ao longo do tempo para se adequar a possíveis alterações nas distribuições dos dados. O foco desta técnica são problemas de regressão com a presença de *concept drift*, devido à etapa de atualização de pesos de saída levar em consideração métricas comumente utilizadas em problemas de regressão.

1.1 Hipótese

Conforme apresentado, existem problemas no treinamento de redes neurais com grandes bases de dados, seja utilizando algoritmos iterativos como o *backpropagation* ou ELM, ou de métodos empilhados do tipo DSN. Este trabalho parte da hipótese de que a combinação de um algoritmo rápido de treinamento de módulos, compostos de SLFN, em uma rede empilhada e o compartilhamento de pesos entre os módulos, pode tanto acelerar

[1] Neste trabalho, a nomenclatura adotada será "online", utilizada no contexto de ELM, para se referir aos métodos capazes de lidar com dados que são disponibilizados aos poucos ao longo do tempo.

o processo de treinamento quanto reduzir significativamente o uso de memória deste tipo de arquitetura, mantendo a qualidade dos resultados, além de ser possível ajustar tal método para lidar com dados que são disponibilizados de forma sequencial, adaptando sua arquitetura, caso necessário.

1.2 Objetivos

O objetivo geral deste trabalho é propor um método baseado em algoritmos rápidos (por exemplo, ELM) de treinamento de redes neurais para acelerar tal etapa em uma arquitetura do tipo DSN. Para atingir tal resultado, os seguintes objetivos específicos são propostos:

- Treinar uma arquitetura DSN utilizando apenas algoritmos rápidos, como ELM, tal que ela possa tratar problemas de classificação e regressão, uni e multidimensionais.
- Propor uma forma de reduzir o uso de memória e tempo de treinamento da arquitetura em geral.
- Analisar teoricamente o custo computacional do treinamento e do armazenamento dos modelos propostos.
- Avaliar experimentalmente o modelo em bases de dados públicas de classificação e regressão, analisando custo computacional, armazenamento e desempenho, o comparando com outras abordagens da literatura.
- Propor uma forma do algoritmo lidar com dados sequenciais, modificando a arquitetura existente e/ou atualizando os seus pesos.
- Avaliar experimentalmente o modelo que lida com dados sequenciais em bases de dados de regressão, analisando desempenho e custo computacional, o comparando com outras abordagens da literatura.

1.3 Contribuições

Ao longo do desenvolvimento desta Tese de Doutorado foi possível fazer as seguintes contribuições à literatura:

- A proposta de um critério de parada para o algoritmo GOR-ELM baseado em resíduos primal e dual.
- A proposta de uma forma rápida para o treinamento de uma DSN, chamada *Fast* DSN (FDSN).

- A proposta de variantes da FDSN que utilizam abordagens distintas de inicialização de pesos, além de uma forma que utiliza *kernels*.
- Avaliação téorica do custo computacional (tempo e memória) do ELM, *Stacked* ELM e DSN/FDSN.
- A proposta de uma versão sequencial do algoritmo FDSN para lidar com problemas de regressão com *concept drift*, chamado OSFDSN.

1.4 Organização do Trabalho

O Capítulo 1 apresenta conceitos de redes neurais e contextualiza o problema de treinamento das mesmas quando estas possuem muitos neurônios e camadas, além de tratar a hipótese e os objetivos deste trabalho. Um referencial téorico, explicando técnicas nas quais este trabalho se baseou é apresentado no Capítulo 2. No Capítulo 3 são apresentadas as técnicas propostas nestes trabalho que lidam com dados de forma *offline*, ou seja, não consideram que serão disponibilizados mais dados após o treinamento das redes, bem como experimento realizados para avaliar os métodos propostos. No Capítulo 4 é apresentado um estudo de caso que considera bases de dados de problemas reais de previsão de geração de energia solar e eólica, considerando apenas métodos *offline*. Já no Capítulo 5 é apresentado o método *online* proposto, bem como experimentos realizados para avaliar o seu desempenho em problemas de regressão *online* em comparação a outras técnicas propostas na literatura. Finalmente, a conclusão e os trabalhos futuros são apresentados no Capítulo 6.

1.5 Notações utilizadas

Neste trabalho, para facilidade de entendimento e padronização, foram adotadas algumas notações. Vetores são representados em letras minúsculas e em negrito. A menos que o contrário seja dito, vetores são matrizes com apenas uma coluna (vetor coluna). Matrizes são representadas por letras maiúsculas e em negrito, enquanto escalares são representados por letras minúsculas em itálico. O elemento localizado na primeira linha e segunda coluna de uma matriz $\boldsymbol{A}$ é representado por $a_{1,2}$, e a j-ésima linha de $\boldsymbol{A}$ é representada por $\boldsymbol{a}_{j,\bullet}$, enquanto a k-ésima coluna por $\boldsymbol{a}_{\bullet,k}$.

No contexto de métodos de aprendizado de máquina (especialmente o ELM), L representa o número de neurônios ocultos em uma rede neural de camada oculta única, $\boldsymbol{W}$ é a matriz de pesos entre a entrada e camada oculta, $\boldsymbol{\nu}$ é o vetor de vieses, $\boldsymbol{H} \in \mathbb{R}^{N \times L}$ é a matriz que representa a saída da camada oculta desta rede em relação à uma matriz de entrada $\boldsymbol{X} \in \mathbb{R}^{N \times n}$ com N amostras n-dimensionais, $\boldsymbol{T} \in \mathbb{R}^{N \times m}$ (ou $\boldsymbol{t} \in \mathbb{R}^{N}$ para uma dimensão) é a matriz com as N saídas correspondentes as amostras de entrada, e

$\boldsymbol{B} \in \mathbb{R}^{L \times m}$ (ou $\boldsymbol{\beta} \in \mathbb{R}^{N}$ para uma dimensão) é a matriz de pesos que conecta as camadas oculta e de saída desta rede neural. Para métodos empilhados, S representa o número de módulos que serão utilizados, $L^{'}$ é o número reduzido de neurônios do método *Stacked* ELM, e $\tilde{\boldsymbol{V}}$ é a matriz de projeção utilizada no módulo para propagar informação entre módulos.

Quanto a algumas operações matemáticas empregadas neste trabalho, a norma-p ℓ_p de um vetor $\boldsymbol{x}$ de n dimensões, representada por $\| \boldsymbol{x} \|_p$, é definida de acordo com a Equação 1.1,

$$\| \boldsymbol{x} \|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}, \tag{1.1}$$

onde as normas ℓ_1 e ℓ_2 são empregadas neste trabalho. A norma de Frobenius (PETERSEN; PEDERSEN, 2007) de uma matriz $\boldsymbol{A} = [\boldsymbol{a}_{1,\bullet}^{\mathsf{T}} \; \boldsymbol{a}_{2,\bullet}^{\mathsf{T}} \cdots \boldsymbol{a}_{n,\bullet}^{\mathsf{T}}]^{\mathsf{T}}$ com n linhas e m colunas, onde $\boldsymbol{a}_{j,\bullet}$ é a j-ésima linha de $\boldsymbol{A}$, é definida pela Equação 1.2,

$$\| \boldsymbol{A} \|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |a_{ij}|^2} = \sqrt{tr(\boldsymbol{A}^{\mathsf{T}} \boldsymbol{A})}, \tag{1.2}$$

onde $\boldsymbol{A}^{\mathsf{T}}$ representa a transposta da matriz $\boldsymbol{A}$, $tr(\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A})$ é o traço de $\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A}$ e $\| \boldsymbol{A} \|_{2,1}$ é a norma $\ell_{2,1}$ de $\boldsymbol{A}$, definida pela Equação 1.3,

$$\| \boldsymbol{A} \|_{2,1} = \sum_{i=1}^{n} \| \boldsymbol{a}_{1,\bullet} \|_2 = \sum_{i=1}^{n} \left(\sum_{j=1}^{m} |a_{ij}|^2 \right)^{1/2}. \tag{1.3}$$

A operação de produto interno entre dois vetores $\boldsymbol{a}$ e $\boldsymbol{b}$, ambos de dimensão n, é representada pela notação $\boldsymbol{a} \cdot \boldsymbol{b}$, e é definida pela Equação 1.4,

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i, \tag{1.4}$$

enquanto o produto interno entre duas matrizes com o mesmo número de linhas $\boldsymbol{A} = [\boldsymbol{a}_{\bullet,1}, \ldots, \boldsymbol{a}_{\bullet,n}]$ e $\boldsymbol{R} = [\boldsymbol{r}_{\bullet,1}, \ldots, \boldsymbol{r}_{\bullet,m}]$, representado por $\langle \boldsymbol{A}, \boldsymbol{R} \rangle$, é definido de acordo com a Equação 1.5,

$$\langle \boldsymbol{A}, \boldsymbol{R} \rangle = tr(\boldsymbol{A}^{\mathsf{T}} \boldsymbol{R}), \tag{1.5}$$

enquanto a concatenação horizontal de $\boldsymbol{A}$ e $\boldsymbol{R}$, representada pela notação $[\boldsymbol{A}, \boldsymbol{R}]$, é definida de acordo com a Equação 1.6,

$$[\boldsymbol{A}, \boldsymbol{R}] = [\boldsymbol{a}_{\bullet,1} \cdots \boldsymbol{a}_{\bullet,n} \; \boldsymbol{r}_{\bullet,1} \cdots \boldsymbol{r}_{\bullet,m}]. \tag{1.6}$$

Em problemas de otimização com múltiplos objetivos, como em um caso onde se deseja encontrar um x que ao mesmo tempo minimize duas funções $f(x)$ e $g(x)$, sendo que múltiplas soluções razoáveis podem ser obtidas, é comum considerar um compromisso entre as duas funções consideradas para encontrar uma solução única. Uma das formas de

alcançar isso é minimizando uma soma ponderada das funções desejadas (MIETTINEN, 1999).

Exemplos de regressão regularizada como a regularização de Tikhonov (TIKHONOV; ARSENIN, 1977), podem ser vistos como problemas de otimização multiobjetivo onde os pesos da ponderação são chamados de parâmetros de regularização. Estes parâmetros controlam um compromisso entre várias funções distintas, por exemplo entre o erro de uma rede neural em um conjunto de treino contra a ordem de grandeza de seus parâmetros.

1.6 Publicações

Os seguintes trabalhos foram publicados durante esta Tese:

- SILVA, B. L. S. da; INABA, F. K.; CIARELLI, P. M. A FAST DEEP STACKED NETWORK USING EXTREME LEARNING MACHINE. In: Proceedings XXII Congresso Brasileiro de Automática. SBA Sociedade Brasileira de Automática, 2018. ISSN 2525-8311. Disponível em: ¡https://doi.org/10.20906/cps/cba2018-0809¿.
- SILVA, B. L. S. da; INABA, F. K.; CIARELLI, P. M.; SALLES, E. O. T. Outlier Robust Extreme Machine Learning for multi-target regression. Expert Systems with Applications, v. 140, p. 112877, feb 2020. ISSN 09574174. Disponível em: ¡https://linkinghub.elsevier.com/retrieve/pii/S0957417419305871¿.
- SILVA, B. L. S. da; INABA, F. K.; CIARELLI, P. M.; SALLES, E. O. T. *Fast Deep Stacked Networks Based on Extreme Learning Machine Applied to Regression Problems*. Neural Networks, v. 131, p. 14–28, 2020. ISSN 0893-6080. Disponível em: ¡https://www.sciencedirect.com/science/article/pii/S0893608020302598¿.

2 Referencial Teórico

Neste capítulo é apresentada uma breve revisão de literatura sobre as principais técnicas utilizadas neste trabalho. Na Seção 2.1 são apresentados algoritmos rápidos de treinamento de redes neurais de uma única camada oculta, bem como variações destes para lidar com múltiplas camadas ocultas, que consideram que não há a possibilidade de surgimento de novas amostras das bases de dados. Neste trabalho, estas técnicas serão chamadas de *offline*. Na Seção 2.2 são apresentados algoritmos de aprendizado de máquina *online*, que lidam com dados de treinamento que vão surgindo ao longo do tempo. Ainda são apresentados métodos de inicialização de pesos na Seção 2.3, comumente utilizados em conjunto com tais algoritmos, que muitas vezes melhoram os resultados obtidos por tais redes. Vale destacar que este trabalho foca principalmente nas técnicas *Deep Stacked Network*, *Regularized Extreme Learning Machine*, *Reduced Kernel Extreme Learning Machine* e *Online Sequential Regularized Extreme Learning Machine*.

2.1 Técnicas de Aprendizado de Máquina Offline

Nesta seção são apresentadas técnicas de aprendizado de máquina *offline*, como *Extreme Learning Machine* e algumas de suas variantes, além de técnicas empilhadas.

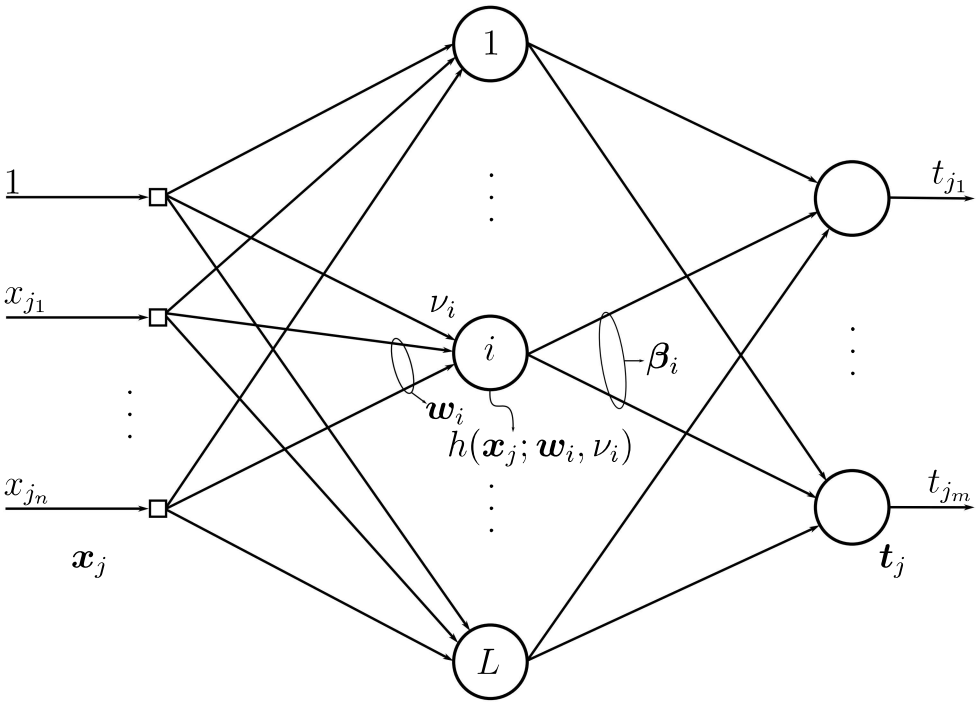
2.1.1 *Extreme Learning Machine*

O algoritmo ELM, proposto por Huang, Zhu e Siew (2004) para treinamento de SLFN, se tornou popular por ser rápido e obter modelos com desempenhos similares a técnicas clássicas de aprendizado de máquina, como o SVM e SLFNs treinadas com algoritmos baseados em *backpropagation* (HUANG; ZHU; SIEW, 2006).

A arquitetura SLFN possui apenas uma camada oculta de neurônios, como mostrado na Figura 1, e dois conjuntos de pesos que devem ser determinados em sua etapa de treinamento. Ao treinar tal arquitetura usando o algoritmo ELM, o conjunto de pesos entre a entrada e a camada oculta é escolhido de forma aleatória, de acordo com qualquer distribuição de probabilidade contínua, e os pesos de saída são obtidos utilizando a inversa generalizada de Moore-Penrose da matriz de saída da camada oculta(HUANG; ZHU; SIEW, 2006).

Considerando um conjunto de dados com N amostras distintas e rotuladas $(\boldsymbol{x}_j, \boldsymbol{t}_j)$, $j = 1, \ldots, N$, onde $\boldsymbol{x}_j = [x_{j1} \cdots x_{jn}]^\mathsf{T} \in \mathbb{R}^n$ e $\boldsymbol{t}_j \in \mathbb{R}^m$, pode-se dizer que a saída estimada

Figura 1 – Um exemplo de arquitetura de rede neural do tipo SLFN.



Fonte: Adaptado de (INABA et al., 2018)

$\hat{\boldsymbol{t}}_j$ de uma SLFN para qualquer amostra $\boldsymbol{x}_j$ é obtida pela Equação 2.1,

$$\hat{\boldsymbol{t}}_j = \sum_{i=1}^{L} \boldsymbol{\beta}_i h(\boldsymbol{w}_i; \boldsymbol{x}_j; \nu_i), \tag{2.1}$$

onde $\boldsymbol{w}_i = [w_{i1} \cdots w_{in}]^\mathsf{T}$ é o vetor de pesos que liga os nós de entrada ao i-ésimo nó oculto, $\boldsymbol{\beta}_i = [\beta_{i1} \cdots \beta_{im}]$ é o vetor de pesos que liga o i-ésimo nó oculto aos nós de saída, ν_i é o viés do i-ésimo nó oculto, e $h(\cdot)$ é uma função de ativação contínua.

Considerando que essa arquitetura pode aproximar todas as N amostras com erro igual a zero[1], i.e., $\hat{\boldsymbol{t}}_j = \boldsymbol{t}_j$, é possível usar a notação matricial para agrupar a saída de todas as amostras, como visto na Equação 2.2,

$$\boldsymbol{H}\boldsymbol{B} = \boldsymbol{T}, \tag{2.2}$$

onde $\boldsymbol{B} = [\boldsymbol{\beta}_1 \cdots \boldsymbol{\beta}_L]^\mathsf{T}$, $\boldsymbol{T} = [\boldsymbol{t}_1 \cdots \boldsymbol{t}_N]^\mathsf{T}$, $\boldsymbol{H}$ é dado pela Equação 2.3,

$$\boldsymbol{H}(\boldsymbol{X}; \boldsymbol{W}, \boldsymbol{\nu}) = \begin{bmatrix} h(\boldsymbol{x}_1; \boldsymbol{w}_1, \nu_1) & \dots & h(\boldsymbol{x}_1; \boldsymbol{w}_L, \nu_L) \\ \vdots & \ddots & \vdots \\ h(\boldsymbol{x}_N; \boldsymbol{w}_1, \nu_1) & \dots & h(\boldsymbol{x}_N; \boldsymbol{w}_L, \nu_L) \end{bmatrix}_{N \times L}, \tag{2.3}$$

[1] Mais detalhes são disponibilizados em (HUANG; ZHU; SIEW, 2004).

onde $\boldsymbol{W} = [\boldsymbol{w}_1 \cdots \boldsymbol{w}_L]$, $\boldsymbol{\nu} = [\nu_1 \cdots \nu_L]^\mathsf{T}$ e $\boldsymbol{X} = [\boldsymbol{x}_1^\mathsf{T} \cdots \boldsymbol{x}_N^\mathsf{T}]^\mathsf{T}$.

No contexto do ELM, $\boldsymbol{W}$ e $\boldsymbol{\nu}$ são amostrados aleatoriamente de acordo com uma distribuição contínua e os pesos de saída $\boldsymbol{B}$ são calculados de forma a encontrar a solução que minimiza o erro quadrático médio, como pode ser visto na Equação 2.4,

$$\hat{\boldsymbol{B}} = \arg\min_{\boldsymbol{B}} \| \boldsymbol{H}\boldsymbol{B} - \boldsymbol{T} \|_F^2, \tag{2.4}$$

que possui uma solução analítica, mostrada na Equação 2.5,

$$\boldsymbol{B} = \boldsymbol{H}^\dagger \boldsymbol{T}, \tag{2.5}$$

onde $\boldsymbol{H}^\dagger$ é a inversa generalizada de Moore-Penrose (RAO; MITRA, 1971) da matriz $\boldsymbol{H}$, que pode ser determinada usando o método de projeção ortogonal em dois casos (RAO; MITRA, 1971): quando $\boldsymbol{H}^\mathsf{T}\boldsymbol{H}$ é não singular, então $\boldsymbol{H}^\dagger = (\boldsymbol{H}^\mathsf{T}\boldsymbol{H})^{-1}\boldsymbol{H}^\mathsf{T}$ (INABA, 2018).

2.1.2 *Regularized Extreme Learning Machine*

Embora tenha apresentado bons resultados em diversas aplicações, uma SLFN treinada pelo algoritmo ELM possui uma grande sensibilidade ao número de neurônios em sua camada oculta. Isso significa que problemas de sobreajuste (*overfitting*) e subajuste (*underfitting*) ocorrem quando o tamanho da rede (o valor do hiperparâmetro L) não é devidamente escolhido.

Para evitar estes tipos de problemas e obter uma melhor generalização e uma solução mais estável, Deng, Zheng e Chen (2009) propuseram o *Weighted Regularized Extreme Learning Machine* (WR-ELM), que resolve o problema de otimização apresentado na Equação 2.6,

$$\begin{aligned} \underset{\boldsymbol{B}}{\text{minimizar}} \quad & \frac{C}{2} \| \boldsymbol{D}\boldsymbol{\Theta} \|_F^2 + \frac{1}{2} \| \boldsymbol{B} \|_F^2 \\ \text{sujeito à} \quad & \boldsymbol{H}\boldsymbol{B} - \boldsymbol{T} = \boldsymbol{\Theta}, \end{aligned} \tag{2.6}$$

onde $\boldsymbol{\Theta}$ é a matriz de erro, C é o parâmetro de regularização e $\boldsymbol{D}$ é uma matriz diagonal de pesos adicionados ao modelo com o objetivo de melhorar a robustez à *outliers*. Neste caso, a solução do problema de otimização apresentado na Equação 2.6 é dada pela Equação 2.7,

$$\boldsymbol{B} = \left(\frac{\boldsymbol{I}}{C} + \boldsymbol{H}^\mathsf{T}\boldsymbol{D}\boldsymbol{H} \right)^{-1} \boldsymbol{H}^\mathsf{T}\boldsymbol{D}\boldsymbol{T}, \tag{2.7}$$

e caso a matriz $\boldsymbol{D}$ seja igual à matriz identidade, $\boldsymbol{B}$ é dado pela Equação 2.8,

$$\boldsymbol{B} = \left(\frac{\boldsymbol{I}}{C} + \boldsymbol{H}^\mathsf{T}\boldsymbol{H} \right)^{-1} \boldsymbol{H}^\mathsf{T}\boldsymbol{T}. \tag{2.8}$$

que é a solução denominada *Unweighted Regularized Extreme Learning Machine* (UWR-ELM). Vale notar que o algoritmo ELM é um caso especial de UWR-ELM quando $C \to \infty$ (DENG; ZHENG; CHEN, 2009).

Outros trabalhos estenderam o conceito de UWR-ELM, onde Huang et al. (2012) analisaram o impacto do desempenho das redes treinadas com o algoritmo usando diversas funções de ativação na camada oculta. Uma extensão do método que usa o conceito de *kernels* também foi proposta por estes autores.

Martínez-Martínez et al. (2011) também estenderam o conceito de UWR-ELM considerando diferentes regularizações no cálculo dos pesos de saída em seu algoritmo. A técnica, chamada de *Regularized Extreme Learning Machine* (R-ELM), considera apenas SLFN com um neurônio na camada de saída, e resolve o problema de otimização de norma vetorial ℓ_1, apresentado na Equação 2.9,

$$\underset{(\beta_0,\boldsymbol{\beta})\in\mathbb{R}^{L+1}}{\text{minimizar}} \frac{C}{2} \parallel \boldsymbol{H}\boldsymbol{\beta} + \beta_0 - \boldsymbol{t} \parallel_2^2 + \frac{(1-\alpha)}{2} \parallel \boldsymbol{\beta} \parallel_2^2 + \alpha \parallel \boldsymbol{\beta} \parallel_1, \tag{2.9}$$

onde β_0 também é um parâmetro de regularização, e α controla o *trade-off* entre o método de regularização Tikhonov ($\alpha = 0$) e o *least absolute shrinkage and selection operator* (LASSO) ($\alpha = 1$). Este tipo de regularização é chamado de *Elastic Net* (ZOU; HASTIE, 2005).

A ideia de se utilizar a *Elastic Net* é de identificar o grau de relevância dos pesos que conectam os nós de camada oculta aos da saída e podar aqueles considerados irrelevantes, sem comprometer a capacidade de generalização do modelo treinado (MARTÍNEZ-MARTÍNEZ et al., 2011).

É possível notar que UWR-ELM é um caso particular de R-ELM, quando $\beta_0 = 0$ e $\alpha = 0$, ou seja, se apenas a regularização Tikhonov for utilizada, o que faz com que a rede perca a habilidade de podar neurônios de forma eficiente, embora continue apresentando boa generalização (MARTÍNEZ-MARTÍNEZ et al., 2011). Apesar de ser um caso particular, muitos autores (INABA et al., 2018; SILVA et al., 2020b; ZHANG; LUO, 2015) se referem ao UWR-ELM como R-ELM.

2.1.3 *Generalized Regularized Extreme Learning Machine*

Embora os algoritmos ELM, WR-ELM e UWR-ELM tenham sido propostos considerando múltiplas saídas (HUANG; ZHU; SIEW, 2006; DENG; ZHENG; CHEN, 2009), o R-ELM foi definido por Martínez-Martínez et al. (2011) para o treinamento de SLFN com apenas uma saída. Para casos multi-dimensionais, porém, é possível considerar cada saída separadamente e treinar os neurônios de saída um independente do outro (para problemas de classificação, essa estratégia de treinamento possui o nome de *one-hot encoding*, onde cada classe é representada por um neurônio diferente na rede neural, e uma função do tipo *argmax* é usada para associar uma amostra à sua classe). Isto pode ser considerado uma limitação dos algoritmos, já que possíveis relações entre as saídas, que podem ser importantes em problemas de classificação *multi-label* e regressão *multi-target*, não são consideradas no problema de otimização.

Considerando o algoritmo R-ELM com $\alpha \neq 0$, i.e., quando o LASSO é utilizado, é possível podar os neurônios menos importantes da arquitetura com baixo impacto no desempenho. Ao treinar cada saída separadamente, um neurônio deve ser podado no treinamento de todas as saídas simultaneamente para que ele possa ser efetivamente removido da rede, o que torna, nestes casos, a tarefa de poda mais difícil.

Para lidar com tais limitações, Inaba et al. (2018) propuseram a utilização de normas matriciais no problema de otimização resolvido para encontrar os pesos de saída da SLFN a ser treinada. Assim, o algoritmo *Generalized Regularized Extreme Learning Machine* (GR-ELM) foi proposto.

Portanto, a principal diferença entre o R-ELM e o GR-ELM é que o segundo substitui as normas ℓ_1 e ℓ_2 do problema de otimização da Equação 2.9 pelas normas matriciais $\ell_{2,1}$ e Frobenius, respectivamente, além de considerar que a saída da rede para uma amostra é um vetor com m dimensões, ao invés de apenas uma.

Considerando uma base de dados com N amostras distintas e saídas com dimensão m, $(\boldsymbol{x}_j, \boldsymbol{t}_j)$, onde $\boldsymbol{x}_j = [x_{j1} \cdots x_{jn}] \in \mathbb{R}^n$ e $\boldsymbol{t}_j = [t_{j1} \cdots t_{jm}] \in \mathbb{R}^m$, é possível reescrever a Equação 2.2 na forma apresentada na Equação 2.10,

$$\boldsymbol{H}\boldsymbol{B} = \boldsymbol{T}, \tag{2.10}$$

onde $\boldsymbol{T} = [\boldsymbol{t}_1^\intercal \cdots \boldsymbol{t}_N^\intercal]^\intercal$ e $\boldsymbol{B}$ é a matriz de pesos conectando a camada oculta e de saída da SLFN, i.e., $\boldsymbol{B} = [\boldsymbol{\beta}_1 \ \boldsymbol{\beta}_2 \cdots \boldsymbol{\beta}_m] \in \mathbb{R}^{L \times m}$. Assim, o problema de otimização resolvido no algoritmo GR-ELM é dado pela Equação 2.11,

$$\begin{aligned} \underset{\boldsymbol{B},\boldsymbol{Z}}{\text{minimizar}} \quad & \frac{C}{2} \parallel \boldsymbol{H}\boldsymbol{B} - \boldsymbol{T} \parallel_F^2 + \frac{\lambda(1-\alpha)}{2} \parallel \boldsymbol{B} \parallel_F^2 + \lambda\alpha \parallel \boldsymbol{Z} \parallel_{2,1} \\ \text{sujeito à} \quad & \boldsymbol{B} - \boldsymbol{Z} = \boldsymbol{0}, \end{aligned} \tag{2.11}$$

onde C, λ e α são parâmetros de regularização, e $\boldsymbol{Z}$ é uma matriz auxiliar no problema de otimização.

Este problema de otimização é resolvido utilizando o método *Alternating Direction Method of Multipliers* (ADMM) (BOYD, 2010). Como a função objetivo da Equação 2.11 é separável em $\boldsymbol{B}$ e $\boldsymbol{Z}$, o ADMM realiza uma minimização alternada da função Lagrangiana aumentada sobre tais variáveis em cada uma de suas iterações.

A função Lagrangiana aumentada da Equação 2.11, usada no ADMM, é dada pela Equação 2.12,

$$\begin{aligned} \mathcal{L}(\boldsymbol{B}, \boldsymbol{Z}, \boldsymbol{Y}) = & \frac{C}{2} \parallel \boldsymbol{H}\boldsymbol{B} - \boldsymbol{T} \parallel_F^2 + \frac{\lambda(1-\alpha)}{2} \parallel \boldsymbol{B} \parallel_F^2 + \lambda\alpha \parallel \boldsymbol{Z} \parallel_{2,1} + \\ & \frac{\rho}{2} \parallel \boldsymbol{B} - \boldsymbol{Z} \parallel_F^2 + \langle \boldsymbol{Y}, \boldsymbol{B} - \boldsymbol{Z} \rangle, \end{aligned} \tag{2.12}$$

onde $\boldsymbol{Y}$ é a matriz de multiplicadores de Lagrange e ρ é um parâmetro de regularização.

De acordo com Inaba et al. (2018), é possível reescrever a Equação 2.12 usando a variável dual escalonada $\boldsymbol{U} = (1/\rho)\boldsymbol{Y}$, como pode ser visto na Equação 2.13,

$$\mathcal{L}(\boldsymbol{B}, \boldsymbol{Z}, \boldsymbol{Y}) = \frac{C}{2} \parallel \boldsymbol{HB} - \boldsymbol{T} \parallel_F^2 + \frac{\lambda(1-\alpha)}{2} \parallel \boldsymbol{B} \parallel_F^2 + \lambda\alpha \parallel \boldsymbol{Z} \parallel_{2,1} + \frac{\rho}{2} \parallel \boldsymbol{B} - \boldsymbol{Z} + \boldsymbol{U} \parallel_F^2 - \frac{\rho}{2} \parallel \boldsymbol{U} \parallel_F^2 . \tag{2.13}$$

Assim, o algoritmo ADMM otimiza a função Lagrangiana aumentada da Equação 2.13 alternadamente entre $\boldsymbol{B}$ e $\boldsymbol{Z}$, atualizando a variável dual escalonada ao final de cada iteração. As variáveis $\boldsymbol{B}$, $\boldsymbol{Z}$ e $\boldsymbol{U}$ são iniciadas com todos os elementos iguais a zero, isto é, $\boldsymbol{B}^0 = \boldsymbol{0}$, $\boldsymbol{Z}^0 = \boldsymbol{0}$ e $\boldsymbol{U}^0 = \boldsymbol{0}$. A atualização da variável $\boldsymbol{B}$ na k-ésima iteração do ADMM é dada pela Equação 2.14,

$$\boldsymbol{B}^{k+1} = (\boldsymbol{H}^\top \boldsymbol{H} + \eta \boldsymbol{I})^{-1} \left[\boldsymbol{H}^\top \boldsymbol{T} + \frac{\rho}{C} \left(\boldsymbol{Z}^k - \boldsymbol{U}^k \right) \right], \tag{2.14}$$

onde $\eta = (1-\alpha) + \rho/C$. A atualização da variável $\boldsymbol{Z}$ na k-ésima iteração do ADMM é feita separadamente para cada linha da matriz $\boldsymbol{Z} = [\boldsymbol{z}_{1,\bullet} \cdots \boldsymbol{z}_{L,\bullet}]$, como pode ser visto na Equação 2.15,

$$\boldsymbol{z}_{i,\bullet}^{k+1} = S_{\frac{\alpha}{\rho}} \left(\boldsymbol{b}_{i,\bullet}^{k+1} + \boldsymbol{u}_{i,\bullet}^{k} \right) \tag{2.15}$$

onde $S_\kappa : \mathbb{R}^m \to \mathbb{R}$ é o operador *block soft-thresholding* (BOYD, 2010), definido na Equação 2.16

$$S_\kappa(\boldsymbol{a}) = \left(1 - \frac{\kappa}{\parallel \boldsymbol{a} \parallel_2} \right)_+ \boldsymbol{a}, \tag{2.16}$$

onde $\kappa \in \mathbb{R}$, $(d)_+ = max(0, d)$ e $S_\kappa(\boldsymbol{0}) = \boldsymbol{0}$. Este operador "encolhe" o seu argumento para zero se ele for suficientemente pequeno e, caso contrário, ele o "move" por uma quantidade κ em direção à origem. Quando o argumento do operador *block soft-thresholding* é um escalar, o resultado é equivalente à aplicação do *soft-thresholding*. Por fim, a regra de atualização para os multiplicadores de Lagrange é dada pela Equação 2.17,

$$\boldsymbol{U}^{k+1} = \boldsymbol{U}^k + \rho \left(\boldsymbol{B}^{k+1} - \boldsymbol{Z}^{k+1} \right). \tag{2.17}$$

O ADMM repete tais atualizações até que o critério de parada seja atingido. Inaba et al. (2018) propuseram parar o algoritmo quando os resíduos dual e primal, dados pelas Equações 2.18 e 2.19, atingirem as tolerâncias dadas pelas Equações 2.20 e 2.21, respectivamente.

$$\boldsymbol{R}^k = \boldsymbol{B}^k - \boldsymbol{Z}^k \tag{2.18}$$

$$\boldsymbol{S}^k = \rho \left(\boldsymbol{Z}^k - \boldsymbol{Z}^{k-1} \right) \tag{2.19}$$

$$\parallel \boldsymbol{R}^k \parallel_F \leq \sqrt{mL}\epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\{\parallel \boldsymbol{B}^k \parallel_F, \parallel \boldsymbol{Z}^k \parallel_F\} \tag{2.20}$$

$$\| \boldsymbol{S}^k \|_F \leq \sqrt{mL}\epsilon^{\text{abs}} + \epsilon^{\text{rel}}\rho \| \boldsymbol{U}^k \|_F, \tag{2.21}$$

onde $\epsilon^{\text{abs}} > 0$ e $\epsilon^{\text{rel}} > 0$ são tolerâncias pré-definidas. Inaba et al. (2018) sugerem a utilização de tolerâncias $\epsilon^{\text{abs}} = 10^{-3}$ e $\epsilon^{\text{rel}} = 10^{-2}$.

O Algoritmo 1 resume o método GR-ELM.

Algoritmo 1: GR-ELM: *Generalized Regularized Extreme Learning Machine*

Requer: Amostras e alvos de Treino ($\boldsymbol{X}$,$\boldsymbol{T}$), onde $\boldsymbol{X} = [\boldsymbol{x}_1\ \boldsymbol{x}_2 \cdots \boldsymbol{x}_N]^\mathsf{T}$, função de ativação $h(\cdot)$, número de nós iniciais na camada oculta $\tilde{N}$, e parâmetros de regularização λ, α, C e ρ.
Inicialização: $\boldsymbol{B}^0 = \mathbf{0}$, $\boldsymbol{U}^0 = \mathbf{0}$, $\boldsymbol{Z}^0 = \mathbf{0}$ e $k = 0$;
Gera pesos $\mathbf{A}$ e vieses $\boldsymbol{\nu}$ aleatoriamente.
Calcula a matriz da camada oculta $\boldsymbol{H}$ usando Equação 2.2.
repita
  $\mathbf{B}^{k+1} := \arg\min_{\mathbf{B}} \frac{C}{2} \| \mathbf{HB} - \mathbf{T} \|_F^2 + \frac{\lambda(1-\alpha)}{2} \| \mathbf{B} \|_F^2 + \frac{\rho}{2} \| \mathbf{B} - \mathbf{Z}^k + \mathbf{U}^k \|_F^2$
  $\mathbf{Z}^{k+1} := \arg\min_{\mathbf{Z}} \frac{\lambda\alpha}{\rho} \| \mathbf{Z} \|_{2,1} + \frac{1}{2} \| \mathbf{B}^{k+1} - \mathbf{Z} + \mathbf{U}^k \|_F^2$
  $\mathbf{U}^{k+1} := \mathbf{U}^k + \rho \left(\mathbf{B}^{k+1} - \mathbf{Z}^{k+1} \right)$
  k = k+1
até *atingir critério de parada*;

2.1.4 *Outlier Robust Extreme Learning Machine*

Devido à várias causas, como erro de instrumentos de medição ou erros humanos, bases de dados são contaminadas com amostras que podem estar bem afastadas das demais, que são chamadas de *outliers* e usualmente implicam em um prejuízo ao desempenho dos modelos que irão analisá-los (SILVA et al., 2020b).

No trabalho de Zhang e Luo (2015), o desempenho de algoritmos como o ELM e o R-ELM foram avaliados na presença de *outliers*, e foi constatada uma degradação quando estes elementos estão presentes. Isto se deve ao fato de que estes algoritmos consideram a norma ℓ_2 do erro de treino, que é uma relação bem estabelecida na literatura.

Uma das formas de contornar tal problema é considerar normas mais esparsas do erro, como a norma ℓ_0, ao invés da norma ℓ_2. Porém, problemas de otimização analítica que empregam a norma ℓ_0 não são convexos, o que dificulta encontrar suas soluções. Isto pode ser contornado ao realizar um relaxamento sem perder a característica de esparsidade, substituindo a norma ℓ_0 pela norma ℓ_1 (ZHANG; LUO, 2015).

Assim, Zhang e Luo (2015) propõem alterar o problema de otimização da Equação 2.9 substituindo a norma ℓ_2 do erro pela ℓ_1, e utilizando apenas a regularização Tikhonov, dando origem assim ao algoritmo *Outlier Robust Extreme Learning Machine* (OR-ELM). Assim como o algoritmo R-ELM, o OR-ELM lida apenas com SLFN com apenas um

neurônio de saída. Isto resulta no problema apresentado na Equação 2.22,

$$\begin{aligned} \underset{\boldsymbol{\beta}}{\text{minimizar}} \quad & \tau \parallel \boldsymbol{\epsilon} \parallel_1 + \frac{1}{2} \parallel \boldsymbol{\beta} \parallel_2^2 \\ \text{sujeito à} \quad & \boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{t} = \boldsymbol{\epsilon}, \end{aligned} \tag{2.22}$$

onde τ é um parâmetro de regularização. O método *Augmented Lagrange Multiplier Method* (ALM) é utilizado para resolver o problema da Equação 2.22.

Testes realizados mostraram que o desempenho do OR-ELM é superior ao do R-ELM quando a base de dados possui dados contaminados por *outliers*. Contudo, tal método é definido apenas para redes com uma saída, e a rede resultante é densa, o que necessita de uma escolha apropriada do número de neurônios na camada oculta (ZHANG; LUO, 2015).

2.1.5 *Generalized Outlier Robust Extreme Learning Machine*

Da mesma forma que o algoritmo GR-ELM foi definido como uma extensão do método R-ELM para múltiplas saídas, o método *Generalized Outlier Robust Extreme Learning Machine* (GOR-ELM) foi proposto como uma extensão do método OR-ELM (INABA, 2018; SILVA et al., 2020b).

Tal método foi proposto para lidar com problemas de regressão de múltiplas saídas, de valores reais e de forma simultânea a partir do mesmo conjunto de dados de entrada. Uma forma natural de lidar com problemas deste tipo é obter um modelo separado para cada saída. Contudo, é desejável que as relações entre as saídas sejam consideradas na modelagem (ZHEN et al., 2018).

De forma a considerar conjuntamente as relações entre saídas e a robustez à *outliers*, Inaba (2018) propôs o método GOR-ELM, que quando comparado ao método OR-ELM, considera a regularização *Elastic Net* e substitui as normas vetoriais ℓ_1 e ℓ_2 pelas normas matriciais $\ell_{2,1}$ e Frobenius. Segundo Ma et al. (2017), a norma $\ell_{2,1}$ apresenta robustez à *outliers*, assim como a norma ℓ_1. Assim, o algoritmo GOR-ELM também resulta em uma rede robusta.

O algoritmo GOR-ELM resolve o problema de otimização apresentado na Equação 2.23,

$$\begin{aligned} \underset{\boldsymbol{E},\boldsymbol{B},\boldsymbol{Z}}{\text{minimizar}} \quad & \tau \parallel \boldsymbol{E} \parallel_{2,1} + \frac{\lambda}{2}(1-\alpha) \parallel \boldsymbol{B} \parallel_F^2 + \lambda\alpha \parallel \boldsymbol{Z} \parallel_{2,1} \\ \text{sujeito à} \quad & \boldsymbol{E} = \boldsymbol{H}\boldsymbol{B} - \boldsymbol{T} \\ & \boldsymbol{B} - \boldsymbol{Z} = \boldsymbol{0}, \end{aligned} \tag{2.23}$$

cuja função objetivo é separável em $\boldsymbol{E}$, $\boldsymbol{B}$ e $\boldsymbol{Z}$.

A Equação 2.23 pode ser reescrita como visto na Equação 2.24,

$$\begin{aligned} \underset{\boldsymbol{E},\boldsymbol{B},\boldsymbol{Z}}{\text{minimizar}} \quad & \tau \parallel \boldsymbol{E} \parallel_{2,1} + \frac{\lambda(1-\alpha)}{2} \parallel \boldsymbol{B} \parallel_F^2 + \lambda\alpha \parallel \boldsymbol{Z} \parallel_{2,1} \\ \text{sujeito à} \quad & \widetilde{\boldsymbol{A}}\boldsymbol{E} + \widetilde{\boldsymbol{B}}\boldsymbol{B} + \widetilde{\boldsymbol{C}}\boldsymbol{Z} + \widetilde{\boldsymbol{D}} = \boldsymbol{0}, \end{aligned} \tag{2.24}$$

onde $\tilde{\boldsymbol{A}} = [-\boldsymbol{I}_N\,,\, \boldsymbol{0}_{N\times L}]^\mathsf{T}$, $\widetilde{\boldsymbol{B}} = \left[\boldsymbol{H}^\mathsf{T}\,,\, \boldsymbol{I}_L\right]^\mathsf{T}$, $\widetilde{\boldsymbol{C}} = [\boldsymbol{0}_{L\times N}\,,\, -\boldsymbol{I}_L]^\mathsf{T}$, e $\widetilde{\boldsymbol{D}} = \left[-\boldsymbol{T}^\mathsf{T}\,,\, \boldsymbol{0}_{m\times L}\right]^\mathsf{T}$, onde o operador $[\cdot,\cdot]$ representa a concatenação horizontal de duas matrizes. De acordo com Chen et al. (2016), o problema de otimização da Equação 2.24 pode ser resolvido usando um algoritmo ADMM de 3 blocos, que não possui garantia de convergência. Porém, uma condição suficiente para a convergência é que pelo menos uma destas afirmações seja verdadeira: $\tilde{\boldsymbol{A}}^\mathsf{T}\tilde{\boldsymbol{B}} = \boldsymbol{0}$, $\tilde{\boldsymbol{A}}^\mathsf{T}\tilde{\boldsymbol{C}} = \boldsymbol{0}$ ou $\tilde{\boldsymbol{B}}^\mathsf{T}\tilde{\boldsymbol{C}} = \boldsymbol{0}$. Como a afirmação $\tilde{\boldsymbol{A}}^\mathsf{T}\tilde{\boldsymbol{C}} = \boldsymbol{0}$ é verdadeira para o GOR-ELM, o algoritmo converge.

A função Lagrangiana aumentada da Equação 2.24 é visto na Equação 2.25,

$$\begin{aligned} L(\boldsymbol{E},\boldsymbol{B},\boldsymbol{Z},\boldsymbol{Y}) = {} & \tau \parallel \boldsymbol{E} \parallel_{2,1} + \frac{\lambda(1-\alpha)}{2} \parallel \boldsymbol{B} \parallel_F^2 + \lambda\alpha \parallel \boldsymbol{Z} \parallel_{2,1} \\ & + \frac{\rho}{2} \parallel \widetilde{\boldsymbol{A}}\boldsymbol{E} + \widetilde{\boldsymbol{B}}\boldsymbol{B} + \widetilde{\boldsymbol{C}}\boldsymbol{Z} + \widetilde{\boldsymbol{D}} \parallel_F^2 \\ & + \langle \boldsymbol{Y}, \widetilde{\boldsymbol{A}}\boldsymbol{E} + \widetilde{\boldsymbol{B}}\boldsymbol{B} + \widetilde{\boldsymbol{C}}\boldsymbol{Z} + \widetilde{\boldsymbol{D}} \rangle, \end{aligned} \tag{2.25}$$

onde $\boldsymbol{Y}$ representa os multiplicadores de Lagrange e $\rho > 0$ é um parâmetro de regularização.

Usando a variável dual escalada $\boldsymbol{U} = (1/\rho)\boldsymbol{Y}$, pode-se reescrever a função Lagrangiana aumentada como visto na Equação 2.26,

$$\begin{aligned} L(\boldsymbol{E},\boldsymbol{B},\boldsymbol{Z},\boldsymbol{U}) = {} & \tau \parallel \boldsymbol{E} \parallel_{2,1} + \frac{\lambda(1-\alpha)}{2} \parallel \boldsymbol{B} \parallel_F^2 + \lambda\alpha \parallel \boldsymbol{Z} \parallel_{2,1} \\ & + \frac{\rho}{2} \parallel \widetilde{\boldsymbol{A}}\boldsymbol{E} + \widetilde{\boldsymbol{B}}\boldsymbol{B} + \widetilde{\boldsymbol{C}}\boldsymbol{Z} + \widetilde{\boldsymbol{D}} + \boldsymbol{U} \parallel_F^2 - \frac{\rho}{2} \parallel \boldsymbol{U} \parallel_F^2 . \end{aligned} \tag{2.26}$$

Como a norma Frobenius é separável, a função Lagrangiana pode ser escrita como visto na Equação 2.27,

$$\begin{aligned} L(\boldsymbol{E},\boldsymbol{B},\boldsymbol{Z},\boldsymbol{U}) = {} & \tau \parallel \boldsymbol{E} \parallel_{2,1} + \frac{\lambda(1-\alpha)}{2} \parallel \boldsymbol{B} \parallel_F^2 + \lambda\alpha \parallel \boldsymbol{Z} \parallel_{2,1} \\ & + \frac{\rho}{2} \parallel \boldsymbol{H}\boldsymbol{B} - \boldsymbol{T} - \boldsymbol{E} + \boldsymbol{U}_1 \parallel_F^2 + \frac{\rho}{2} \parallel \boldsymbol{B} - \boldsymbol{Z} + \boldsymbol{U}_2 \parallel_F^2 \\ & - \frac{\rho}{2} \parallel \boldsymbol{U}_1 \parallel_F^2 - \frac{\rho}{2} \parallel \boldsymbol{U}_2 \parallel_F^2, \end{aligned} \tag{2.27}$$

onde $\boldsymbol{U} = \begin{bmatrix}\boldsymbol{U}_1^\mathsf{T} & \boldsymbol{U}_2^\mathsf{T}\end{bmatrix}^\mathsf{T}$, com $\boldsymbol{U}_1 \in \mathbb{R}^{N\times m}$ e $\boldsymbol{U}_2 \in \mathbb{R}^{L\times m}$.

Assim, o algoritmo ADMM de 3 blocos otimiza a função Lagrangiana aumentada da Equação 2.27 alternadamente entre $\boldsymbol{B}$, $\boldsymbol{Z}$ e $\boldsymbol{E}$, atualizando os multiplicadores de Lagrange ao final de cada iteração. A atualização da variável $\boldsymbol{B}$ na k-ésima iteração é dada pela Equação 2.28,

$$\boldsymbol{B}^{k+1} = \left(\boldsymbol{H}^\mathsf{T}\boldsymbol{H} + \eta\boldsymbol{I}\right)^{-1}\left[\boldsymbol{H}^\mathsf{T}\left(\boldsymbol{T} + \boldsymbol{E}^k - \boldsymbol{U}_1^k\right) + \left(\boldsymbol{Z}^k - \boldsymbol{U}_2^k\right)\right], \tag{2.28}$$

onde $\eta = (\lambda(1-\alpha)+\rho)/\rho$.

Da mesma forma que é feito no método GR-ELM, a atualização da variável $\boldsymbol{Z}$ na k-ésima iteração é dada pela Equação 2.29,

$$\boldsymbol{Z}^{k+1} = \mathcal{S}_{\frac{\lambda\alpha}{\rho}}\left(\boldsymbol{B}^{k+1} + \boldsymbol{U}_2^k\right), \tag{2.29}$$

onde $\mathcal{S}_\kappa(\boldsymbol{A})$ é uma função que aplica o operador *block soft-thresholding* S_κ (BOYD, 2010) em cada linha de $\boldsymbol{A}$. O operador S_κ foi definido na Seção 2.1.3.

O terceiro passo da k-ésima iteração do algoritmo ADMM de 3 blocos é atualizar a variável $\boldsymbol{E}$, que obedece à Equação 2.30,

$$\boldsymbol{E}^{k+1} = \mathcal{S}_{\frac{\tau}{\rho}}\left(\boldsymbol{H}\boldsymbol{B}^{k+1} - \boldsymbol{T} + \boldsymbol{U}_1^k\right). \tag{2.30}$$

Finalmente, a regra de atualização para os multiplicadores de Lagrange é dada na Equação 2.32,

$$\boldsymbol{U}_1^{k+1} = \boldsymbol{U}_1^k + \left(\boldsymbol{H}\boldsymbol{B}^{k+1} - \boldsymbol{T} - \boldsymbol{E}^{k+1}\right) \tag{2.31}$$

$$\boldsymbol{U}_2^{k+1} = \boldsymbol{U}_2^k + \left(\boldsymbol{B}^{k+1} - \boldsymbol{Z}^{k+1}\right). \tag{2.32}$$

O ADMM de 3 blocos repete as atualizações em cada iteração até que o critério de parada seja atingido. Silva et al. (2020b) propuseram parar o algoritmo quando os resíduos dual e primal, dados pelas Equações 2.33 e 2.34, atingirem as tolerâncias dadas pelas Equações 2.35 e 2.36, respectivamente. Ressalta-se que isto é uma contribuição alcançada durante o desenvolvimento desta Tese.

$$\boldsymbol{R}^k = \tilde{\boldsymbol{A}}\boldsymbol{E}^{k+1} + \tilde{\boldsymbol{B}}\boldsymbol{B}^{k+1} + \tilde{\boldsymbol{C}}\boldsymbol{Z}^{k+1} + \tilde{\boldsymbol{D}} \tag{2.33}$$

$$\boldsymbol{S}^k = \rho\tilde{\boldsymbol{B}}^\mathsf{T}\tilde{\boldsymbol{A}}(\boldsymbol{E}^{k+1} - \boldsymbol{E}^k) + \rho\tilde{\boldsymbol{B}}^\mathsf{T}\tilde{\boldsymbol{C}}(\boldsymbol{Z}^{k+1} - \boldsymbol{Z}^k) \tag{2.34}$$

$$\boldsymbol{R}^k \leq \sqrt{mL}\epsilon^{abs} + \epsilon^{rel}\max(\parallel \tilde{\boldsymbol{A}}\boldsymbol{E}^k \parallel_F, \parallel \tilde{\boldsymbol{B}}\boldsymbol{B}^k \parallel_F, \parallel \tilde{\boldsymbol{C}}\boldsymbol{Z}^k \parallel_F, \parallel \tilde{\boldsymbol{D}} \parallel_F) \tag{2.35}$$

$$\boldsymbol{S}^k \leq \sqrt{mL}\epsilon^{abs} + \epsilon^{rel}\rho \parallel \boldsymbol{H}^\mathsf{T}\boldsymbol{U}_1 + \boldsymbol{U}_2 \parallel_F, \tag{2.36}$$

onde ϵ^{abs} e ϵ^{rel} são tolerâncias pré-definidas, com os mesmos valores sugeridos na Seção 2.1.3.

O Algoritmo 2 resume o método GOR-ELM.

2.1.6 Kernel Extreme Learning Machine

De acordo com Huang et al. (2012), se o mapeamento aleatório das características $h(\boldsymbol{x})$ não é desejado, é possível aplicar as condições de Mercer ao ELM. Para isso, uma

Algoritmo 2: GOR-ELM: *Generalized Outlier Robust Extreme Learning Machine*

Requer: Amostras de treino ($\boldsymbol{X}$,$\boldsymbol{T}$), função de ativação $h(\cdot)$, número inicial de nós na camada oculta $\tilde{N}$ e parâmetros de regularização λ, α, τ e ρ.
Inicialização: $\boldsymbol{B}^0$, $\boldsymbol{U}^0$, $\boldsymbol{Z}^0$, $\boldsymbol{E}^0$ e $k = 0$.
Gera pesos $\boldsymbol{W}$ e vieses $\boldsymbol{\nu}$ aleatoriamente.
Calcula a matriz da camada oculta $\boldsymbol{H}$ usando Equação 2.2.
repita
  $\mathbf{B}^{k+1} := \arg\min_{\mathbf{B}} \frac{\rho}{2} \parallel \mathbf{HB}-\mathbf{T}-\mathbf{E}^k+\mathbf{U}_1^k \parallel_F^2 + \frac{\lambda(1-\alpha)}{2} \parallel \mathbf{B} \parallel_F^2 + \frac{\rho}{2} \parallel \mathbf{B}-\mathbf{Z}^k+\mathbf{U}_2^k \parallel_F^2$
  $\mathbf{Z}^{k+1} := \arg\min_{\mathbf{Z}} \frac{\lambda\alpha}{\rho} \parallel \mathbf{Z} \parallel_{2,1} + \frac{1}{2} \parallel \mathbf{B}^{k+1} - \mathbf{Z} + \mathbf{U}_2^k \parallel_F^2$
  $\mathbf{E}^{k+1} := \arg\min_{\mathbf{E}} \frac{\tau}{\rho} \parallel \mathbf{E} \parallel_{2,1} + \frac{1}{2} \parallel \mathbf{HB}^{k+1} - \mathbf{T} - \mathbf{E} + \mathbf{U}_1^k \parallel$
  $\mathbf{U}_1^{k+1} := \mathbf{U}_1^k + \rho\left(\mathbf{HB}^{k+1} - \mathbf{T} - \mathbf{E}^{k+1}\right)$
  $\mathbf{U}_2^{k+1} := \mathbf{U}_2^k + \rho\left(\mathbf{B}^{k+1} - \mathbf{Z}^{k+1}\right)$
  k = k+1
até *Atingir critério de parada*;

matriz $\boldsymbol{\Omega}_{ELM} = \boldsymbol{H}^\top\boldsymbol{H}$ pode ser utilizada, onde cada elemento $\Omega_{ELM_{i,j}}$ dessa matriz pode ser definido como mostra a Equação 2.37, onde $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ é uma função simétrica.

$$\Omega_{ELM_{i,j}} = h(\boldsymbol{x}_i) \cdot h(\boldsymbol{x}_j) = K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.37}$$

Assim, é possível reescrever a solução do R-ELM, dada pela Equação 2.8 como mostra a Equação 2.38, que é chamada como a solução do Kernel ELM (K-ELM).

$$\hat{\boldsymbol{\beta}}_{\text{K-ELM}} = \left(\boldsymbol{H}^\top\boldsymbol{H} + \frac{\boldsymbol{I}}{C}\right)^{-1} \boldsymbol{H}^\top\boldsymbol{t} = \left(\boldsymbol{\Omega}_{ELM} + \frac{\boldsymbol{I}}{C}\right)^{-1} \boldsymbol{t} \tag{2.38}$$

A saída de uma K-ELM, com respeito a uma entrada $\boldsymbol{x}$ pode ser escrita usando notação matricial como mostra a Equação 2.39.

$$\hat{t}_j = \begin{bmatrix} K(\boldsymbol{x}, \boldsymbol{x}_1) \\ \vdots \\ K(\boldsymbol{x}, \boldsymbol{x}_n) \end{bmatrix}^\top \hat{\boldsymbol{\beta}}_{\text{K-ELM}}. \tag{2.39}$$

No método K-ELM, o mapeamento aleatório e sua dimensionalidade não precisam ser informados. Isto é, não é necessário definir uma função de ativação e nem gerar pesos pseudoaleatórios através de uma semente. O parâmetro, nesse caso, é o tipo do *kernel* e seus parâmetros, onde um exemplo seria o *kernel* Gaussiano $K(\boldsymbol{u}, \boldsymbol{v}) = \exp(-\gamma \parallel \boldsymbol{u}-\boldsymbol{v} \parallel^2)$, onde $\gamma > 0$ é um parâmetro do *kernel*.

Embora tenha a vantagem de não ter uma natureza aleatória, o método K-ELM foi definido principalmente para bases de dados pequenas, devido a sua necessidade de inversão de uma matriz quadrada cuja dimensão é igual ao número de amostras da base de dados na etapa de treinamento. Assim, a sua utilização com bases de dados relativamente

grandes fica dificultada, devido à complexidade computacional $\mathcal{O}(N^3)$ do processo de inversão de matrizes (DENG; ONG; ZHENG, 2016).

Para lidar com bases de dados relativamente maiores, Deng, Ong e Zheng (2016) propuseram o algoritmo *Reduced Kernel* ELM (RK-ELM). Neste método, um número L de amostras, as quais foram chamadas de "vetores suporte" pelos autores do método, $\boldsymbol{\Delta} = [\boldsymbol{\delta}_1^\mathsf{T} \cdots \boldsymbol{\delta}_L^\mathsf{T}]^\mathsf{T}$ são escolhidos aleatoriamente do conjunto de treinamento $\boldsymbol{X}$. Note que $L \leq N$, onde N é o número de amostras da base de dados. Tais "vetores suporte" são utilizados juntamente com os dados $\boldsymbol{X}$ para gerar a matriz de *kernel* $\boldsymbol{H}_{\text{RK-ELM}} \in \mathbb{R}^{N \times L}$, como mostra a Equação 2.40, onde K é a função *kernel* escolhida.

$$\boldsymbol{H}_{\text{RK-ELM}_{i,j}} = K(\boldsymbol{x}_i, \boldsymbol{\delta}_j), \tag{2.40}$$

Com a matriz $\boldsymbol{H}_{\text{RK-ELM}}$ encontrada, é possível calcular os pesos de saída da rede RK-ELM, como mostra a Equação 2.41.

$$\hat{\boldsymbol{\beta}}_{\text{RK-ELM}} = \left(\boldsymbol{H}_{\text{RK-ELM}}^\mathsf{T} \boldsymbol{H}_{\text{RK-ELM}} + \frac{\boldsymbol{I}}{C} \right)^{-1} \boldsymbol{H}_{\text{RK-ELM}}^\mathsf{T} \boldsymbol{t}. \tag{2.41}$$

Já a saída da rede RK-ELM $\hat{t}_j$, dada uma amostra de entrada $\boldsymbol{x}$, pode ser calculada por meio da Equação 2.42.

$$\hat{t}_j = \begin{bmatrix} K(\boldsymbol{x}, \delta_1) \\ \vdots \\ K(\boldsymbol{x}, \delta_L) \end{bmatrix}^\mathsf{T} \hat{\boldsymbol{\beta}}_{\text{RK-ELM}}. \tag{2.42}$$

Para provar que a técnica RK-ELM possui capacidade de generalização similar a um ELM ou R-ELM, Deng, Ong e Zheng (2016) apresentaram um teorema que afirma que para qualquer erro desejado ϵ, irá existir um número L de "vetores suporte" que podem ser usados pelo RK-ELM para aprender N amostras com erro máximo de ϵ.

2.1.7 *Multi-layer Extreme Learning Machine*

De acordo com Deng (2014), até o início dos anos 2010, a maioria dos trabalhos relacionados ao aprendizado de máquina e ao reconhecimento de padrões exploravam majoritariamente o uso de arquiteturas "rasas", onde apenas uma camada de transformação não-linear é usada. Um exemplo de arquitetura "rasa" é a SLFN.

Esse tipo de método é capaz de obter bons resultados em problemas e/ou situações simples, onde, por exemplo, todas as características são independentes. Contudo, em tarefas mais difíceis, como em casos de reconhecimento de fala, tarefas relacionadas a sinais sonoros e tarefas que envolvem processamento de imagens, arquiteturas "rasas" podem apresentar dificuldades para lidar com as estruturas e características complexas que podem

ser obtidas a partir dos dados de entrada (DENG, 2014). Em uma SLFN, por exemplo, lidar com tais características complexas pode requerer um grande número de neurônios ocultos, o que por sua vez pode necessitar um grande poder computacional para treinar a SLFN.

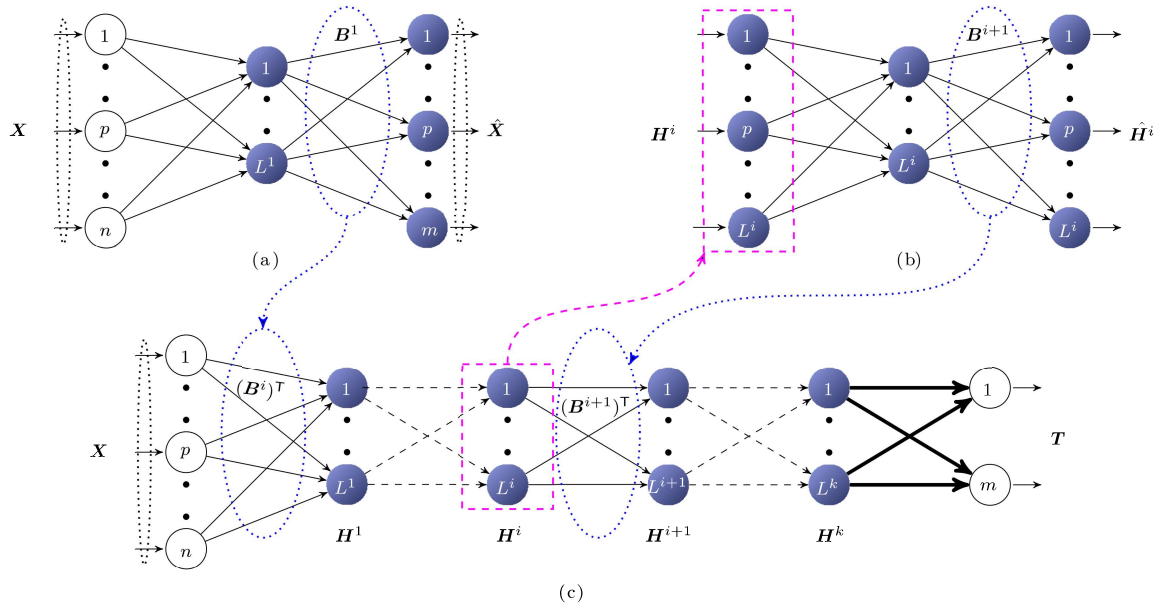
Para lidar com as características complexas, um modelo "profundo" é desejável, onde diversas camadas não lineares são usadas (no escopo de redes neurais artificiais). Assim, o termo "*deep learning*" usualmente se refere a métodos similares à redes neurais com um grande número de camadas ocultas, que também podem ser chamadas de *Deep Neural Networks* (DNN) (DENG, 2014).

Como algoritmos baseados em ELM são definidos para treinar apenas redes do tipo SLFN, não é possível utilizá-los diretamente para treinar redes profundas. No entanto, Kasun et al. (2013) propuseram uma arquitetura com múltiplas camadas ocultas que podem ser treinadas utilizando algoritmos baseados em ELM.

Esta técnica foi chamada de *Multi-Layer Extreme Learning Machine* (ML-ELM), e treina suas camadas intermediárias utilizando uma abordagem "não supervisionada", onde cada uma é treinada de forma isolada utilizando o conceito de *autoencoder*, considerando que as saídas desejadas da rede são iguais à entrada (que vêm da camada anterior do ML-ELM), e posteriormente integrada à rede ML-ELM. Após treinar todas as camadas intermediárias, a camada final é treinada normalmente usando a saída da camada anterior como entrada e os alvos reais da base de dados como saída desejada. Em todas as etapas de treinamento, o algoritmo R-ELM é utilizado. De acordo com Kasun et al. (2013), ML-ELM obteve desempenho similar a arquiteturas profundas treinadas utilizando métodos baseados em *backpropagation*, porém com tempo de treinamento muito inferior.

A Figura 2 mostra um exemplo de arquitetura treinada utilizando o algoritmo ML-ELM. A primeira camada é treinada como se fosse um autoencoder, em que os dados X são usados tanto como entrada como alvo de um algoritmo de treinamento, como ELM ou UWR-ELM. A transposta da matriz $\boldsymbol{B}^1$, obtida pelo treinamento deste *autoencoder*, é utilizada como matriz de pesos entre a entrada da rede e a primeira camada oculta do ML-ELM, como mostra a Figura 2c. Em sequência, a segunda camada também é treinada utilizando um *autoencoder*. Neste caso, a entrada do *autoencoder* é a saída da primeira camada oculta (dada pela multiplicação da entrada - "camada" anterior - pela respectiva matriz de pesos). O segundo *autoencoder* é treinado com a saída desejada igual à entrada, e uma matriz $\boldsymbol{B}^2$ é obtida. Da mesma forma que no primeiro *autoencoder*, a transposta de $\boldsymbol{B}^2$ é utilizada como matriz de pesos entre a segunda camada oculta e a camada anterior. O processo é repetido até que todas as camadas ocultas sejam treinadas, e o vetor final de pesos é calculado utilizando a matriz de saída da última camada oculta como entrada, os alvos $\boldsymbol{T}$ como saída e o algoritmo baseado em ELM.

Figura 2 – Um exemplo da arquitetura treinada utilizando ML-ELM. Em (a), é possível ver o primeiro *autoencoder* treinado pelo algoritmo. Em (b), um *autoencoder* intermediário, e em (c), uma rede ML-ELM completa. Note que é possível ver as relações entre os pesos da rede completa e dos *autoencoders* intermediários.



Fonte: Adaptado de (KASUN et al., 2013)

2.1.8 *Stacked Extreme Learning Machine*

Embora o algoritmo ELM seja capaz de obter uma boa capacidade de generalização com pouco tempo de treinamento, ele pode sofrer com limitações de memória ao se utilizar um grande número de neurônios na camada oculta e uma base de dados com muitas amostras (ZHOU et al., 2014).

Um ponto notado por Zhou et al. (2014) foi que, ao usar a base de dados MNIST, havia espaço para uma melhora do desempenho do modelo, obtida ao aumentar o número de neurônios na camada oculta. Porém, o *hardware* disponível não possuía memória suficiente para lidar com esse treinamento.

Um conceito existente na literatura é o chamado de "modelos empilhados" (WOLPERT, 1992), que diz ser possível melhorar o desempenho de modelos simples ao "empilhar" outros sequencialmente, com a finalidade de aprender características mais complexas dos dados de entrada. Assim, Zhou et al. (2014) propuseram o algoritmo *Stacked Extreme Learning Machine* (S-ELM), que usa projeções da camada oculta, utilizando *Principal Component Analysis* (PCA), de um módulo como parte do próximo.

O primeiro módulo de uma arquitetura S-ELM é treinada utilizando o algoritmo UWR-ELM. Assim, a matriz $\boldsymbol{B}_1 = [\boldsymbol{\beta}_1 \cdots \boldsymbol{\beta}_m]$ é encontrada. O algoritmo usa PCA em $\boldsymbol{B}_1^\intercal$ para determinar os autovalores e autovetores da matriz de covariância.

Com a finalidade de reduzir a dimensão dos dados, os $L^{'}$ ($L^{'} \leq L$) autovalores mais significativos são encontrados, e seus autovetores correspondentes são usados para construir uma matriz $\tilde{\boldsymbol{V}}_1$, que é utilizada para reduzir a dimensão da matriz que representa a saída da camada oculta do primeiro módulo, como mostra a Equação 2.43.

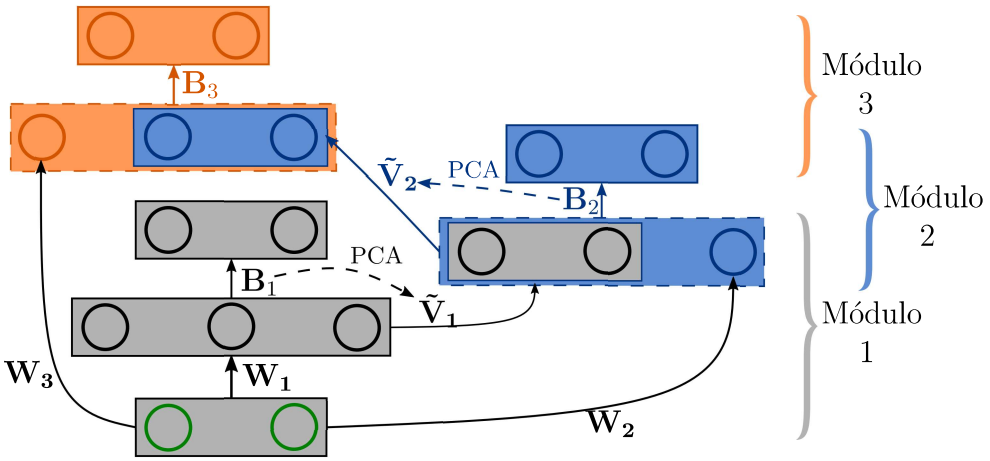
$$\boldsymbol{H}_1^{'} = \boldsymbol{H}_1 \tilde{\boldsymbol{V}} \tag{2.43}$$

A matriz projetada de um módulo é utilizada como parte da camada oculta do próximo, como mostra a Equação 2.44,

$$\boldsymbol{H}_i = [\boldsymbol{H}_{i-1}^{'}, \boldsymbol{H}_{new}] \tag{2.44}$$

onde $\boldsymbol{H}_{new}$ é uma matriz que representa a camada oculta de novos neurônios com pesos aleatórios. Esse procedimento é feito para cada um dos S módulos (exceto o último, pois ele deve retornar uma saída estimada). Um exemplo de uma arquitetura S-ELM é apresentada na Figura 3, onde módulos são representados por uma mesma cor. O primeiro módulo, na cor cinza, é treinado usando UWR-ELM. O PCA é aplicado em $\boldsymbol{B}_1^{\mathrm{T}}$, de dimensão $m \times L$, resultando na matriz reduzida de projeção $\tilde{\boldsymbol{V}}_1$. O segundo módulo, em azul, tem sua camada oculta construída através da concatenação da multiplicação matricial (projeção) $\boldsymbol{H}_1\tilde{\boldsymbol{V}}_1$ e de neurônios aleatórios novos, e é treinado usando o UWR-ELM. O processo continua até que todos os módulos sejam treinados, onde o processo de PCA não é necessário no último.

Figura 3 – Um exemplo da arquitetura S-ELM. Retângulos da mesma cor representam camadas de um mesmo módulo, círculos verdes representam os dados originais, usados como entrada em todos os módulos. Cada seta representa uma multiplicação matricial, enquanto uma linha tracejada representa a redução de dimensionalidade obtida através de PCA.



Fonte: (SILVA; INABA; CIARELLI, 2018)

Zhou et al. (2014) observaram que a técnica S-ELM foi capaz de obter bons resultados quando comparados a outros métodos, profundos ou não, sendo possível treiná-

lo e utilizá-lo em sistemas com menos recursos, devido a não necessidade de inversão de uma matriz de altas dimensões.

Cabe destacar que o método S-ELM aplica o algoritmo PCA numa matriz $\boldsymbol{B}_i^{\mathsf{T}} \in \mathbb{R}^{m \times L}$, onde é comum que $m < L$, para obter uma matriz de projeção. Segundo Turk e Pentland (1991), aplicar PCA numa matriz não quadrada implica que serão obtidos um número de autovalores significativos igual a menor dimensão da matriz subtraída de um. Assim, no máximo $m - 1$ autovalores significativos serão obtidos. Ou seja, a matriz de projeção $\tilde{\boldsymbol{V}}_1$ terá no máximo $m - 1$ autovetores significativos. O S-ELM usa essa matriz para projetar informação de um módulo no seguinte, o que significa que no máximo $m - 1$ neurônios projetados serão significativos. Caso $m = 1$, nenhum neurônio projetado será significativo.

2.1.9 *Deep Stacked Network*

Trabalhos que utilizam DNN foram capazes de obter bons resultados em diversas tarefas de origens distintas. Porém, o treinamento destas redes é computacionalmente muito custoso, principalmente se estes forem feitos na presença de bases de dados com grande número de amostras, o que pode requerer a utilização de computadores com alta capacidade de processamento ou placas gráficas (ZHOU et al., 2014). Isto ocorre, pois existe um grande número de parâmetros que devem ser encontrados no processo de treinamento das redes. Este inconveniente é denominado Problema de Escalabilidade (do inglês, *scalability problem*) (DENG; YU, 2011).

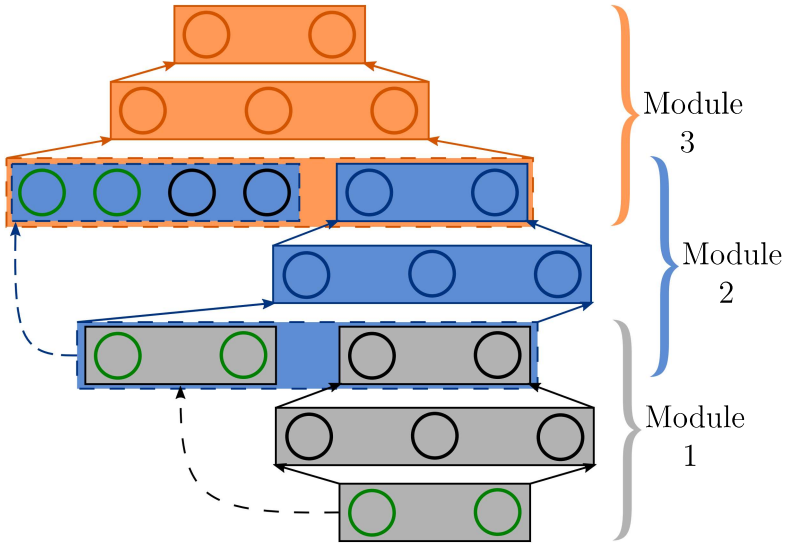
Assim, Deng e Yu (2011) propuseram uma nova arquitetura de aprendizado profundo que utiliza o conceito de modelos empilhados, denominada *Deep Stacked Network* (DSN). Esta arquitetura tem a característica de que a entrada de um módulo é composta pela concatenação da entrada e da saída do módulo anterior. Isto foi motivado pelos resultados obtidos ao trabalhar com os métodos *deep-structured conditional random field* (Yu; Wang; Deng, 2010), que trabalham de forma similar.

Considerando um conjunto de treinamento formado por $(\boldsymbol{X}, \boldsymbol{T})$ com N amostras, o algoritmo de treinamento da DSN pode ser resumido da seguinte maneira: o primeiro módulo (inferior) é treinado normalmente com qualquer técnica de aprendizado de máquina, considerando $\boldsymbol{X}_1 = \boldsymbol{X}$ e $\boldsymbol{T}_1 = \boldsymbol{T}$, sendo $\boldsymbol{X}_1$ e $\boldsymbol{T}_1$ entrada e saída desejada do primeiro módulo, e tem uma saída estimada para o conjunto de treino $\hat{\boldsymbol{T}}_1$. Os demais módulos são treinados da mesma forma, porém com entradas que são construídas a partir da concatenação da entrada do módulo anterior com a saída estimada deste, i.e., o segundo módulo é treinado considerando $\boldsymbol{X}_2 = [\boldsymbol{X}_1, \hat{\boldsymbol{T}}_1]$ como entrada, e $\boldsymbol{T}$ como saída. A Equação 2.45 mostra a regra de construção da entrada do k-ésimo módulo de uma DSN.

$$\boldsymbol{X}_k = \begin{cases} \boldsymbol{X}, & k = 1 \\ [\boldsymbol{X}_{k-1}, \hat{\boldsymbol{T}}_{k-1}], & \text{caso contrário} \end{cases} \tag{2.45}$$

Com as entradas "construídas", os módulos de uma DSN podem ser treinados sequencialmente e, ao empilhá-los, é possível obter resultados similares aos obtidos por redes neurais com múltiplas camadas, mas com tempo de treinamento reduzido (DENG; YU, 2011). Um exemplo desta arquitetura é apresentado na Figura 4, que mostra uma DSN com módulos SLFN. O primeiro módulo, na cor cinza, é treinado utilizando, por exemplo, o UWR-ELM. O segundo módulo tem sua entrada construída através da concatenação da entrada e da saída do primeiro módulo, e também é treinado. Este processo continua até que todos os módulos sejam treinados.

Figura 4 – Um exemplo da arquitetura DSN. Retângulos da mesma cor representam camadas de um mesmo módulo, e círculos verdes representam os dados originais de entrada.



Fonte: (SILVA; INABA; CIARELLI, 2018)

2.2 Técnicas de Aprendizado de Máquina Online

Técnicas de aprendizado de máquina lidam com um conjunto de dados, extraindo informações a partir destes com o objetivo de inferir algum conhecimento. Porém, o treinamento usualmente é feito de forma *offline*, onde todas as amostras disponíveis são utilizadas de uma vez para criar um modelo, que não costuma ser alterado ao longo do tempo.

Em alguns casos, o conjunto de dados inicial não é capaz de representar adequadamente as características do problema tratado, o que pode interferir no desempenho da solução utilizada. Entretanto, novas amostras do mesmo problema podem ser obtidas ao longo do tempo, que podem trazer novas informações importantes para o modelo de inferência, melhorando a representatividade dos dados. Assim, novos modelos de aprendi-

zado de máquina foram propostos com algumas características que os tornam capazes de adaptar a novos dados recebidos (MINKU, 2011; POLIKAR et al., 2001):

- Capacidade de aprender informação adicional de novos dados;
- O aprendizado de dados novos não requer armazenamento dos dados antigos, podendo apenas guardar informações sobre eles (como a média);
- Deve preservar o conhecimento adquirido.

Estes sistemas podem aprender novas informações a cada amostra recebida ou quando um grupo de novas amostras é completado. Os sistemas do primeiro tipo são considerados como sistemas de aprendizado online, enquanto os demais são considerados como aprendizado sequencial[2] (MINKU, 2011).

Um efeito chamado de *concept drift* (LU et al., 2018) pode ocorrer em dados que são obtidos ao longo do tempo, onde ocorre uma variação das propriedades estatísticas das características que representam as amostras. Um exemplo disto seria uma base de dados que foi construída durante um longo período de tempo, onde as características intrínsecas de um sensor variam com o tempo devido a sua degradação e/ou envelhecimento. Assim, dados de uma classe podem ter valores com uma distribuição no início da base, mas com outra no final.

O *concept drift* pode ser classificado de acordo com a sua velocidade, recorrência, escopo, e outras características (SOARES; ARAÚJO, 2015a). O *drift* abrupto consiste numa mudança rápida (brusca) nos valores das características, enquanto o *drift* gradual ocorre lentamente. O *drift* recorrente é caracterizado por eventos cíclicos, como estações do ano, que se repetem com uma certa recorrência. *Drifts* globais podem se referir a todas as características e/ou medições tendo suas distribuições estatísticas se alterando em conjunto, enquanto *drifts* locais ocorrem apenas em uma característica e/ou medição isolada.

A mudança nas distribuições estatísticas das amostras usualmente afeta negativamente o desempenho de métodos de aprendizado de máquina, principalmente os *offline*. Assim, algumas técnicas de detecção de *drift* foram desenvolvidas, como por exemplo o *Drift Detection Method* (DDM), que avisa ao algoritmo que ocorreu *drift* nos dados. Ao detectar um *drift*, uma espécie de "sinal" é enviado para que o modelo se adapte ao novo conceito. Este tipo de abordagem é chamado de detecção explícita de *drift*. Alguns algoritmos podem aplicar uma abordagem implícita para lidar com a alteração nos conceitos, e estes usualmente aprendem constantemente com as amostras, ajustando seus parâmetros ao longo do tempo (SOARES; ARAÚJO, 2015a).

[2] Minku (2011) define tal classe de técnicas como "técnicas de aprendizado incremental". Porém, no contexto de ELM, algoritmos com a capacidade de se adaptar a novos dados são chamados de *online* ou "sequenciais".

Entre os métodos que lidam implicitamente com *drift*, é possível destacar os que utilizam *ensembles* de modelos de aprendizado de máquina. De acordo com Krawczyk et al. (2017), métodos deste tipo são os mais "promissores" entre os que lidam com dados de forma *online*. O ensemble consiste em métodos distintos que cujas estimativas em relação a uma amostra são combinadas para encontrar a saída geral do ensemble. Como exemplo para problemas de classificação, é possível contar qual a classe que apareceu mais vezes nas predições dos modelos, enquanto que para problemas de regressão, é possível usar uma média ponderada das saídas de todos os membros, onde os pesos da média ponderada podem ser atualizados a cada lote ou amostra, baseado no erro de cada um dos membros do ensemble. Exemplos de métodos que utilizam uma abordagem similar à descrita são Learn^{++}.NSE (ELWELL; POLIKAR, 2011), *Online Weighted Ensemble* (OWE) (SOARES; ARAÚJO, 2015b), *Dynamic Online Ensemble Regression* (DOER) (SOARES; ARAÚJO, 2015a) e *Online Ensemble using Ordered Aggregation* (OEOA) (SOARES; ARAÚJO, 2016).

2.2.1 *Online Sequential Extreme Learning Machine*

Liang et al. (2006) propuseram uma forma fechada, baseada em ELM, de lidar com dados obtidos de forma sequencial (novas amostras são obtidas ao longo do tempo), de modo que não seja necessário ter todos os dados armazenados para que o modelo atenda às características de modelos sequenciais. A esta técnica foi dado o nome de *Online Sequential Extreme Learning Machine* (OS-ELM).

Considerando que inicialmente o modelo ELM foi treinado com N_0 amostras, i.e., uma matriz $\boldsymbol{\beta}_0$ já foi calculada, e N_1 novas observações foram obtidas, pode-se reescrever a Equação 2.4 na forma da Equação 2.46,

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left\| \begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{H}_1 \end{bmatrix} \boldsymbol{\beta} - \begin{bmatrix} \boldsymbol{t}_0 \\ \boldsymbol{t}_1 \end{bmatrix} \right\|_2^2, \tag{2.46}$$

onde $\boldsymbol{H}_0$ e $\boldsymbol{H}_1$ são as matrizes que contêm a saída da camada oculta da SLFN com relação as amostras do primeiro e segundo lote de treinamento, e $\boldsymbol{t}_0$ e $\boldsymbol{t}_1$ as respectivas saídas esperadas.

Também pode-se reescrever a Equação 2.5 como mostra a Equação 2.47,

$$\hat{\boldsymbol{\beta}}_1 = \boldsymbol{K}_1^{-1} \begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{H}_1 \end{bmatrix}^\mathsf{T} \begin{bmatrix} \boldsymbol{t}_0 \\ \boldsymbol{t}_1 \end{bmatrix} \tag{2.47}$$

onde a matriz $\boldsymbol{K}_1$ é dada pela Equação 2.48,

$$\boldsymbol{K}_1 = \begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{H}_1 \end{bmatrix}^\mathsf{T} \begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{H}_1 \end{bmatrix} = \boldsymbol{K}_0 + \boldsymbol{H}_1^\mathsf{T} \boldsymbol{H}_1 \tag{2.48}$$

e $\boldsymbol{H}_1$ representa a matriz de saída da camada oculta para as novas N_1 amostras, e $\boldsymbol{t}_1$ os seus respectivos alvos.

Devido às restrições de que o modelo não deve armazenar dados antigos para atualizar seus parâmetros, é desejável que $\boldsymbol{\beta}_1$ seja função apenas de $\boldsymbol{K}_1$, $\boldsymbol{H}_1$ e $\boldsymbol{t}_1$ (LIANG et al., 2006).

Logo, a Equação 2.47 pode ser reescrita como mostra a Equação 2.49,

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_1 &= \boldsymbol{K}_1^{-1}\left(\boldsymbol{H}_0^\top \boldsymbol{t}_0 + \boldsymbol{H}_1^\top \boldsymbol{t}_1\right) \\
&= \boldsymbol{K}_1^{-1}\left(\boldsymbol{K}_0\boldsymbol{K}_0^{-1}\boldsymbol{H}_0^\top \boldsymbol{t}_0 + \boldsymbol{H}_1^\top \boldsymbol{t}_1\right) \\
&= \boldsymbol{K}_1^{-1}\left(\boldsymbol{K}_0\boldsymbol{\beta}_0 + \boldsymbol{H}_1^\top \boldsymbol{t}_1\right) \\
&= \boldsymbol{K}_1^{-1}\left(\left(\boldsymbol{K}_1 - \boldsymbol{H}_1^\top\boldsymbol{H}_1\right)\boldsymbol{\beta}_0 + \boldsymbol{H}_1^\top \boldsymbol{t}_1\right) \\
&= \boldsymbol{K}_1^{-1}\left(\boldsymbol{K}_1\boldsymbol{\beta}^{(0)} - \boldsymbol{H}_1^\top\boldsymbol{H}_1\boldsymbol{\beta}_0 + \boldsymbol{H}_1^\top \boldsymbol{t}_1\right) \\
&= \boldsymbol{\beta}_0 + \boldsymbol{K}_1^{-1}\boldsymbol{H}_1^\top\left(\boldsymbol{t}_1 - \boldsymbol{H}_1\boldsymbol{\beta}_0\right),
\end{aligned} \tag{2.49}$$

onde $\boldsymbol{K}_1$ é dado pela Equação 2.48. As Equações 2.48 e 2.49 podem ser estendidas para índices posteriores, onde cada um é referente a um novo lote de dados recebido, como pode ser visto nas Equações 2.50 e 2.51.

$$\boldsymbol{K}_{k+1} = \boldsymbol{K}_k + \boldsymbol{H}_{k+1}^\top\boldsymbol{H}_{k+1} \tag{2.50}$$

$$\hat{\boldsymbol{\beta}}_{k+1} = \boldsymbol{\beta}_k + \boldsymbol{K}_{k+1}^{-1}\boldsymbol{H}_{k+1}^\top\left(\boldsymbol{t}_{k+1} - \boldsymbol{H}_{k+1}\boldsymbol{\beta}_k\right) \tag{2.51}$$

Pode-se ainda otimizar a atualização deste modelo se for considerado que $\boldsymbol{P}_{k+1} = \boldsymbol{K}_{k+1}^{-1}$ e a equivalência de Woodbury (PETERSEN; PEDERSEN, 2007) for utilizada para atualizar $\boldsymbol{P}_{k+1}$ a partir de $\boldsymbol{P}_k$, como mostra a Equação 2.52.

$$\begin{aligned}
\boldsymbol{P}_{k+1} &= \boldsymbol{K}_{k+1}^{-1} \\
\boldsymbol{P}_{k+1} &= \left(\boldsymbol{K}_k + \boldsymbol{H}_{k+1}^\top\boldsymbol{H}_{k+1}\right)^{-1} \\
\boldsymbol{P}_{k+1} &= \boldsymbol{K}_k^{-1} - \boldsymbol{K}_k^{-1}\boldsymbol{H}_{k+1}^\top\left(\boldsymbol{I} + \boldsymbol{H}_{k+1}\boldsymbol{K}_k^{-1}\boldsymbol{H}_{k+1}^\top\right)^{-1}\boldsymbol{H}_{k+1}\boldsymbol{K}_k^{-1}
\end{aligned} \tag{2.52}$$

Assim, as regras de atualização do vetor de saída $\boldsymbol{\beta}_{k+1}$, referente às novas N_k amostras $(\boldsymbol{X}^{k+1}, \boldsymbol{t}^{k+1})$, do algoritmo OS-ELM são resumidas nas Equações 2.53 e 2.54.

$$\boldsymbol{P}_{k+1} = \boldsymbol{P}_k - \boldsymbol{P}_k\boldsymbol{H}_{k+1}^\top\left(\boldsymbol{I} + \boldsymbol{H}_{k+1}\boldsymbol{P}_k\boldsymbol{H}_{k+1}^\top\right)^{-1}\boldsymbol{H}_{k+1}\boldsymbol{P}_k \tag{2.53}$$

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \boldsymbol{P}_{k+1}\boldsymbol{H}_{k+1}^\top\left(\boldsymbol{t}_{k+1} - \boldsymbol{H}_{k+1}\boldsymbol{\beta}_k\right) \tag{2.54}$$

Da mesma forma que o ELM, o algoritmo OS-ELM também é definido para apenas uma saída. Porém, da mesma forma que foi mostrado na Seção 2.1.7, é possível construir um modelo para cada saída, e caso esses modelos compartilhem o mesmo conjunto de pesos entre as camadas de entrada e oculta, é possível substituir o vetor $\boldsymbol{t}$ por uma matriz $\boldsymbol{T}$ na Equação 2.54.

De acordo com Liang et al. (2006), OS-ELM e ELM atingem o mesmo desempenho (erro de treinamento e acurácia) quando o $\text{rank}(\boldsymbol{H}_0) = L$. Assim, é necessário um conjunto

de amostras iniciais de tamanho maior ou igual ao número de neurônios da SLFN que está sendo treinada para que o OS-ELM atinja os mesmos resultados de um ELM treinado de forma *offline* com todo o conjunto de amostras observadas.

2.2.2 *Online Sequential Regularized Extreme Learning Machine*

Assim como ELM possui problemas com sobreajuste e subajuste, caso o número de neurônios de sua camada oculta não seja bem definido, o algoritmo OS-ELM sofre do mesmo problema. Além deste, ainda existe a limitação de que o número inicial de amostras deve ser maior ou igual ao número de neurônios da rede (para que atinja o mesmo desempenho do ELM), o que pode limitar o tamanho da arquitetura que pode ser utilizada, ou então causar uma espera pelo número necessário de amostras.

Como foi visto na Seção 2.1.2, é possível resolver o problema de grande sensibilidade ao número de neurônios utilizando uma regularização Tikhonov, cujo uso no contexto de ELM resultou na técnica UWR-ELM, cuja solução é dada pela Equação 2.8. Assim, Shao e Er (2016) mostraram que é possível, de forma similar à feita na técnica OS-ELM (LIANG et al., 2006), atualizar a matriz de pesos $\boldsymbol{\beta}_k$ através da identidade de Woodbury (PETERSEN; PEDERSEN, 2007), apenas substituindo o cálculo inicial de $\boldsymbol{\beta}_0$ pela solução da Equação 2.8, que considera a regularização Tikhonov. Porém, o algoritmo ainda precisaria de um número mínimo de amostras para o seu treino inicial.

Para lidar com este problema, Shao e Er (2016) propuseram uma abordagem para tornar possível um treinamento inicial do OSR-ELM com menos amostras do que número de neurônios. Tal abordagem considera que o cálculo inicial da matriz $\boldsymbol{K}_0^{-1}$ é feito pela Equação 2.55,

$$\boldsymbol{K}_0 = \boldsymbol{H}_0\boldsymbol{H}_0^\mathsf{T} + \frac{\boldsymbol{I}}{C}, \tag{2.55}$$

E o cálculo da matriz $\boldsymbol{\beta}_0$ inicial é feito pela Equação 2.56,

$$\boldsymbol{\beta}_0 = \boldsymbol{H}_0^\mathsf{T}\boldsymbol{K}_0^{-1}\boldsymbol{t}_0. \tag{2.56}$$

É possível notar que o conjunto de Equações 2.55 e 2.56 também é solução do problema de otimização apresentado na Equação 2.6, quando $\boldsymbol{D} = \boldsymbol{I}$, porém, com a vantagem de um custo computacional inferior, quando $N < L$.

Quando um conjunto novo de amostras é recebido, a matriz $\boldsymbol{K}_1$ da Equação 2.48 pode ser reescrita, considerando o termo adicional da regularização, como mostra a Equação 2.57,

$$\boldsymbol{K}_1^{-1} = \begin{bmatrix}\boldsymbol{H}_0\\ \boldsymbol{H}_1\end{bmatrix}\begin{bmatrix}\boldsymbol{H}_0\\ \boldsymbol{H}_1\end{bmatrix}^\mathsf{T} + \frac{\boldsymbol{I}}{C} = \begin{bmatrix}\boldsymbol{H}_0\boldsymbol{H}_0^\mathsf{T} + \frac{\boldsymbol{I}}{C} & \boldsymbol{H}_0\boldsymbol{H}_1^\mathsf{T}\\ \boldsymbol{H}_1\boldsymbol{H}_0^\mathsf{T} & \boldsymbol{H}_1\boldsymbol{H}_1^\mathsf{T} + \frac{\boldsymbol{I}}{C}\end{bmatrix}^{-1}. \tag{2.57}$$

Usando o complemento de Schur (PETERSEN; PEDERSEN, 2007) da matriz $\boldsymbol{K}_1^{-1}$, pode-se escrevê-la da forma apresentada na Equação 2.58,

$$\begin{aligned}\mathbf{K}_1^{-1} &= \begin{bmatrix} \boldsymbol{H}_0\boldsymbol{H}_0^T + \frac{\boldsymbol{I}}{C} & \boldsymbol{H}_0\boldsymbol{H}_1^T \\ \boldsymbol{H}_1\boldsymbol{H}_0^T & \boldsymbol{H}_1\boldsymbol{H}_1^T + \frac{\boldsymbol{I}}{C} \end{bmatrix}^{-1} \\ &= \begin{bmatrix} \mathbf{K}_0^{-1} + \boldsymbol{K}_0^{-1}\boldsymbol{H}_0\boldsymbol{H}_1^T\boldsymbol{S}_1^{-1}\boldsymbol{H}_1\boldsymbol{H}_0^T\boldsymbol{K}_0^{-1} & -\boldsymbol{K}_0^{-1}\boldsymbol{H}_0\boldsymbol{H}_1^T\boldsymbol{S}_1^{-1} \\ -\boldsymbol{S}_1^{-1}\boldsymbol{H}_1\boldsymbol{H}_0^T\boldsymbol{K}_0^{-1} & \boldsymbol{S}_1^{-1} \end{bmatrix}, \end{aligned} \tag{2.58}$$

onde $\boldsymbol{S}_1 = \boldsymbol{H}_1\boldsymbol{H}_1^T + \frac{\boldsymbol{I}}{C} - \boldsymbol{H}_1\boldsymbol{H}_0^T\boldsymbol{K}_0^{-1}\boldsymbol{H}_0\boldsymbol{H}_1^T$ é o complemento de Schur de $\boldsymbol{K}_0^{-1}$. Após o cálculo de $\boldsymbol{K}_1^{-1}$ a partir de $\boldsymbol{K}_0^{-1}$, o valor atualizado do vetor de pesos de saída $\boldsymbol{\beta}_1$ é dado pela Equação 2.59.

$$\boldsymbol{\beta}_1 = \begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{H}_1 \end{bmatrix}^{\mathsf{T}} \boldsymbol{K}_1^{-1} \begin{bmatrix} \boldsymbol{t}_0 \\ \boldsymbol{t}_1 \end{bmatrix}. \tag{2.59}$$

O procedimento apresentado pode ser estendido para os próximos conjuntos de dados, substituindo os índices das matrizes $\boldsymbol{K}$, $\boldsymbol{H}$ e do vetor $\boldsymbol{t}$. É possível notar que as Equações 2.58 e 2.59 necessitam do armazenamento de dados antigos para realizar suas atualizações ($\boldsymbol{H}_0$ e $\boldsymbol{H}_1$ são necessários para calcular $\boldsymbol{K}_1^{-1}$), o que pode causar um problema de memória, dependendo do número de amostras. Porém, o algoritmo OSR-ELM só utiliza o procedimento que necessita esse armazenamento enquanto ele não recebe um total de amostras maior que o número de neurônios da camada oculta, o que significa que armazenar tais matrizes para poucas amostras não é um problema.

O Algoritmo 3 resume o método OSR-ELM.

2.2.3 Learn^{++}.NSE

Alguns outros métodos que lidam com dados sequenciais foram propostos na literatura. Um deles é chamado de Learn^{++} e foi originalmente proposto por Polikar et al. (2001) como um método sequencial cujo objetivo é lidar com bases de dados estacionárias, isto é, dados considerados sem *concept drift*. Assim como os métodos OS-ELM e OSR-ELM, o algoritmo Learn^{++} é capaz de atualizar seus parâmetros com a chegada de novos lotes de dados, com a finalidade de incluir o conhecimento que novos lotes trazem.

O Learn^{++} é baseado no método AdaBoost, sendo composto de um *ensemble* de classificadores (ou regressores) simples. Cada membro do *ensemble* é treinado com amostras ponderadas de forma diferente, com base no erro de cada uma, e contribui para a saída da técnica Learn^{++} com um voto. Assim, a saída deste método com relação a uma amostra é obtida através de uma votação simples majoritária, onde a classe mais votada pelos membros é escolhida (ou algum tipo de média, no caso de regressão). Cada membro do *ensemble* é treinado com um dos lotes de dados que vão chegando ao longo do tempo. O Learn^{++} deu origem a uma família de métodos sequenciais especializados, como

Algoritmo 3: OSR-ELM: *Online Sequential Regularized Extreme Learning Machine*

Requer: Amostras iniciais de treino ($\boldsymbol{X}_0$,$\boldsymbol{t}_0$), função de ativação $h(\cdot)$, número de nós iniciais na camada oculta $\tilde{N}$ e parâmetro de regularização C.
Inicialização: Gera pesos $\mathbf{A}$ e vieses $\boldsymbol{\nu}$ aleatoriamente.
Calcula a matriz da camada oculta $\boldsymbol{H}_0$ usando Equação 2.2 e $\boldsymbol{X}_0$.
Encontra os pesos de saída $\boldsymbol{\beta}_0$ usando a Equação 2.8 e $\boldsymbol{H}_0$.
Faça $k = 0$ e armazene $\boldsymbol{K}_k^{-1} = \boldsymbol{K}_0^{-1} = \left(\boldsymbol{H}_0\boldsymbol{H}_0^\intercal + \frac{I}{C}\right)^{-1}$, $\boldsymbol{H}_k = \boldsymbol{H}_0$ e $\boldsymbol{t}_k = \boldsymbol{t}_0$.
repita
    Aguarda o $(k+1)$-ésimo lote de amostras ($\boldsymbol{X}_{k+1}$,$\boldsymbol{t}_{k+1}$) chegar
    se *total de amostras recebidos for menor que* L **então**
        Encontre $\boldsymbol{K}_{k+1}^{-1}$ a partir de $\boldsymbol{K}_k^{-1}$, $\boldsymbol{H}_k$ e $\boldsymbol{H}_{k+1}$, usando a Equação 2.58.
        Encontre $\boldsymbol{\beta}_{k+1}$ usando a Equação 2.59
        Faça $\boldsymbol{H}_{k+1} = [\boldsymbol{H}_k^\intercal\ \boldsymbol{H}_{k+1}^\intercal]^\intercal$ e $\boldsymbol{t}_{k+1} = [\boldsymbol{t}_k^\intercal\ \boldsymbol{t}_{k+1}^\intercal]^\intercal$.
    senão se *O lote atual torna o número total de amostras maior que* L **então**
        Faça $\boldsymbol{H}_{k+1} = [\boldsymbol{H}_k^\intercal\ \boldsymbol{H}_{k+1}^\intercal]^\intercal$ e $\boldsymbol{t}_{k+1} = [\boldsymbol{t}_k^\intercal\ \boldsymbol{t}_{k+1}^\intercal]^\intercal$.
        Armazene $\boldsymbol{P}_{k+1} = \left(\boldsymbol{H}_{k+1}^\intercal\boldsymbol{H}_{k+1} + \frac{I}{C}\right)^{-1}$.
        $\boldsymbol{\beta}_{k+1} = \boldsymbol{P}_{k+1}\boldsymbol{H}_{k+1}^\intercal\boldsymbol{t}_{k+1}$.
    senão
        Atualize $\boldsymbol{P}_{k+1}$ de acordo com a Equação 2.53.
        Atuailze $\boldsymbol{\beta}_{k+1}$ de acordo com a Equação 2.54.
    $k = k + 1$
até *que não tenham mais dados chegando*;

o Learn^{++}.NC, especializado para lidar com novas classes e o método Learn^{++}.MF, que é especializado em aprender dados com características faltantes, entre outros (ELWELL; POLIKAR, 2011).

Uma das extensões do Learn^{++} é o método chamado Learn^{++} *for non-stationary environments* (ou Learn^{++}.NSE), que foi desenvolvido para lidar com dados sequenciais com a ocorrência de *concept drift*. De acordo com Elwell e Polikar (2011), este método é capaz de lidar com vários cenários distintos de *concept drift*, como os rápidos, graduais, abruptos, recorrentes, entre outros, também costuma ter seu desempenho não muito dependente do classificador base. Assim como o Adaboost, este método pondera amostras de forma distinta, a depender do erro de cada uma. Além disso, possui melhor desempenho caso não seja aplicado um método de poda dos membros do ensemble, principalmente no caso de *drifts* recorrentes.

O Learn^{++}.NSE é um método baseado em ensemble proposto para lidar com tarefas de classificação. Originalmente, uma votação majoritária ponderada é utilizada para definir a qual classe uma amostra pertence, e utiliza de um mecanismo passivo de detecção de *drift*. Supondo que o algoritmo irá receber uma série de lotes $\mathcal{D}^t = \{(\boldsymbol{x}_i^t, y_i^t)\}^{N^t}$, $i = 1, \ldots, N^t$, com N^t amostras cada, onde t é um índice temporal (ou seja, $\boldsymbol{x}_i^t$ é a i-ésima instância do

t-ésimo lote), ele terá t classificadores. A saída do j-ésimo classificador referente a entrada $\boldsymbol{x}_i^t$ pode ser representada por $f_k(\boldsymbol{x}_i^t)$. Já a saída do ensemble, ou seja, a saída do modelo Learn^{++}.NSE, chamada de $F^t(\boldsymbol{x}_i^t)$, é uma combinação ponderada de cada $f_k(\boldsymbol{x}_i^t)$, cujos pesos da ponderação são atualizados a cada novo conjunto de dados seguindo uma regra baseada no erro atual, isto é, os classificadores com menor erro nos lotes mais recentes terão maior peso na hora de encontrar a saída final do ensemble.

Um dos problemas desta técnica é o fato de não haver uma limitação na adição de novos classificadores. Esse problema pode ser mitigado ao utilizar modelos simples, que não utilizam tanta memória, mas ainda assim a memória utilizada pode ser um problema ao longo do tempo. Uma outra possível solução seria a limitação do número máximo de classificadores, removendo os mais antigos ou aqueles com maior erro. Porém, segundo Elwell e Polikar (2011), esta estratégia não é recomendada, já que reduz a habilidade do método de lembrar conceitos recorrentes, bem como impacta a estabilidade do método durante períodos sem a presença de *drift*.

Embora tenha sido proposto para tarefas de classificação, uma adaptação do método Learn^{++}.NSE para lidar com problemas de regressão foi proposta em (SOARES; ARAÚJO, 2015a), onde uma abordagem similar ao método AdaBoost.RT é utilizada para "classificar" a regressão das amostras, separando-as em predições corretas e incorretas. Mais informações acerca da técnica podem ser encontradas em (ELWELL; POLIKAR, 2011).

2.2.4 *Online Weighted Ensemble*

O método Learn^{++}.NSE é baseado em lotes, o que significa que ele precisa esperar um determinado número de amostras chegar para que só então ele atualize seus parâmetros. De acordo com Soares e Araújo (2015b), isso é um problema comum a todos os métodos que utilizam dessa abordagem, já que é possível que seja necessário esperar um longo tempo para que haja o número necessário de amostras para o método. Além disso, se o lote demorar muito, é possível que o lote inclua diversos conceitos diferentes, misturando informações importantes.

Uma alternativa a esses métodos baseados em lotes são aqueles baseados em amostras, que atualizam seus parâmetros para cada amostra recebida. Porém, tantas atualizações trazem dois problemas, que são uma possível instabilidade no modelo e o tempo gasto para atualizar o modelo para cada amostra. Para mitigar o primeiro problema, autores adotam o uso de uma "janela móvel" que guarda as amostras mais recentes, e a utiliza para atualizar os parâmetros toda vez que uma amostra chega ao sistema.

Um dos métodos que utilizam essa abordagem é o *Online Weighted Ensemble* (OWE) (SOARES; ARAÚJO, 2015b), que possui características como: adaptação dos pesos dos modelos (o peso da contribuição do modelo na saída do ensemble), e inclusão e remoção de modelos quando o desempenho começa a ser degradado. Além disso, ao contrário do

Learn^{++}.NSE, ele foi proposto para utilização em bases de dados com regressão.

O método OWE armazena k modelos, uma janela móvel com N^t amostras, e alguns parâmetros: o algoritmo base, um limiar θ para considerar uma amostra como "correta" ou "incorreta", um parâmetro $\alpha \in [0, 1]$, que controla a inclusão de modelos novos, um fator $\kappa \in [0, 1]$ e um número $S \in \mathbb{Z} \cup \{\infty\}$, que representa o número máximo de modelos no ensemble (se $S = \infty$, o método não poda nenhum de seus membros).

Assim, para cada amostra recebida, o ensemble ajusta os pesos de seus membros (isto é, a contribuição de cada membro na saída do ensemble) baseado nos erros obtidos na janela móvel atual. O mecanismo de cálculo de erro considera amostras "corretas" e "incorretas" e é baseado no algoritmo AdaBoost.RT, usando erro relativo e o limiar θ. Assim, se a saída para uma amostra possui erro relativo em relação ao valor desejado maior que o limiar, ela é considerada incorreta.

Cada membro do ensemble também possui um parâmetro "tempo de vida" τ_j, que é incrementado toda vez que uma amostra é recebida pelo algoritmo. Este contador é considerado na atualização dos pesos de cada membro, onde os membros mais novos tendem a receber mais peso do que os membros mais antigos, e também é levado em consideração para decidir se o método irá substituir um de seus membros. Mais informações acerca da técnica podem ser encontradas em (SOARES; ARAÚJO, 2015b).

2.2.5 *Dynamic And Online Ensemble Regression*

Uma outra técnica proposta para lidar com bases de dados de regressão com *concept drift* é o *Dynamic And Online Ensemble Regression* (DOER), que possui algumas características similares ao OWE: atualiza seus pesos a cada amostra recebida e poda membros com base no erro e/ou idade. Porém, ao contrário do OWE, o método DOER não utiliza um limiar para classificar amostras em "corretas" ou "incorretas" e considera que os membros têm a capacidade de atualizar os seus parâmetros internos (ou seja, um modelo do tipo OS-ELM poderia ser utilizado).

Da mesma forma que o OWE, o método DOER possui k modelos, uma janela móvel que armazena as N^t amostras mais recentes, e alguns parâmetros: um algoritmo base de treinamento de seus membros, que deve ser *online*, um parâmetro $\alpha \in [0, 1]$, que controla a inclusão de novos modelos e um número máximo de membros do ensemble $S \in \mathbb{Z} \cup \{\infty\}$. Assim, quando uma nova amostra chega, a janela móvel é atualizada e utilizada para atualizar todos os membros do ensemble.

Enquanto o OWE considerava amostras como acertos ou erros baseados em um limiar de distância θ, o DOER utiliza um critério baseado em Erro Quadrático Médio para decidir se membros devem ser podados e substituídos por modelos novos. O erro do j-ésimo membro com respeito a amostra mais recente $(\boldsymbol{x}^t, y^t)$ pode ser calculado como

mostra a Equação 2.60.

$$e_j^t = (y^t - f_j(\boldsymbol{x}^t))^2. \tag{2.60}$$

Da mesma forma que o OWE, cada membro do ensemble possui um contador τ_j que é incrementado toda vez que uma nova amostra chega ao ensemble. Com esse contador, um erro normalizado é calculado para cada membro, considerando a janela móvel armazenada, como mostra a Equação 2.61,

$$\text{MSE}_j^t = \begin{cases} 0, & \text{if } \tau_j = 0 \\ \frac{\tau_j - 1}{\tau_j}\text{MSE}_j^{t-1} + \frac{1}{\tau_j}e_j^t & \text{if } 1 \leq \tau_j \leq N^t \\ \text{MSE}_j^{t-1} + \frac{e_j^t}{N^t} - \frac{e_j^{t-N^t}}{N^t} & \text{if } \tau_j > N^t \end{cases}, \tag{2.61}$$

onde τ_j representa a "idade" de um modelo, ou seja, o número de amostras recebidas desde que o membro foi incluído no *ensemble*. A medida MSE_j^t é utilizada como critério para cálculo dos pesos de cada membro na saída final do ensemble e para decidir se um membro será ou não substituído por um novo. Mais informações acerca da técnica podem ser encontradas em (SOARES; ARAÚJO, 2015a).

2.2.6 *Online Ensemble Using Ordered Aggregation*

Um método similar ao DOER, chamado de *Online Ensemble Using Ordered Aggregation* (OEOA), foi proposto por Soares e Araújo (2016). Enquanto o DOER inicia com um membro e vai adicionando novos membros ao longo de sua utilização (até o número máximo S), OEOA inicia com um número S_{max} de membros. A cada nova amostra, o membro com o maior erro pode ser substituído por um novo membro, que é treinado com a janela móvel atual, caso um limiar α seja atingido. Cada membro do ensemble também deve ser um modelo online/sequencial, que terá um peso e um contador de idade τ_j. Assim como é feito no método DOER, o peso também é calculado com base nos erros e_j^t e MSE_j^t das Equações 2.60 e 2.61, obtidos utilizando a janela móvel mais recente.

Uma outra diferença entre o DOER e o OEOA é que o segundo sempre possui S_{max} membros, e nem sempre todos são utilizados para construir a saída final do ensemble. Em alguns casos, a contribuição de alguns membros é "ignorada" com o objetivo de melhorar o desempenho do sistema. Isso é feito escolhendo um número ótimo S de membros, onde $S_{\text{min}} \leq S \leq S_{\text{max}}$, isto é, é estimado que o menor erro seria obtido utilizando apenas um número S dos membros, ao invés de utilizar todos. É possível notar que se for definido que $S_{\text{min}} = S_{\text{max}}$, todos os membros do ensemble sempre são utilizados para calcular a saída do ensemble. Mais informações acerca da técnica podem ser encontradas em (SOARES; ARAÚJO, 2016).

2.3 Métodos de Inicialização de Pesos de ELM e suas variantes

A teoria que suporta o ELM (HUANG et al., 2012) considera que parte dos pesos de uma SLFN são gerados aleatoriamente a partir de uma distribuição de probabilidade contínua. Embora muitos trabalhos tenham apresentado bons resultados utilizando essa estratégia, alguns autores (ZENG et al., 2018; KASUN et al., 2013; WANG; LIU, 2017) utilizaram abordagens distintas para encontrar estes pesos e conseguirem melhorar o desempenho das redes treinadas. Além disso, os algoritmos K-ELM ou RK-ELM não utilizam pesos gerados aleatoriamente e ainda assim obtêm bons resultados em diversas aplicações. Apesar das técnicas citadas não utilizarem uma matriz de pesos obtida de forma aleatória como pesos entre a entrada e camada oculta da SLFN, como fazem a maior parte dos algoritmos baseados em ELM, mas sim o resultado de uma operação matemática sobre essa matriz, a nomenclatura ELM foi mantida.

Além da amostragem aleatória, dois principais métodos são utilizados para encontrar os pesos entre a entrada e camada oculta de uma SLFN, no contexto de ELM, que são os métodos de inicialização ortogonal e o de *autoencoder*.

2.3.1 Inicialização ortogonal

Segundo Wang e Liu (2017), a aleatoriedade dos neurônios ocultos é um dos pontos-chave que destaca o ELM de outros algoritmos clássicos, especialmente pela alta eficiência e boa generalização. Porém, a utilização de pesos aleatórios pode degenerar o aprendizado e a capacidade de generalização da SLFN. Assim, um método unificado de projeção ortogonal, que provém a transformação ótima em termos de minimização de perdas de todas as distâncias entre amostras, foi proposto (WANG; LIU, 2017).

Assim, o algoritmo chamado *ELM - Random Orthogonalization* (ELM-RO) foi proposto por Wang e Liu (2017), cuja principal diferença do algoritmo R-ELM padrão é a utilização de uma base ortogonal (que pode ser obtida a partir dos pesos amostrados aleatoriamente) entre as camadas de entrada e oculta da SLFN.

O ELM-RO gera uma matriz de pesos aleatórios $\boldsymbol{W} = [\boldsymbol{w}_1^\mathsf{T} \cdots \boldsymbol{w}_L^\mathsf{T}]^\mathsf{T} \in \mathbb{R}^{\tilde{n} \times L}$, onde $\tilde{n} = \min(L, n)$, com n sendo a dimensão dos dados de entrada. Então, um método capaz de gerar uma base ortogonal a partir de uma matriz, como o Gram-Schmidt ou a Decomposição em valores singulares (SVD), é usado em $\boldsymbol{W}$, resultando em $\boldsymbol{W}^{\text{orth}}$. Segundo Petersen e Pedersen (2007), a SVD sobre uma matriz real gera matrizes ortonormais.

Se a SLFN possui menos nós ocultos que de entrada (i.e., $L < n$), uma etapa adicional é necessária. Uma matriz $\boldsymbol{W}^{\text{pca}} \in \mathbb{R}^{n \times L}$ é calculada usando PCA nos dados de entrada $\boldsymbol{X}$, onde cada uma das L colunas são componentes principais. Então, a matriz

final de pesos é encontrada usando a Equação 2.62.

$$\boldsymbol{W}^{\mathrm{RO}} = \begin{cases} \boldsymbol{W}^{\mathrm{pca}}\boldsymbol{W}^{\mathrm{orth}}, & \text{se } L < n \\ \boldsymbol{W}^{\mathrm{orth}}, & \text{caso contrário} \end{cases} \tag{2.62}$$

Finalmente, ELM-RO utiliza $\boldsymbol{W}^{RO}$ como pesos entre a camada oculta e de entrada da SLFN que está treinando.

2.3.2 *Autoencoder*

Kasun et al. (2013) combinou os conceitos de ELM e de *Autoencoder* para propor os métodos *Autoencoder-ELM* (ELM-AE) e o ML-ELM. Cada camada de ML-ELM é treinada usando um ELM-AE, onde os pesos são encontrados considerando que a entrada e a saída da SLFN são os mesmos dados (as amostras de entrada).

O método ELM-AE realiza seu treinamento em dois passos, onde no primeiro uma SLFN é treinada normalmente com o algoritmo R-ELM, utilizando pesos aleatórios e calculando a matriz $\boldsymbol{B}$ entre a camada oculta e a de saída. Posteriormente, o algoritmo ELM-AE realiza o seu segundo passo, onde $\boldsymbol{B}^{\mathsf{T}}$ é usado como pesos entre a entrada e camada oculta de uma nova SLFN (ou substituindo os pesos da SLFN original), de mesmo tamanho da primeira, que é treinada (ou retreinada), encontrando a matriz de pesos $\boldsymbol{B}_{ae}$

Assim, o conjunto de pesos resultantes do treinamento de uma SLFN usando o ELM-AE são $\boldsymbol{B}^{\mathsf{T}}$ e $\boldsymbol{B}_{ae}$, onde o primeiro é utilizado entre a entrada e a camada oculta, e o segundo é utilizado da oculta para a camada de saída.

Uma vez que $\boldsymbol{H}$ é o espaço de características projetado a partir de $\boldsymbol{X}$ utilizando a função de ativação, Kasun et al. (2013) levanta a hipótese de que a matriz de saída $\boldsymbol{B}$ do ELM-AE aprende a representar as características dos dados de entrada via valores singulares. Após alguns experimentos, foi possível ver que os modelos ELM-AE apresentavam resultados melhores quando seus pesos eram obtidos a partir do SVD (KASUN et al., 2013).

2.4 Considerações Finais

O Quadro 1 resume as características dos métodos apresentados neste capítulo.

Vale ressaltar que este trabalho utiliza principalmente as técnicas DSN, ELM, R-ELM, RK-ELM, OSR-ELM e DOER. Os métodos propostos no Capítulo 3 são derivados do método empilhado DSN quando utilizados em conjunto com algoritmos baseados em ELM, principalmente o R-ELM e o RK-ELM. Estes métodos possuem as mesmas características do método DSN apresentadas no Quadro 1. Já o método proposto no Capítulo 5 é uma variante do método proposto no Capítulo 3, quando utilizado em conjunto com o algoritmo

Quadro 1 – Características dos métodos apresentados. Células com "-" representam uma resposta que depende de outros fatores, como a escolha de um algoritmo de treinamento.

Métodos	Características						
	Permite múltiplas saídas	Regularização de pesos	Capacidade de podar neurônios	Robustez à *Outliers*	Empilhado	Se adapta a novos dados	Lida com *concept drift*
ELM	✓	✗	✗	✗	✗	✗	✗
UWR-ELM	✓	✓	✗	✗	✗	✗	✗
R-ELM	✗	✓	✓	✗	✗	✗	✗
GR-ELM	✓	✓	✓	✗	✗	✗	✗
OR-ELM	✗	✓	✗	✓	✗	✗	✗
GOR-ELM	✓	✓	✓	✓	✗	✗	✗
K-ELM	✓	✓	✗	✗	✗	✗	✗
RK-ELM	✓	✓	✗	✗	✗	✗	✗
ML-ELM	✓	✓	✗	✗	✗	✗	✗
S-ELM	✓	✓	✗	✗	✓	✗	✗
DSN	✓	-	-	-	✓	-	-
OS-ELM	✓	✗	✗	✗	✗	✓	✗
OSR-ELM	✓	✓	✗	✗	✗	✓	✗
Learn^{++}	-	-	-	-	✗	✓	✗
Learn^{++}.NSE	-	-	-	-	✗	✓	✓
OWE	✗	-	-	-	✗	✓	✓
DOER	✗	-	-	-	✗	✓	✓
OEOA	✗	-	-	-	✗	✓	✓

Fonte: Produção do próprio autor.

OSR-ELM. Neste último caso, também é feita uma atualização de parâmetros similar à feita no método DOER, de modo que os membros da rede empilhada atualizem tais parâmetros ao longo do tempo. Este método possui as mesmas características do método OSR-ELM apresentadas no Quadro 1, além de ser um método empilhado e conseguir lidar com *concept drift*.

3 Propostas considerando métodos *Offline*

Neste capítulo são apresentadas algumas limitações dos algoritmos citados na Seção 2.1. Em seguida, são apresentados algoritmos de treinamento para uma rede do tipo DSN que são propostos neste trabalho com o objetivo de superar tais limitações. Na Seção 3.3 é feita uma discussão de complexidade computacional dos métodos propostos. Experimentos são apresentados e discutidos Seção 3.4, enquanto considerações finais são feitas na Seção 3.5.

3.1 Problemas Comuns de Métodos Offline

Ao analisar o desempenho de métodos de aprendizado de máquina, como os baseados em ELM, com relação ao seu tamanho e ao número de amostras das bases de dados, alguns autores como Deng (2014), Deng e Yu (2011), Zhou et al. (2014) concluíram que era possível obter melhoras no desempenho de redes através da utilização do conceito de "empilhamento", proposto por Wolpert (1992), obtendo um bom compromisso de desempenho e uso de memória, através do "empilhamento" de módulos simples. Porém, ainda existem algumas desvantagens ou restrições quanto ao uso das técnicas propostas por estes autores (DSN e S-ELM).

O modelo DSN é composto por módulos, comumente pequenos, de modelos de aprendizado de máquina que possuem um conjunto de parâmetros, como pesos entre camadas, que precisam ser armazenados para o funcionamento correto do modelo. Se cada módulo é uma SLFN, por exemplo, dois conjuntos de pesos são necessários para propagar informação entre camadas. Além disso, a entrada dos módulos aumenta de tamanho, devido à concatenação dos dados de entrada de um módulo com sua saída, como pode ser visto na Equação 2.45 e Figura 4. Isto pode se tornar um problema, pois o treinamento de cada módulo se torna mais custoso que o do anterior, principalmente se o modelo estiver sendo usado em um ambiente com restrição de recursos computacionais.

Ao analisar o método S-ELM, ele também empilha módulos com L neurônios cada. Porém, a dimensão da entrada de cada um deles é sempre a mesma, não sofrendo do problema do modelo DSN. Os módulos treinados por este método também não precisam armazenar suas matrizes de pesos $\boldsymbol{B}_i \in \mathbb{R}^{L \times m}$ entre as camadas ocultas e de saída de cada um dos $S-1$ módulos intermediários, já que o único que prediz uma saída é o último. Porém, estes módulos precisam armazenar a matriz de projeção da sua camada oculta, que será usada para construir a camada oculta do módulo posterior. Tal matriz, $\tilde{\boldsymbol{V}}_i \in \mathbb{R}^{L \times L'}$, normalmente tem mais parâmetros que a matriz $\boldsymbol{B}_i$, pois é comum que a dimensão m de saída do módulo seja menor que L', que é o tamanho reduzido da camada oculta após a

projeção do PCA.

S-ELM ainda possui uma outra desvantagem: o algoritmo PCA é aplicado a uma matriz $\boldsymbol{B}_i^\intercal \in \mathbb{R}^{m \times L}$. Como normalmente $m < L$, ocorre o caso especial do PCA onde o número de autovalores diferentes de zero é limitado a $m - 1$ (TURK; PENTLAND, 1991). Isso significa que o S-ELM consegue propagar no máximo $m - 1$ neurônios da camada oculta de forma significativa, o que pode dificultar o aproveitamento de informações entre módulos em bases de dados cuja dimensão de saída é pequena. Outra desvantagem deste método é que não é possível propagar neurônios em bases de dados que tenham saídas unidimensionais ($m = 1$).

3.2 Métodos Offline Propostos

Nesta seção são apresentados os algoritmos *Fast Deep Stacked Network* e *Fast Kernel Deep Stacked Network*, propostos neste trabalho.

3.2.1 *Fast Deep Stacked Network*

Tendo em vista os problemas apresentados na Seção 3.1, neste trabalho é proposto um algoritmo de treinamento de um método empilhado, baseado em redes do tipo DSN, que usam SLFN em cada um dos módulos, que são treinados com algoritmos baseados em ELM. O uso destes no treinamento das SLFN presentes nos módulos de uma DSN ocasiona maior rapidez no treinamento, quando comparado ao uso do algoritmo *backpropation* no treinamento de tais redes, conforme a proposta original (DENG; YU, 2011). Entretanto, além da escolha do algoritmo de aprendizado, o tempo de treinamento também é afetado por outros fatores, como o aumento da dimensionalidade das entrada dos módulos de uma DSN. Dessa forma, propõe-se um método que, em adição ao uso de um algoritmo baseado em ELM, lida diretamente com o mencionado problema da dimensionalidade, reduzindo o tempo de treinamento e o uso de memória da técnica. Este método é denonimado *Fast Deep Stacked Network* (FDSN).

Numa rede treinada com o algoritmo FDSN, o primeiro módulo é treinado da mesma forma que o de uma rede DSN, usando um algoritmo baseado em ELM: um conjunto de pesos $\boldsymbol{W}_1$ e vieses $\boldsymbol{\nu}_1$ são obtidos (seja por amostragem aleatória ou outro método de inicialização qualquer), e a matriz de saídas da camada oculta $\boldsymbol{H}_1$ é calculada usando a Equação 2.3, que pode ser reescrita como mostra a Equação 3.1,

$$\boldsymbol{H}_1 = \boldsymbol{G}(\boldsymbol{\Pi}_1), \tag{3.1}$$

onde $\boldsymbol{G}(\cdot)$ é um operador que aplica uma função de ativação $h(\cdot)$ em todos os elementos da matriz fornecida $\boldsymbol{\Pi}_1$, que pode ser escrita como mostra a Equação 3.2,

$$\boldsymbol{\Pi}_1 = \boldsymbol{X}_1\boldsymbol{W}_1 + \boldsymbol{N}_1, \tag{3.2}$$

onde $\boldsymbol{X}_1$ é a matriz de entrada do primeiro módulo, onde cada linha é uma amostra da base de dados utilizada para treinar a rede, e cada uma das linhas da matriz $\boldsymbol{N}_1$ é igual ao vetor $\boldsymbol{\nu}_1^{\mathsf{T}}$. O treinamento do primeiro módulo continua de acordo com o algoritmo baseado em ELM escolhido. Como exemplo, se for usado o UWR-ELM, a matriz de pesos $\boldsymbol{B}_1$ pode ser calculada usando a Equação 2.8. Após o treinamento, a saída deste módulo é dada pela Equação 3.3,

$$\hat{\boldsymbol{T}}_1 = \boldsymbol{H}_1\boldsymbol{B}_1. \tag{3.3}$$

Seguindo o treinamento de uma DSN, como proposto por Deng e Yu (2011) e discutido na Seção 2.1.9, a entrada do segundo módulo é dada pela concatenação da matriz de entrada do primeiro módulo com sua matriz de saída, como mostra a Equação 3.4,

$$\boldsymbol{X}_2 = [\boldsymbol{X}_1\ \hat{\boldsymbol{T}}_1], \tag{3.4}$$

que será usada para calcular a matriz de saída da camada oculta $\boldsymbol{H}_2$ do segundo módulo, que pode ser reescrita na mesma notação das Equações 3.1 e 3.2, como mostra a Equação 3.5,

$$\boldsymbol{H}_2 = \boldsymbol{G}(\boldsymbol{\Pi}_2), \tag{3.5}$$

onde $\boldsymbol{\Pi}_2$ é dado pela Equação 3.6,

$$\boldsymbol{\Pi}_2 = \boldsymbol{X}_2\boldsymbol{W}_2 + \boldsymbol{N}_2. \tag{3.6}$$

Por sua vez, a Equação 3.6 pode ser reescrita na forma da Equação 3.7,

$$\boldsymbol{\Pi}_2 = [\boldsymbol{X}_1\ \hat{\boldsymbol{T}}_1]\begin{bmatrix}\boldsymbol{W}_2^{(1)}\\ \boldsymbol{W}_2^{(2)}\end{bmatrix} + \boldsymbol{N}_2, \tag{3.7}$$

onde $\boldsymbol{W}_2^{(1)}$ e $\boldsymbol{W}_2^{(2)}$ são submatrizes de $\boldsymbol{W}_2 \in \mathbb{R}^{(n+m)\times L}$, onde $\boldsymbol{W}_2^{(1)} \in \mathbb{R}^{n\times L}$ possui as n primeiras linhas de $\boldsymbol{W}_2$ e $\boldsymbol{W}_2^{(2)} \in \mathbb{R}^{m\times L}$ as demais m linhas. Desta forma, é possível reescrever a Equação 3.7 de acordo com a Equação 3.8,

$$\boldsymbol{\Pi}_2 = \boldsymbol{X}_1\boldsymbol{W}_2^{(1)} + \hat{\boldsymbol{T}}_1\boldsymbol{W}_2^{(2)} + \boldsymbol{N}_2. \tag{3.8}$$

É possível notar na Equação 3.8 que $\boldsymbol{\Pi}_2$, assim como $\boldsymbol{\Pi}_1$, possui um termo que multiplica os dados de entrada $\boldsymbol{X}$ e um termo que representa os viéses. Porém, $\boldsymbol{\Pi}_2$ possui um terceiro termo, $\hat{\boldsymbol{T}}_1\boldsymbol{W}_2^{(2)}$, adicionado aos demais. Isto vale também para outros módulos, isto é, $\boldsymbol{\Pi}_i$ possui uma soma e uma multiplicação matricial a mais que $\boldsymbol{\Pi}_{i-1}$, somado a um termo $\hat{\boldsymbol{T}_{i-1}}\boldsymbol{W}_i^{(2)}$. Isto causa um aumento do tempo de treinamento e do uso de memória (número de parâmetros do modelo), já que o número de cálculos necessários para treinar um módulo é sempre maior que o do anterior, como foi discutido na Seção 3.1.

Ao invés de considerar que cada módulo é treinado separadamente, com novos valores para seus pesos, o algoritmo FDSN faz as considerações apresentadas nas Equações

3.9 e 3.10 para acelerar o processo de treinamento,

$$\boldsymbol{W}_2^{(1)} = \boldsymbol{W}_1 \tag{3.9}$$

$$\boldsymbol{N}_2 = \boldsymbol{N}_1. \tag{3.10}$$

Desta forma, a Equação 3.8 pode ser reescrita como mostra a Equação 3.11,

$$\boldsymbol{\Pi}_2 = \boldsymbol{X}_1\boldsymbol{W}_1 + \hat{\boldsymbol{T}}_1\boldsymbol{W}_2^{(2)} + \boldsymbol{N}_1, \tag{3.11}$$

que possui um termo em comum com a Equação 3.2. Ao aplicar a substituição na Equação 3.11, é obtida a Equação 3.12,

$$\boldsymbol{\Pi}_2 = \boldsymbol{\Pi}_1 + \hat{\boldsymbol{T}}_1\boldsymbol{W}_2^{(2)}, \tag{3.12}$$

onde $\boldsymbol{W}_2^{(2)} \in \mathbb{R}^{m\times L}$ é uma nova matriz de pesos. Da mesma forma que no primeiro módulo, a matriz $\boldsymbol{B}_2$ é calculada usando um algoritmo baseado em ELM, a ser escolhido pelo usuário, e a saída do segundo módulo é dada pela Equação 3.13,

$$\hat{\boldsymbol{T}}_2 = \boldsymbol{H}_2\boldsymbol{B}_2. \tag{3.13}$$

O método descrito pode ser estendido para calcular as matrizes $\boldsymbol{\Pi}_i$ dos demais módulos da rede FDSN, como mostra a Equação 3.14,

$$\boldsymbol{\Pi}_i = \boldsymbol{\Pi}_{i-1} + \hat{\boldsymbol{T}}_{i-1}\boldsymbol{W}_i^{(2)}. \tag{3.14}$$

Ao fazer as considerações das Equações 3.9 e 3.10, o algoritmo FDSN pode reaproveitar cálculos feitos no treino de um módulo anterior no seguinte, reduzindo o tempo total de treinamento da rede. Outra vantagem é que não é necessário gerar uma nova matriz de pesos em cada módulo, já que pesos são reaproveitados. Assim, cada um dos S módulos da rede (com exceção do primeiro) só precisa armazenar uma matriz de pesos $\boldsymbol{W}_i^{(2)}$ para ser capaz de fazer a atualização da Equação 3.12. Isso também implica em um menor número de parâmetros necessários para o funcionamento da rede.

Em compensação, o fato de reutilizar pesos ao longo dos módulos também pode ser um problema para uma rede treinada com FDSN. Assim, este algoritmo é suscetível ao conjunto inicial de pesos do primeiro módulo, já que eles não vão ser alterados. Se um conjunto não adequado for escolhido, assim como pode acontecer ao utilizar pesos aleatórios em algoritmos ELM (WANG; LIU, 2017), os desempenhos de todos os módulos poderão ser prejudicados, e consequentemente o da rede como um todo.

Dessa forma, é importante que o conjunto inicial de pesos seja devidamente escolhido, e isso pode ser alcançado associando uma técnica de inicialização de pesos ao algoritmo de treinamento de uma FDSN. Vale notar que, ao empregar algoritmos baseados em ELM, é pertinente associar técnicas de inicialização de pesos já validadas no contexto de ELM. Aliado a isso, o uso dos algoritmos baseados em ELM capazes de podar neurônios considerados inadequados também pode impactar positivamente nos resultados.

3.2.2 *Fast Kernel Deep Stacked Network*

Como mencionado na Seção 3.2.1, um dos problemas da técnica FDSN é que a inicialização de seus pesos é feita de forma aleatória, e se o conjunto não for devidamente escolhido, o desempenho de toda a rede fica comprometido. Já na Seção 2.1.6 foi apresentada uma alternativa ao mapeamento aleatório de características dos métodos ELM, que é a aplicação de *kernels*, o que origina o algoritmo K-ELM, que possui limitações quanto ao seu uso em conjunto com bases de dados com um número considerável de amostras. Nesses casos, o algoritmo RK-ELM é mais adequado (DENG; ONG; ZHENG, 2016).

Assim, este trabalho propõe uma variação do método FDSN, denominada *Fast Kernel Deep Stacked Network* (FKDSN), que utiliza *kernels* e o algoritmo RK-ELM para treinar os módulos de uma rede DSN, mantendo características do FDSN como a velocidade e o reaproveitamento de pesos (que no caso serão "vetores suporte"), mas sem a aleatoriedade do mapeamento dos algoritmos baseados em ELM.

O primeiro módulo do método FKDSN é treinado usando o algoritmo RK-ELM comum: um número L de "vetores suporte" $\boldsymbol{\Delta} = [\boldsymbol{\delta}_1^\mathsf{T} \cdots \boldsymbol{\delta}_L^\mathsf{T}]^\mathsf{T}$ são amostrados aleatoriamente do conjunto de treinamento $\boldsymbol{X}$, com $L \leq N$ e a matriz $\boldsymbol{H}^{(1)}_{\text{RK-ELM}} \in \mathbb{R}^{N \times L}$ pode ser calculada como mostrado na Equação 2.40. Esta matriz pode ser reescrita como mostra a Equação 3.15, onde $\boldsymbol{G}$ é um operador que aplica a função não linear de kernel $K(\cdot, \cdot)$ para cada elemento da matriz $\boldsymbol{\Pi}^{(1)}$[1].

$$\boldsymbol{H}^{(1)}_{\text{RK-ELM}} = \boldsymbol{G}(\boldsymbol{\Pi}^{(1)}) \tag{3.15}$$

Assumindo que o kernel aplica uma função $K(\cdot, \cdot)$ aos vetores de $\boldsymbol{X}$ e $\boldsymbol{\Delta}$, é possível guardar a matriz $\boldsymbol{\Pi}^{(k)}$ e utilizá-la para encontrar a matriz $\boldsymbol{\Pi}^{(k+1)}$, de uma forma similar ao feito na técnica FDSN. As equações a seguir foram desenvolvidas considerando o *kernel* gaussiano, mas considerações similares podem ser feitas com outros tipos de *kernel*, como o linear ou o polinomial.

Considerando o *kernel* gaussiano com parâmetro γ aplicado a dois vetores $\boldsymbol{x}_i$ e δ_j, como mostra a Equação 3.16, pode-se escrever o elemento (i, j) da matriz $\boldsymbol{\Pi}^{(1)}$ conforme a Equação 3.17, calcular a matriz $\boldsymbol{H}^{(1)}_{\text{RK-ELM}}$ como mostra a Equação 3.15, e os pesos de saída $\boldsymbol{B}^{(1)}_{\text{RK-ELM}}$ podem ser encontrados de acordo com a Equação 3.18.

$$K(\boldsymbol{x}_i, \delta_j) = \exp\left(-\gamma \parallel \boldsymbol{x}_i - \delta_j \parallel_2^2\right) = \boldsymbol{G}(-\gamma(\boldsymbol{x}_i - \delta_j)^\mathsf{T}(\boldsymbol{x}_i - \delta_j)) \tag{3.16}$$

$$\boldsymbol{\Pi}^{(1)}_{(i,j)} = (\boldsymbol{x}_i - \delta_j)^\mathsf{T}(\boldsymbol{x}_i - \delta_j) \tag{3.17}$$

$$\boldsymbol{B}^{(1)}_{\text{RK-ELM}} = \left(\left(\boldsymbol{H}^{(1)}_{\text{RK-ELM}}\right)^\mathsf{T} \boldsymbol{H}^{(1)}_{\text{RK-ELM}} + \frac{\boldsymbol{I}}{C}\right)^{-1} \boldsymbol{H}^\mathsf{T}_{\text{RK-ELM}} \boldsymbol{T}. \tag{3.18}$$

[1] Note que nesta seção o índice sobrescrito entre parênteses se refere ao número do módulo.

Com a matriz $\boldsymbol{B}^{(1)}_{\text{RK-ELM}}$, é possível encontrar a saída do primeiro módulo $\hat{\boldsymbol{T}}_1 = [\hat{\boldsymbol{t}}_1^{(1)\mathsf{T}}, \ldots, \hat{\boldsymbol{t}}_N^{(1)\mathsf{T}}]$ com respeito a entrada $\boldsymbol{X}$ através da Equação 3.19.

$$\hat{t}^{(1)}(\boldsymbol{x}_j) = \hat{t}_j^{(1)} = \begin{bmatrix} K(\boldsymbol{x}_j, \delta_1) \\ \vdots \\ K(\boldsymbol{x}_j, \delta_L) \end{bmatrix}^{\mathsf{T}} \hat{\boldsymbol{B}}^{(1)}_{\text{RK-ELM}} \tag{3.19}$$

Assim como no FDSN, a entrada do segundo módulo é construída a partir da concatenação da entrada do primeiro módulo com a sua saída, como mostra as Equações 3.20 e 3.21, e essa entrada será usada para calcular a saída da camada oculta do segundo módulo. Assim, é preciso encontrar a matriz $\boldsymbol{\Pi}_2$, cujos elementos podem ser encontrados através da Equação 3.23, onde $\hat{\boldsymbol{y}}_j^{(1)} = \hat{\boldsymbol{t}}^{(1)}(\boldsymbol{\delta}_j)$ é a saída do primeiro módulo com respeito a entrada $\boldsymbol{\delta}_j$.

$$\boldsymbol{X}^{(2)} = [\boldsymbol{X}^{(1)}, \hat{\boldsymbol{T}}^{(1)}] \tag{3.20}$$

$$\boldsymbol{x}_i^{(2)} = [\boldsymbol{x}_i^{(1)}, \hat{\boldsymbol{t}}_i^{(1)}] \tag{3.21}$$

$$\boldsymbol{\Pi}^{(2)}_{(i,j)} = (\boldsymbol{x}_i^{(2)} - \delta_j^{(2)})^{\mathsf{T}}(\boldsymbol{x}_i^{(2)} - \delta_j^{(2)}) \tag{3.22}$$

$$\boldsymbol{\Pi}^{(2)}_{(i,j)} = \left([\boldsymbol{x}_i^{(1)}\ \hat{\boldsymbol{t}}_j^{(1)}] - [\boldsymbol{\delta}_j\ \hat{\boldsymbol{y}}_j^{(1)}]\right)^{\mathsf{T}} \left([\boldsymbol{x}_i^{(1)}\ \hat{\boldsymbol{t}}_j^{(1)}] - [\boldsymbol{\delta}_j\ \hat{\boldsymbol{y}}_j^{(1)}]\right) \tag{3.23}$$

A Equação 3.23 pode ser reescrita como mostra as Equações 3.24, 3.25 e 3.26.

$$\boldsymbol{\Pi}^{(2)}_{(i,j)} = [(\boldsymbol{x}_i^{(1)} - \boldsymbol{\delta}_j)\ (\hat{\boldsymbol{t}}_j^{(1)} - \hat{\boldsymbol{y}}_j^{(1)})]^{\mathsf{T}}[(\boldsymbol{x}_i^{(1)} - \boldsymbol{\delta}_j)\ (\hat{\boldsymbol{t}}_j^{(1)} - \hat{\boldsymbol{y}}_j^{(1)})] \tag{3.24}$$

$$\boldsymbol{\Pi}^{(2)}_{(i,j)} = (\boldsymbol{x}_i^{(1)} - \delta_j)^{\mathsf{T}}(\boldsymbol{x}_i^{(1)} - \delta_j) + (\hat{\boldsymbol{t}}_j^{(1)} - \hat{\boldsymbol{y}}_j^{(1)})^{\mathsf{T}}(\hat{\boldsymbol{t}}_j^{(1)} - \hat{\boldsymbol{y}}_j^{(1)}) \tag{3.25}$$

$$\boldsymbol{\Pi}^{(2)}_{(i,j)} = \boldsymbol{\Pi}^{(1)}_{(i,j)} + (\hat{\boldsymbol{t}}_j^{(1)} - \hat{\boldsymbol{y}}_j^{(1)})^{\mathsf{T}}(\hat{\boldsymbol{t}}_j^{(1)} - \hat{\boldsymbol{y}}_j^{(1)}) \tag{3.26}$$

Desta forma, é possível encontrar as matrizes $\boldsymbol{\Pi}^{(2)}$ e $\boldsymbol{H}^{(2)}_{\text{RK-ELM}}$ usando as Equações 3.26 e 3.15, respectivamente. Este método de atualização pode ser estendido para os demais módulos de uma FKDSN, como mostram as Equações 3.27 e 3.28, e os pesos de saída $\boldsymbol{B}^{(k)}_{\text{RK-ELM}}$ do k-ésimo módulo podem ser encontrados de acordo com a Equação 3.18.

$$\boldsymbol{\Pi}^{(k+1)}_{(i,j)} = \boldsymbol{\Pi}^{(k)}_{(i,j)} + (\hat{\boldsymbol{t}}_j^{(k)} - \hat{\boldsymbol{y}}_j^{(k)})^{\mathsf{T}}(\hat{\boldsymbol{t}}_j^{(k)} - \hat{\boldsymbol{y}}_j^{(k)}) \tag{3.27}$$

$$\boldsymbol{H}^{(k)}_{\text{RK-ELM}} = \boldsymbol{G}(\boldsymbol{\Pi}^{(k)}) \tag{3.28}$$

Como mencionado, FKDSN mantém algumas características da FDSN: possui treinamento acelerado e necessita de menos memória para ser armazenada que uma DSN

usual, sendo adequada para sistemas com recursos limitados. Embora não tenha o problema de pesos iniciais completamente aleatórios, o FKDSN ainda é susceptível a uma má escolha dos "vetores suporte" iniciais, já que um conjunto não adequado de amostras pode ser sorteado. De qualquer forma, embora possa ser um conjunto não adequado, esse conjunto ainda é um subconjunto da base de dados analisada, o que diminui a influência aleatória do algoritmo ELM.

3.2.3 Resumo das Técnicas Propostas

Na Seção 3.2.1 foi proposta a técnica FDSN, que é uma variante da DSN que reutiliza pesos e aplica um algoritmo baseado em ELM, para reduzir o tempo de treinamento e o uso de memória. A DSN possui a característica de que a entrada de um módulo é a concatenação da saída com a entrada do anterior. Assim, há uma semelhança no cálculo de algumas matrizes entre os módulos. O FDSN se aproveita dessa semelhança e compartilha pesos entre módulos. Logo, a arquitetura utiliza um menor número de parâmetros distintos, reduzindo a quantidade de memória para armazená-la. Além disso, cálculos em um módulo podem ser reaproveitados no posterior, originando uma forma recursiva de calcular alguns parâmetros no treinamento da rede, acelerando o mesmo.

O reaproveitamento de parâmetros, contudo, traz desvantagens devido ao fato de que eles são obtidos aleatoriamente nos algoritmos baseados em ELM. Como variante do FDSN, o método FKDSN também foi proposto, utilizando *kernels* ao invés de pesos aleatórios, quando isso for desejado. Essa técnica também faz a consideração de reaproveitamento de pesos, e possui as mesmas vantagens do FDSN, porém ela é menos suscetível a parâmetros aleatórios, já que utiliza *kernels*.

3.3 Complexidade Computacional dos algoritmos FDSN e FKDSN

Esta seção tem a finalidade de comparar as complexidades assintóticas dos algoritmos citados, como o ELM, S-ELM, DSN e FDSN. O tempo e a memória necessários para treinar a rede são comparados, juntamente com o número de parâmetros necessários para o funcionamento dos respectivos modelos. É possível notar que a complexidade do algoritmo FKDSN é similar à complexidade do FDSN, substituindo o número de neurônios ocultos do FDSN pelo número de "vetores suporte" do FKDSN.

Segundo Zhou et al. (2014), para lidar com largas bases de dados com N amostras, usualmente é necessário um grande número L de neurônios na camada oculta de uma SLFN para que eles adquiram uma boa capacidade de generalização dos dados. O número de características da base de dados é representado por d, e o número de saídas (alvos) dessa mesma base é representado por m.

3.3.1 Análisa Assintótica do Tempo de Treinamento

De acordo com Akusok et al. (2015), a complexidade do algoritmo ELM é da ordem de $\mathcal{O}(NL^2 + L^3)$, o que implica que o tempo de treinamento de uma SLFN através do ELM aumenta substancialmente com o aumento do número de amostras e de neurônios em sua camada oculta. Para o caso do algoritmo UWR-ELM, a mesma complexidade assintótica é válida, pois são incluídas apenas operações pouco custosas no algoritmo de treinamento.

De forma similar, é possível fazer a análise assintótica do tempo de treinamento dos algoritmos DSN (considerando módulos treinados com ELM) e FDSN, que consistem em treinar S módulos com tais algoritmos. Isto implica que o custo computacional destes métodos é dado por $\mathcal{O}(SNL^2 + SL^3)$.

Já para a técnica S-ELM, também são treinados S módulos, onde nos intermediários é feita uma operação de redução de dimensionalidade usando PCA. De acordo com Elgamal et al. (2015), a complexidade dessa operação em uma matriz de tamanho $m \times L$ é $\mathcal{O}(mL^2 + L^3)$, onde m é a dimensão de saída das redes, no contexto de ELM. Assim, a complexidade total da S-ELM é $\mathcal{O}(SmL^2 + SNL^2 + SL^3)$, que pode ser reduzida para $\mathcal{O}(SNL^2 + SL^3)$, já que usualmente $m \ll N$.

Ao comparar as complexidades das técnicas citadas, é possível perceber que existe um *trade-off* entre o número de módulos de uma técnica empilhada e a quantidade de neurônios que compõem cada camada oculta. Se for considerada que a ideia do conceito de empilhamento de técnicas de aprendizado de máquina (WOLPERT, 1992) é utilizar módulos pequenos, pode-se perceber que os modelos empilhados possuem uma vantagem no tempo de treinamento, já que S não está elevado à nenhuma potência, enquanto L está, considerando as complexidades Big-O dos métodos. O Quadro 2 resume as complexidades citadas, quando o tempo de treinamento é levado em consideração.

Quadro 2 – Resumo das complexidades dos métodos analisados, quando o tempo de treinamento é levado em consideração.

Método	Complexidade
UWR-ELM	$\mathcal{O}(NL^2 + L^3)$
DSN (UWR-ELM)	$\mathcal{O}(SNL^2 + SL^3)$
FDSN (UWR-ELM) / FKDSN	$\mathcal{O}(SNL^2 + SL^3)$
S-ELM	$\mathcal{O}(SmL^2 + SNL^2 + SL^3)$

Fonte: Produção do próprio autor

3.3.2 Análisa Assintótica da Memória Usada no Treino

Uma análise similar pode ser feita para estimar o uso de memória para o treinamento desses algoritmos. De acordo com Akusok et al. (2015), o custo de memória do algoritmo

ELM (e suas versões regularizadas) é de $\mathcal{O}(NL)$, já que a matriz de maior dimensão a ser manipulada é a matriz $\boldsymbol{H}$, de tamanho $N \times L$. Como notado por Zhou et al. (2014), isto pode ser um problema caso sejam usadas muitas amostras ou a rede tenha muitos neurônios na camada oculta.

Ao analisar a complexidade de memória do algoritmo DSN, a dimensão da entrada do módulo de número $i+1$, representada por n_i, é maior que a do i-ésimo módulo. Esta dimensão segue a Equação 3.29,

$$n_i = n + (i-1)m. \tag{3.29}$$

Assim, deve-se considerar o último módulo, que representa o pior caso na utilização de memória, pois ele lida com matrizes de maior dimensão. O treinamento deste módulo lida com as matrizes $\boldsymbol{X}_S$ e $\boldsymbol{W}_S$, que possuem dimensão $N \times (n+(S-1)m)$ e $L \times (n+(S-1)m)$, respectivamente. Normalmente $N > L$, o que significa que a matriz de maior dimensão entre as duas citadas costuma ser $\boldsymbol{X}_S$.

Assim, a complexidade assintótica de memória do treinamento de uma DSN pode ser dada pela maior dimensão entre as matrizes $\boldsymbol{X}_S$ e $\boldsymbol{H}_S$, o que pode ser resumido pela expressão $\mathcal{O}(\max(N(n+(S-1)m), NL))$.

Já que os módulos de uma rede treinada com o algoritmo FDSN apenas atualizam as suas matrizes $\boldsymbol{\Pi}_i$, eles não constroem uma matriz $\boldsymbol{X}_S$, e portanto a complexidade de memória deste algoritmo é $\mathcal{O}(NL)$.

O algoritmo S-ELM também treina S módulos, um por vez, o que indica que a sua complexidade de memória também poderia ser $\mathcal{O}(NL)$. Entretanto, em cada módulo é realizada uma etapa de redução de dimensão através do PCA. Esta operação tem complexidade de memória de $\mathcal{O}(\max((m+L)L', L^2))$. No caso do S-ELM, $L' < L$ e normalmente $m < L$, então a complexidade pode ser simplificada para $\mathcal{O}(L^2)$. Portanto, a complexidade de memória total do algoritmo S-ELM é dada por $\mathcal{O}(\max(L^2, NL))$.

Quando a complexidade de métodos empilhados é comparada com o algoritmo ELM, é possível ver que o treinamento destes usualmente requer menos memória, já que a ideia por trás é usar módulos com menos neurônios. Entretanto, existem casos onde o método DSN pode exigir mais memória em seu treino que o ELM, quando muitos módulos são utilizados. O Quadro 3 resume as complexidades citadas, quando a memória necessária para treinamento é levada em consideração.

3.3.3 Número de Parâmetros dos Modelos

Outro aspecto importante de modelos de aprendizado de máquina é o número de parâmetros (ou bytes) necessários para armazená-los em um computador. Modelos com menos parâmetros normalmente levam menos tempo para estimar uma saída a partir de

Quadro 3 – Resumo das complexidades dos métodos analisados, quando a memória necessária para treinamento é levada em consideração

Método	Complexidade
UWR-ELM	$\mathcal{O}(NL)$
DSN (UWR-ELM)	$\mathcal{O}(\max(N(n+(S-1)m, NL))$
FDSN (UWR-ELM) / FKDSN	$\mathcal{O}(NL)$
S-ELM	$\mathcal{O}(\max(N^2, NL))$

Fonte: Produção do próprio autor

uma entrada e podem ser aplicados em ambientes com menos recursos, devido a menor necessidade de memória para armazená-los.

Ao considerar modelos SLFN, é necessário armazenar dois conjuntos de parâmetros que representam os mapeamentos de entrada e de saída, isto é, duas matrizes $\boldsymbol{W}$, $\boldsymbol{B}$ e um vetor $\boldsymbol{\nu}$, que representa os vieses da camada oculta. As dimensões desses parâmetros são $n \times L$, $L \times m$, e L, respectivamente. Assim, a memória total necessária para armazenar um modelo SLFN é resumida na Equação 3.30,

$$Z_{\text{SLFN}} = L(n + m + 1). \tag{3.30}$$

Um modelo S-ELM possui S módulos, onde o primeiro precisa armazenar as matrizes $\boldsymbol{W}_1$ e $\boldsymbol{\nu}_1$, e o último a matriz $\boldsymbol{B}_S$. Como uma redução de dimensionalidade é feita, neurônios são reaproveitados e os módulos de número 2 até S precisam gerar novos nós para compensar tal operação, o que significa que é necessário armazenar matrizes $\boldsymbol{W}_{\text{new}}$ e $\boldsymbol{\nu}_{\text{new}}$, de dimensão $L - L^{'}$. Além destas, do primeiro até o penúltimo módulo, é necessário armazenar a matriz de projeção $\tilde{\boldsymbol{V}}_i$, de dimensão $L \times L^{'}$. Assim, a memória necessária para armazenar um modelo S-ELM é resumida na Equação 3.31,

$$Z_{\text{S-ELM}} = L(n + m + 1) + (S - 1)(LL^{'} + Ln - L^{'}n + L - L^{'}). \tag{3.31}$$

Já para um modelo treinado pelo algoritmo DSN, deve-se considerar o aumento na dimensão de entrada de cada módulo, dada pela Equação 3.29. Considerando que cada módulo é uma SLFN treinada com ELM, o número total de parâmetros de uma DSN é resumido na Equação 3.32[2],

$$Z_{\text{DSN}} = L(n + m + 1) + L\left(\sum_{i=1}^{S-1}(1 + m + n + mi)\right). \tag{3.32}$$

Para uma arquitetura treinada com o algoritmo FDSN, o primeiro módulo é uma SLFN padrão, treinada com ELM. Os demais módulos precisam armazenar apenas os novos pesos de entrada, isto é, as matrizes $\boldsymbol{W}_i^{(2)}$, e os pesos de saída $\boldsymbol{B}_i$, de tamanhos

[2] Note que $\sum_{i=2}^{S}(i-1) = \sum_{i=1}^{S-1}(i)$

$m \times L$ e $L \times m$, respectivamente. Assim, o número total de parâmetros de uma rede treinada com FDSN é resumido na Equação 3.33,

$$Z_{\text{FDSN}} = L(n + m + 1) + 2(S - 1)mL. \tag{3.33}$$

Comparando o número de parâmetros de cada modelo usando o princípio de empilhamento, e considerando que usualmente há mais atributos de entrada na rede do que saídas, isto é, $m < n$, é possível observar que um modelo FDSN precisa de uma quantidade inferior de memória para ser armazenado. Quando comparado à uma rede SLFN com um grande número de neurônios, a tendência é que o FDSN necessite de uma quantidade inferior de memória. O Quadro 4 resume as complexidades citadas, quando o número de parâmetros é levado em consideração.

Quadro 4 – Resumo das complexidades dos métodos analisados, quando o número de parâmetros é levado em consideração

Método	Complexidade
UWR-ELM	$L(n + m + 1)$
DSN (UWR-ELM)	$L(n + m + 1) + L(\sum_{i=1}^{S-1}(1 + m + n + mi))$
FDSN (UWR-ELM) / FKDSN	$L(n + m + 1) + 2(S - 1)mL$
S-ELM	$L(n + m + 1) + (S - 1)(LL' + Ln - L'n + L - L')$

Fonte: Produção do próprio autor

3.4 Experimentos Considerando FDSN e FKDSN

Nesta seção é feita uma avaliação dos métodos propostos na Seção 3.2 em diferentes situações. Na Seção 3.4.1 são apresentados os conjuntos de dados, métricas e testes estatísticos utilizados, especificação dos hiperparâmetros dos métodos e alguns detalhes de implementação e execução. Na Seção 3.4.2 é feita uma análise considerando bases de dados artificiais com diferentes tamanhos. Na Seção 3.4.3 é feita uma análise considerando bases de dados escolhidas e diferentes arquiteturas de redes FDSN. Isto é, diferentes números de módulos e diferentes números de neurônios na camada oculta de cada módulo. Na Seção 3.4.4 são feitos experimentos e comparações das técnicas propostas em problemas de classificação e regressão, com saídas unidimensionais e multidimensionais, utilizando métricas e testes de hipótese.

3.4.1 Metodologia dos Experimentos

Esta seção detalha informações dos conjuntos de dados, métricas e testes estatísticos utilizados nos experimentos. Também é detalhado como os hiperparâmetros dos métodos foram especificados e alguns detalhes de implementação e execução.

3.4.1.1 Conjuntos de dados utilizados

Exceto quando explicitado, os conjuntos de dados de problemas de classificação multi-classe utilizados para avaliação das técnicas *offline* propostas neste trabalho são os apresentados na Tabela 1. O mesmo vale para os conjuntos de dados de problemas de regressão *single* e *multi-target*, apresentados na Tabela 2. Todos os conjuntos de dados são públicos e foram obtidos de problemas reais, disponíveis nos repositórios Keel (ALCALÁ-FDEZ et al., 2011), UCI (DUA; GRAFF, 2017) e LIBSVM (CHANG; LIN, 2011).

Tabela 1 – Conjuntos de dados de problemas de classificação utilizadas nos experimentos. As colunas mostram o número de amostras N utilizadas para treino e teste, número de atributos d e número de classes m.

Nome do Conjunto	#Amostras de Treino	#Amostras de Teste	#Atributos	#Classes
CNAE-9	1080	-	857	9
Coil2000	9822	-	85	2
Contraceptive	1473	-	9	3
Marketing	6876	8993	13	9
Page-blocks	5472	-	10	5
Penbased	10922	-	10	5
Phoneme	5404	-	5	2
SatImage	4435	2000	36	6
Segment	2310	-	19	7
Spambase	4597	-	57	2
Thyroid	7200	-	21	3
Titanic	2201	-	3	2
Vowel	528	462	10	11
winequality-red	1599	-	11	11
winequality-white	4898	-	11	11

Fonte: Produção do próprio autor.

Vale notar que, neste trabalho, as m classes de problemas de classificação são representadas utilizando *one-hot encoding*, ou seja, a saída é representada por um vetor de tamanho m com todos os elementos com valor zero, com exceção de um deles, que possui valor 1 para indicar a classe que pertence à amostra correspondente. Um outro detalhe é o fato de que existem alguns conjuntos de dados, como a SatImage, que já fornece um subconjunto previamente separado para testes. Quando este não é fornecido, as bases de dados são divididas aleatoriamente em um subconjunto de treino e um de teste, respeitando a proporção de 2/3 das amostras para treino e o restante para testes.

3.4.1.2 Métricas e Testes Estatísticos

Para os testes de problemas de classificação, a métrica Acurácia é utilizada. Esta representa a razão entre o número de amostras classificadas corretamente pelo método

Tabela 2 – Bases de dados de problemas de regressão utilizadas nos experimentos. As colunas mostram, para cada base de dados, o número de amostras N, número de atributos d e número de alvos m.

Base	#Amos.	#Atr.	#Alvos	Base	#Amos.	#Atr.	#Alvos
abalone	4177	8	1	laser	993	4	1
ailerons	13750	40	1	mortgage	1049	15	1
airfoil	1503	5	1	mv	40768	10	1
anacalt	4052	7	1	Naval Prop.	11934	16	2
bank32nh	8192	32	1	plastic	1650	2	1
bank8fm	8192	8	1	pole	14998	2	1
california	20640	8	1	puma32h	8192	32	1
carbon nanot.	10721	5	3	puma8NH	8192	8	1
comb. cycle	9568	4	1	quake	2178	3	1
compactiv	8192	21	1	rf1	9005	64	8
concrete	1030	8	1	rf2	7679	576	8
cpu_act	8192	21	1	scm1d	9803	280	16
cpu_small	8192	12	1	scm20d	8966	61	16
delta_ail	7129	5	1	stock	950	9	1
delta_elv	9517	6	1	Supercond.	21263	81	1
ele2	1056	4	1	tic	9822	85	1
elevators	6599	18	1	treasury	1049	15	1
friedman	1200	5	1	wankara	1609	9	1
house	22784	16	1	wizmir	1461	9	1
house8L	22784	16	1	wq	1060	16	14
kin8nm	22784	16	1	-	-	-	-

Fonte: Produção do próprio autor.

avaliado e o número total de amostras, multiplicado por 100%. Esta métrica possui um valor limitado entre 0 e 100% e quanto maior o seu valor, melhor é a eficácia do método para realizar a classificação.

Para os testes em bases de dados de problemas de regressão, a métrica escolhida foi a aRRMSE (*average relative root mean square error*), utilizada por autores como Inaba (2018) e Silva et al. (2020b). Note que por se tratar de uma métrica de erro, quanto menor o seu valor, melhor é a eficácia do método.

Considerando que $\boldsymbol{t}^{(\ell)} = \left(t_1^{(\ell)} \cdots t_N^{(\ell)}\right)^\top$, com $\ell = 1, 2, \ldots, m$, é a ℓ-ésima saída do problema, a métrica aRRMSE é definida como a média da RRMSE (*relative root mean square error*) de cada uma das m saídas do problema:

$$\begin{aligned} aRRMSE(\hat{\boldsymbol{T}}, \boldsymbol{T}) &= \frac{1}{m}\sum_{l=1}^{m} RRMSE(\hat{\boldsymbol{t}}^{(l)}, \boldsymbol{t}^{(l)}) \\ &= \frac{1}{m}\sum_{l=1}^{m} \sqrt{\frac{\sum_{i=1}^{N}(\hat{t_i}^{(l)} - t_i^{(l)})^2}{\sum_{i=1}^{N}(\bar{\boldsymbol{t}}^{(l)} - t_i^{(l)})^2}}, \end{aligned} \tag{3.34}$$

onde $\hat{\boldsymbol{t}}^{(l)} = \left(\hat{t}_1^{(l)} \cdots \hat{t}_N^{(l)}\right)^\mathsf{T}$ é a l-ésima saída de uma SLFN em relação à uma entrada $\boldsymbol{X}$, e $\bar{\boldsymbol{t}}^{(l)} = (1/N)\sum_{i=1}^{N} t_i^{(l)}$.

Tanto para problemas de regressão quanto de classificação, também foram analisados o tempo gasto pelos métodos para realizar o seu treinamento e a quantidade de bytes usados para armazenamento dos modelos.

Para uma melhor comparação do desempenho de diversos métodos sobre os diversos bancos de dados analisados, são utilizados dois testes estatísticos: o de Friedman e o de Nemenyi (DEMŠAR, 2006). Tais testes tem como pressuposto que todos os métodos foram avaliados nas mesmas condições (partições de dados e métricas, por exemplo), e não avaliam diretamente as métricas obtidas, mas sim o *rank* de cada método. Tal abordagem foi utilizada por Losing, Hammer e Wersing (2018) e Fernández-Delgado et al. (2014) para comparação de diversos métodos sobre muitas bases de dados.

Segundo Demšar (2006), procedimentos estatísticos são mais adequados para análise de diferenças entre médias de resultados, e recomenda o uso do teste de Friedman em conjunto com um teste *post-hoc* para comparação de mais de duas técnicas de aprendizado de máquina quando estas são avaliadas sobre vários conjuntos de dados. Além de ser não-paramétrico e robusto, experimentos mostraram que este teste concorda, na maioria das vezes, com o teste paramétrico ANOVA quando as premissas de distribuição dos dados são atendidas.

O teste de Friedman (FRIEDMAN, 1937) ordena os métodos de acordo com o seu desempenho, onde o melhor método recebe *rank* 1, o segundo melhor 2, e assim por diante (para empates, um *rank* médio é usado). Considerando que r_i^j é o *rank* do j-ésimo algoritmo dos F testados em uma i-iésima bases de dados das D utilizadas, pode-se associar um *rank* médio $R_j = \frac{1}{D}\sum_{i=1}^{D} r_i^j$ a este algoritmo.

O teste trabalha com a hipótese nula de que todos os algoritmos possuem desempenhos equivalentes e seus *ranks* médios R_j devem ser iguais. Assim, a estatística do teste de Friedman Q é calculada pela Equação 3.35,

$$Q = \frac{12D}{F(F+1)}\left[\sum_{j=1}^{F} R_j^2 - \frac{F(F+1)^2}{4}\right]. \tag{3.35}$$

Quando D e F são grandes o suficiente ($D \geq 10$ e $F \geq 5$ (DEMŠAR, 2006)), esta estatística segue uma distribuição similar à $\chi^2_{(F-1)}$. Neste caso, o valor-p do teste é dado por $\mathcal{P}(\chi^2_{(F-1)} \geq Q)$. Caso os valores de D e F não sejam suficientemente grandes, a distribuição $\chi^2_{(F-1)}$ não é adequada e o valor-p do teste deve ser obtido de tabelas preparadas para o teste de Friedman (HOLLANDER; WOLFE, 1999).

Caso o valor-p do teste seja inferior a um nível de significância γ, a hipótese nula é rejeitada (os métodos não são considerados estatisticamente equivalentes) e um teste *post-hoc* pode ser feito para comparar os métodos. Uma das possibilidades é a utilização

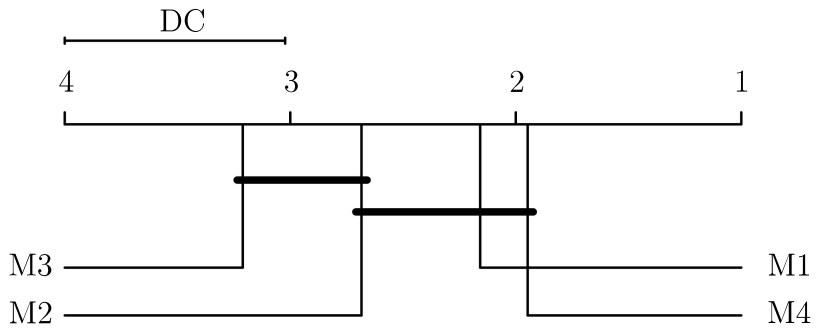
do teste de Nemenyi (NEMENYI, 1963), que compara os métodos dois a dois e avalia se a diferença de seus *ranks* médios é significante, isto é, maior que a distância crítica (DC), dada pela Equação 3.36,

$$DC = q_\gamma \sqrt{\frac{k(k+1)}{6D}}, \tag{3.36}$$

onde q_γ, valor crítico para um nível de significância γ, é tabelado (DEMŠAR, 2006). Neste trabalho, foi utilizado $\gamma = 5\%$ para ambos os testes.

O resultado do teste de Nemenyi pode ser representado por um diagrama (DEMŠAR, 2006), como mostra a Figura 5 para quatro métodos: M1, M2, M3 e M4. A linha superior no diagrama indica a distânca crítica, e logo abaixo está o eixo no qual é indicado o *rank* médio dos métodos, onde os melhores *ranks* estão à direita.

Figura 5 – Representação gráfica do teste de Nemenyi para quadro métodos diferentes



Fonte: Adaptado de (INABA, 2018)

Métodos que não são considerados estatisticamente diferentes (isto é, a distância entre seus *ranks* médios é menor que a distância crítica obtida) são ligados por uma linha. Assim, analisando a Figura 5, os métodos M2 e M3 não são estatisticamente diferentes, mas M1 e M3 são. Isso acontece pois a transitividade não é válida neste teste, assim, M2 foi considerado estatisticamente equivalente a M3 e M1, mas M1 foi considerado estatisticamente superior a M3, pois seu *rank* é melhor e a diferença dos *ranks* das duas técnicas é maior que a distância crítica calculada pelo teste (representada pelo segmento de reta DC).

Outro teste de hipótese foi empregado nos experimentos para avaliar diferenças pareadas entre dois ou mais grupos de métodos que possuem uma característica distinta. Assim, é possível afirmar com mais propriedade sobre certas conclusões. Neste teste de hipótese, chamado de Teste de Wilcoxon, um par de grupos é considerado, e duas variáveis, R^+ e R^-, são calculadas com base nos *ranks* das diferenças d_i das métricas dos grupos

(DEMŠAR, 2006). Tais variáveis são calculadas como mostram as Equações 3.37 e 3.38.

$$R^+ = \sum_{d_i>0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i) \tag{3.37}$$

$$R^- = \sum_{d_i<0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i) \tag{3.38}$$

Com esses valores e o número de amostras D, é possível consultar tabelas de valores críticos para algumas situações. Ou, ainda, pode-se calcular uma estatística z, dada pela Equação 3.39,

$$z = \frac{min(R^+, R^-) - \frac{1}{4}D(D+1))}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}}, \tag{3.39}$$

onde é considerado que z segue uma distribuição normal. Para o nível de significância de 5% considerado neste trabalho, a hipótese nula de que os algoritmos são equivalentes é rejeitada se $z < -1.96$ (DEMŠAR, 2006).

Vale notar que existem versões unicaudais à esquerda e à direita do teste de Wilcoxon. No primeiro caso, a hipótese alternativa é que a distribuição das diferenças entre valores do primeiro e do segundo grupo possui mediana inferior a zero e, portanto, o primeiro grupo possui valores inferiores ao segundo grupo. Já no teste unicaudal à direita, é considerado que a mediana das diferenças é maior que zero, e que os valores do primeiro grupo são superiores ao segundo grupo.

3.4.1.3 Detalhes da Implementação e Ambiente de Execução

Todos os métodos utilizados nos testes deste trabalho foram implementados no software MATLAB 9.4 e executados em um computador com sistema operacional Ubuntu 18.04.3 LTS com processador Intel® Core™ i9-9900KF (8 núcleos a 3.6 GHz) com 64 GB de memória RAM. Nenhum teste foi realizado com a ajuda de placas de processamento gráfico. Em um experimento, a base de dados é dividida entre treino e teste (caso seja necessário), em seguida uma validação cruzada é feita para definição dos hiperparâmetros do método, usando apenas o conjunto de treinamento. O método então é treinado com o melhor conjunto de hiperparâmetros e as métricas obtidas utilizando o conjunto de teste são calculadas. Todos os experimentos foram repetidos 100 vezes com diferentes sementes aleatórias, de forma que 100 números são gerados a partir de uma semente aleatória única, e cada repetição utiliza um desses números como semente aleatória inicial. Todos os códigos e resultados obtidos neste trabalho podem ser consultados em ⟨https://doi.org/10.17632/y5ytz6kydf.1⟩.

Nos experimentos realizados neste capítulo, os métodos considerados foram definidos através da combinação de arquiteturas, algoritmos de treinamento e algoritmos de inicialização de pesos. Tais métodos são apresentados no Quadro 5. Nove métodos foram

usados para treinar SLFN maiores sem usar o princípio de empilhamento, enquanto outros quatorze métodos que empregam o princípio de empilhamento foram considerados.

Quadro 5 – Métodos considerados nos experimentos realizados.

Nome Associado	Arquitetura	Algoritmo de Treinamento	Inicialização de Pesos
R-ELM$_{\text{rfm}}$	SLFN	R-ELM	Pesos Aleatórios
R-ELM$_{\text{ae}}$			Por Autoencoder
R-ELM$_{\text{svd}}$			Pesos Orgotonais
GR-ELM$_{\text{rfm}}$		GR-ELM	Pesos Aleatórios
GR-ELM$_{\text{ae}}$			Por Autoencoder
GR-ELM$_{\text{svd}}$			Pesos Orgotonais
GOR-ELM$_{\text{rfm}}$		GOR-ELM	Pesos Aleatórios
GOR-ELM$_{\text{ae}}$			Por Autoencoder
GOR-ELM$_{\text{svd}}$			Pesos Orgotonais
DSN$_{\text{rfm}}$	DSN	R-ELM	Pesos Aleatórios
DSN$_{\text{ae}}$			Por Autoencoder
DSN$_{\text{svd}}$			Pesos Orgotonais
FDSN$_{\text{rfm}}$			Pesos Aleatórios
FDSN$_{\text{ae}}$			Por Autoencoder
FDSN$_{\text{svd}}$			Pesos Orgotonais
FDSN-GR$_{\text{rfm}}$		GR-ELM	Pesos Aleatórios
FDSN-GR$_{\text{ae}}$			Por Autoencoder
FDSN-GR$_{\text{svd}}$			Pesos Orgotonais
FDSN-GOR$_{\text{rfm}}$		GOR-ELM	Pesos Aleatórios
FDSN-GOR$_{\text{ae}}$			Por Autoencoder
FDSN-GOR$_{\text{svd}}$			Pesos Orgotonais
FKDSN		RK-ELM	-
S-ELM[3]	S-ELM	S-ELM	-

Fonte: Produção do próprio autor.

Vale notar que o método GR-ELM foi proposto para problemas de classificação, bem como o método GOR-ELM para problemas de regressão. Portanto, métodos que envolvem GR-ELM não serão aplicados em testes que envolvem regressão (o mesmo vale para GOR-ELM e classificação). Além disso, vale lembrar que o método S-ELM, embora seja similar ao método proposto, possui limitação para lidar com bases de dados com saída de dimensão única, o que restringe o seu uso.

É possível notar que FDSN-GR (e FDSN-GOR) pode ser visto como uma abordagem intermediária entre a DSN e a FDSN, onde o primeiro não reaproveita nenhum peso entre módulos, e o segundo reaproveita todos. Os métodos GR-ELM e GOR-ELM possuem a capacidade de podar neurônios menos importantes da rede e, portanto, apenas os pesos dos restantes são aproveitados no próximo módulo. A poda realizada pelos algoritmos em um módulo é compensada por novos neurônios no módulo seguinte.

[3] O S-ELM foi usado apenas na Seção 3.4.3.

3.4.1.4 Especificações dos Hiperparâmetros

Todos os métodos utilizados neste trabalho têm pelo menos um hiperparâmetro que deve ser definido previamente ou escolhido através de algum método. Já que as técnicas citadas são algoritmos para treinar redes neurais de uma camada oculta, um dos principais hiperparâmetros a serem definidos é o seu número de neurônios. Deng, Zheng e Chen (2009) notaram que este valor possui um papel importante na complexidade da SLFN e influenciam no *trade-off* entre sobreajuste e subajuste da rede, a não ser que uma regularização seja usada.

Outros hiperparâmetros, como o α dos algoritmos GR-ELM e GOR-ELM, também interferem no desempenho da rede, mas sua função é controlar a esparsidade desta, o que significa que quando seu valor é igual à zero, o treinamento não tem como objetivo utilizar o menor número possível de neurônios, o que acontece quando $\alpha = 1$.

O algoritmo GOR-ELM possui $\rho = \tau = 1$, o R-ELM possui $\alpha = 0$, $\boldsymbol{D} = \boldsymbol{I}$ e $\beta_0 = 0$ (isto é, o UWR-ELM é utilizado). Também são utilizados os critérios de parada para os algoritmos GR-ELM e GOR-ELM de $\epsilon^{abs} = 10^{-3}$ e $\epsilon^{rel} = 10^{-2}$, sugeridos por Inaba et al. (2018) e Silva et al. (2020b), respectivamente. Foi utilizado o *kernel* gaussiano no método FKDSN, como mostra a Equação 3.16, que necessita de um hiperparâmetro γ.

Hiperparâmetros como α (exceto R-ELM), o de regularização e a função de ativação utilizada nas camadas ocultas das redes neurais são escolhidos via validação cruzada k-fold, um *grid* pré-definido e o conjunto de treinamento, sem a presença de dados do conjunto separado para teste, como feito por Inaba et al. (2018), Silva et al. (2020b), Deng, Zheng e Chen (2009), Zhang e Luo (2015) e outros. O hiperparâmetro α é escolhido do conjunto $\{0; 0,25; 0,5; 0,75; 1\}$, o de regularização C (λ para o GOR-ELM) do conjunto $\{2^{-20}; 2^{-19}; \ldots; 2^{20}\}$ (SILVA et al., 2020b), e a função de ativação é escolhida entre as funções propostas por Huang et al. (2012): funções sigmoide, *hard-limit*, gaussiana e multiquádrica, definidas pelas Equações 3.40, 3.41, 3.42 e 3.43, respectivamente (vale notar que essas funções são adaptadas para serem aplicadas a todos os elementos de um vetor resultante da multiplicação dos pesos de entrada por uma amostra $\boldsymbol{x}$). O hiperparâmetro γ também foi escolhido do *grid* $\{2^{-20}; 2^{-19}; \ldots; 2^{20}\}$.

$$h(x) = \frac{1}{1 + e^{-x}} \tag{3.40}$$

$$h(x) = \begin{cases} 1, & \text{se } x \geq 0 \\ 0, & \text{caso contrário} \end{cases} \tag{3.41}$$

$$h(x) = e^{(-x^2)} \tag{3.42}$$

$$h(x) = \sqrt{1 + x^2} \tag{3.43}$$

Exceto quando explicitado, os métodos R-ELM, GR-ELM e GOR-ELM utilizados neste trabalho têm valor inicial de $L = 1000$. Como a ideia do princípio de empilhamento (WOLPERT, 1992) é utilizar módulos menores para obter desempenhos similares a modelos mais complexos, os métodos DSN e FDSN foram treinados com 10 módulos de 100 neurônios cada, para uma comparação mais justa com redes com a mesma quantidade total de neurônios. Todos os módulos destes métodos também são treinados por algoritmos idênticos (no caso, R-ELM com o mesmo hiperparâmetro de regularização C, obtido na etapa de validação cruzada).

3.4.2 Análise Considerando o Tamanho da Base da Dados

Com o objetivo de analisar o desempenho dos métodos FDSN e FKDSN, proposto neste trabalho com relação ao número de amostras de uma base de dados, foram utilizadas as bases de dados propostas por Fukunaga (1990). Este autor propõe 4 bases de dados diferentes e artificiais de problemas de classificação binária (2 classes) que são compostas de características amostradas de gaussianas multidimensionais.

Os dados de uma das classes são amostrados de gaussianas com média igual a zero e desvio padrão igual a 1, para todas as características. Já as amostras da segunda classe possuem características amostradas de distribuições gaussianas com parâmetros diferentes. Com esses valores de parâmetros, é possível detectar uma interseção espacial entre as gaussianas das duas classes e medir um erro teórico esperado para as técnicas que estão analisando esses dados.

As quatro bases possuem dimensão 8, duas classes e distribuições normais cujas médias das duas classes $\boldsymbol{M}_1$ e $\boldsymbol{M}_2$ seguem as Equações 3.44 e 3.45. Já as matrizes de covariância das distribuições seguem as Equações 3.46 e 3.47.

$$\boldsymbol{M}_1 = [0, \ldots, 0]^\mathsf{T} \tag{3.44}$$

$$\boldsymbol{M}_2 = [m_1, \ldots, m_8]^\mathsf{T} \tag{3.45}$$

$$\boldsymbol{\Sigma}_1 = \boldsymbol{I} \tag{3.46}$$

$$\boldsymbol{\Sigma_2} = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \lambda_8 \end{bmatrix} \tag{3.47}$$

As quatro bases de dados propostas por Fukunaga (1990) possuem distribuições distintas para as suas respectivas segundas classes. As características dessas distribuições são apresentadas na Tabela 3. Vale notar que as acurácias esperadas para as bases de dados I-I, I-4I, I-Λ e I-I-null são de 90%, 91%, 98,1% e 50%, respectivamente. A baixa acurácia esperada da base I-I-null se deve ao fato de que as duas classes são amostradas da mesma distribuição.

Tabela 3 – Características das bases de dados propostas por Fukunaga (1990).

Nome	Caract. das Médias	Caract. das Variâncias
I-I	$m_1 = 2{,}56$, $m_2 = \ldots = m_8 = 0$	$\lambda_1 = \ldots = \lambda_8 = 1$
I-4I	$m_1 = \ldots = m_8 = 0$	$\lambda_1 = \ldots = \lambda_8 = 4$
I-Λ	[3,86 3,1 0,84 0,84 1,64 1,08 0,26 0,01]	[8,41 12,06 0,12 0,22 1,49 1,77 0,35 2,73]
I-I-null	$m_1 = \ldots = m_8 = 0$	$\lambda_1 = \ldots = \lambda_8 = 1$

Fonte: Adaptado de (FUKUNAGA, 1990)

A metodologia para testar o comportamento de um algoritmo com relação à quantidade de amostras proposta por Fukunaga (1990) envolve a (I) geração das bases de dados com N_i amostras por classe, (II) divisão dos dados em amostras de treino e teste, (III) execução do "experimento planejado", (IV) repetição dos 3 primeiros passos por $\tau = 10$ vezes, onde em cada execução os dados são gerados independentemente. O resultado apresentado é a média das métricas obtidas nas 10 execuções do experimento.

O "experimento planejado" consiste em uma validação cruzada 5-fold usando o conjunto de treino para escolha dos hiperparâmetros dos métodos seguido de 100 treinos e testes distintos dos métodos (usando sementes pseudoaleatórias distintas), onde cada treino e teste usa o seu respectivo conjunto. Com as classes estimadas, é possível obter a acurácia média da técnica.

Os métodos testados nesse experimento foram os seguintes: FDSN treinada com R-ELM usando inicialização aleatória de pesos (chamado de FDSN_{rfm}); FDSN treinada com R-ELM usando o método *autoencoder* (FDSN_{ae}); FDSN treinado com R-ELM usando pesos ortogonais (FDSN_{svd}); e FKDSN. O número de neurônios ocultos é de 100 neurônios por módulo (no caso do FKDSN, o número de neurônios ocultos é o menor número entre o dobro do número de amostras de treinamento por classe e 100), e o número máximo de módulos é de 10.

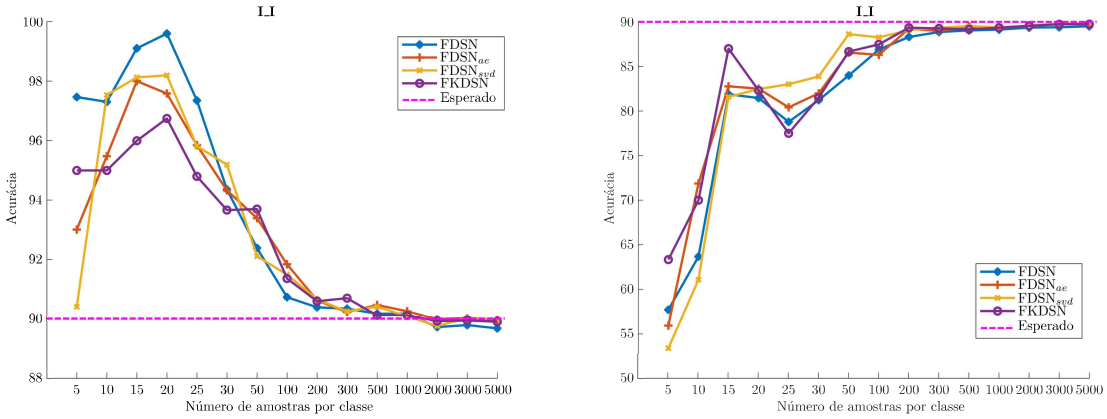
As acurácias obtidas para cada número de amostras por classe para as bases I-I, I-4I, I-Λ e I-I-NULL podem ser vistos nas Figuras 6, 7, 8 e 9, respectivamente. Nestas figuras, é possível ver que as acurácias de treino e teste possuem uma oscilação quando o número de amostras por classe é pequeno, principalmente quando há menos de 50 amostras por classe, o que totaliza 100 amostras na base de dados.

Embora seja possível associar esse número de amostras com o número de neurônios por módulo (isto é, um sobreajuste causado por ter mais neurônios do que o total de amostras), pode-se notar que a técnica FKDSN, ao lidar com menos que 100 amostras, utiliza "número de neurônios" (número de "vetores suporte") igual ao número de amostras. Mesmo assim, houve uma grande oscilação nas acurácias para amostras pequenas, acompanhando as demais técnicas, o que pode indicar que essas bases de dados não são muito bem representadas por poucas amostras.

Como notado, as técnicas obtiveram altas acurácias de treino e baixas acurácias de teste com poucas amostras, o que pode indicar um sobreajuste nos casos citados. Com o aumento do número de amostras, todas as técnicas testadas se aproximam dos erros teóricos esperados tanto nos dados de treino quanto nos dados de teste.

Figura 6 – Acurácia de treino e teste na base de dados Fukunaga I-I de acordo com o número de amostras por classe

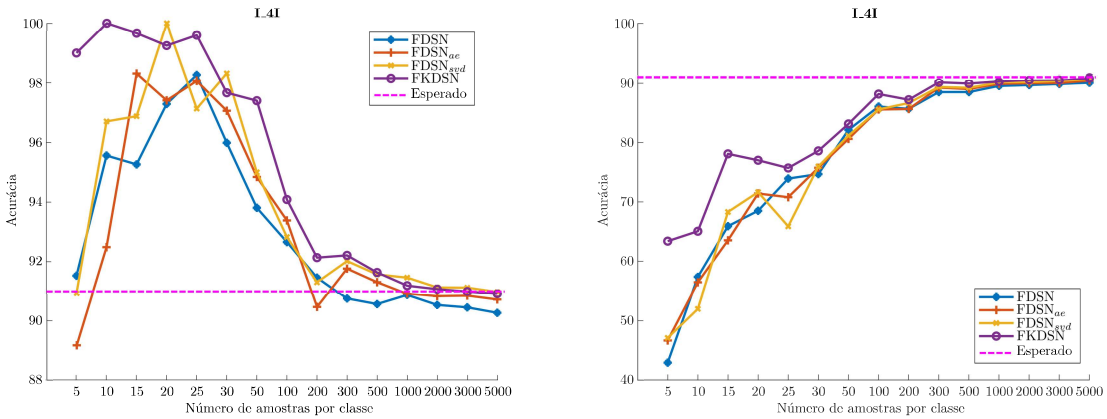
(a) Acurácia de Treino (b) Acurácia de Teste



Fonte: Produção do próprio autor

Figura 7 – Acurácia de treino e teste na base de dados Fukunaga I-4I de acordo com o número de amostras por classe

(a) Acurácia de Treino (b) Acurácia de Teste



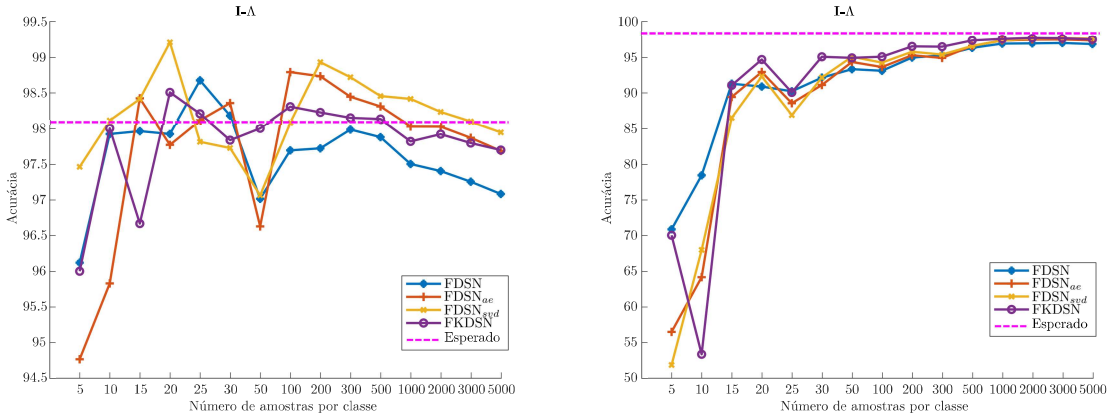
Fonte: Produção do próprio autor

3.4.3 Análise Considerando Diferentes Arquiteturas

Com o objetivo de testar a influência do número de módulos no desempenho da rede FDSN, foram utilizadas as bases de dados apresentadas na Tabela 4. Essas bases de dados são públicas e estão disponíveis em (DUA; GRAFF, 2017). Tais bases são de problemas de classificação multi-classe e a origem de seus dados são informações retiradas

Figura 8 – Acurácia de treino e teste na base de dados Fukunaga I-Λ de acordo com o número de amostras por classe

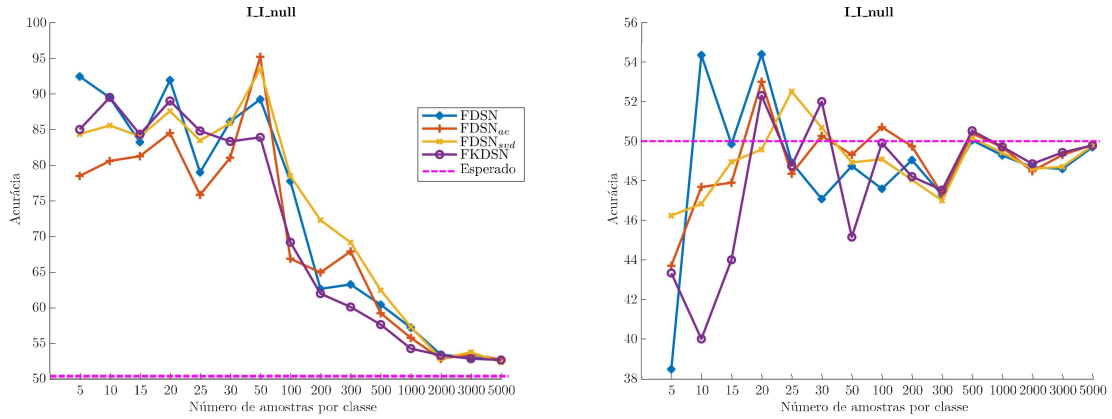
(a) Acurácia de Treino (b) Acurácia de Teste



Fonte: Produção do próprio autor

Figura 9 – Acurácia de treino e teste na base de dados Fukunaga I-I-NULL de acordo com o número de amostras por classe

(a) Acurácia de Treino (b) Acurácia de Teste



Fonte: Produção do próprio autor

de imagens, como médias dos valores de pixel ou então características como coordenadas do centroide de um componente conectado presente em uma janela.

A base de dados Segment foi escolhida para analisar o efeito da acurácia do modelo com a inserção de novos módulos (com número de neurônios $L = 100$). Os métodos escolhidos para comparação foram: R-ELM com 100 neurônios ocultos (RELM-100); R-ELM com 3000 neurônios ocultos (RELM-3000), DSN treinada com R-ELM (DSN), FDSN treinada com R-ELM (FDSN) e S-ELM com 100 neurônios por módulo (e até 30 módulos). As bases já são divididas entre treino e teste, e um k-fold foi utilizado para obter o hiperparâmetro de regularização C entre os possíveis valores $[2^{-10}, 2^{-9}, \cdots, 2^{9}, 2^{10}]$. Já o método S-ELM reduz a dimensionalidade entre módulos em 50% (isto é, 50 dos 100

Tabela 4 – Bases de dados utilizadas no experimento que avalia a arquitetura da FDSN. As colunas indicam o número de amostras de treino e de teste, número de características (d) e número de classes (m).

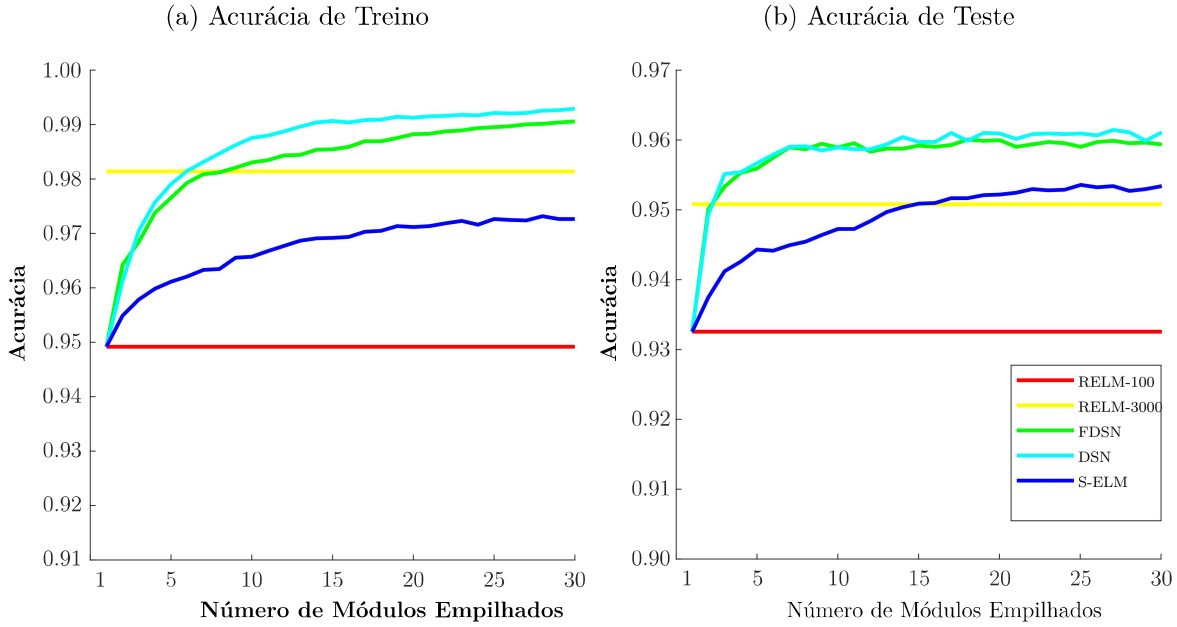
Dataset	#Dados Treino	#Dados Teste	#Caract.	#Clas.
SatImage	4435	2000	36	6
Segment	1155	1155	19	7

Fonte: Produção do próprio autor.

neurônios são "aproveitados" pelo próximo módulo).

Este experimento foi repetido 100 vezes, e as médias das acurácias obtidas neste teste são apresentadas na Figura 10, que mostra que nessa base de dados específica, a inserção de novos módulos foi capaz de melhorar a acurácia dos métodos empilhados, podendo até superar o modelo com 3000 neurônios. Isso pode indicar que usar métodos empilhados pode ser uma boa abordagem para obter boas métricas em sistemas com limitações de memória. As Figuras 10a e 10b mostram que os métodos DSN e FDSN atingiram um estágio de estabilidade quanto a acurácia de teste (adicionar mais módulos não implica em um aumento significativo da métrica), enquanto que o método S-ELM aparenta ter possibilidade de melhora, embora seu resultado seja inferior aos dois anteriores.

Figura 10 – A variação da acurácia dos métodos empilhados ao incluir novos módulos, considerando a base Segment

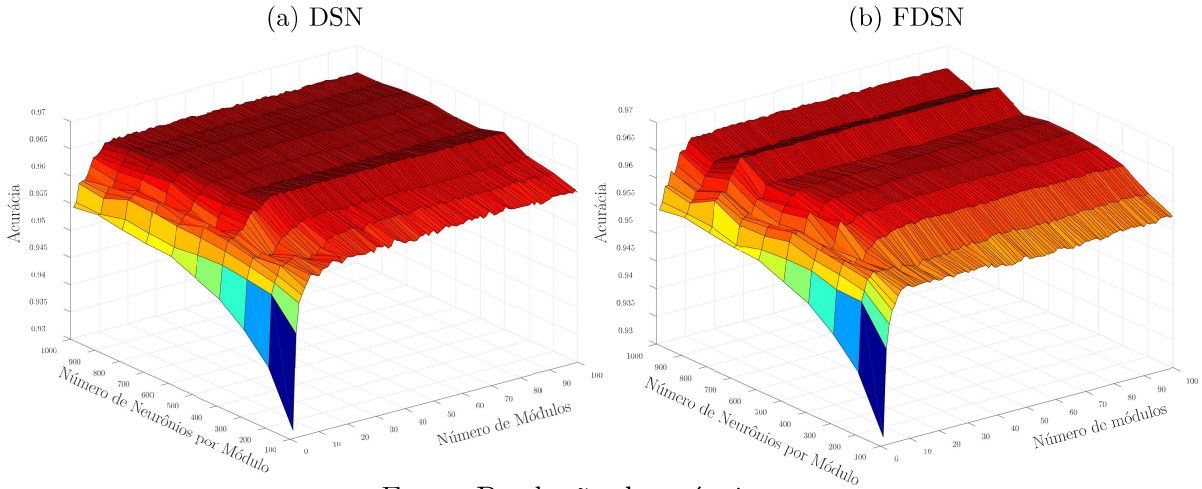


Fonte: Produção do próprio autor

Um segundo experimento foi feito com o objetivo de analisar o compromisso entre o número de neurônios L por módulo e o número máximo de módulos S. Para isso, foi escolhida a base de dados SatImage, e o experimento anterior foi repetido com vários

modelos DSN e FDSN considerando de 1 até 100 módulos, onde cada módulo podia possuir um valor de L entre $[100, 200, \ldots, 1000]$. É possível ver na Figura 11 que, como esperado, incluir novos módulos ou utilizar módulos com mais neurônios tende a aumentar a métrica de acurácia na base de dados testada, até que a métrica atinja uma estabilidade. Assim, a escolha destes valores deve levar em consideração limitações de memória do ambiente em que elas foram aplicadas e quanto isso terá impacto na métrica do problema a ser tratado. Em ambos os casos, é possível ver, após a métrica atingir estabilidade, que há uma combinação (ou combinações) de número de neurônios e número de módulos que atinge uma acurácia maior que as demais, embora estas sejam muito próximas.

Figura 11 – O efeito da escolha de S e L na Acurácia de modelos DSN e FDSN na base de dados SatImage



Fonte: Produção do próprio autor

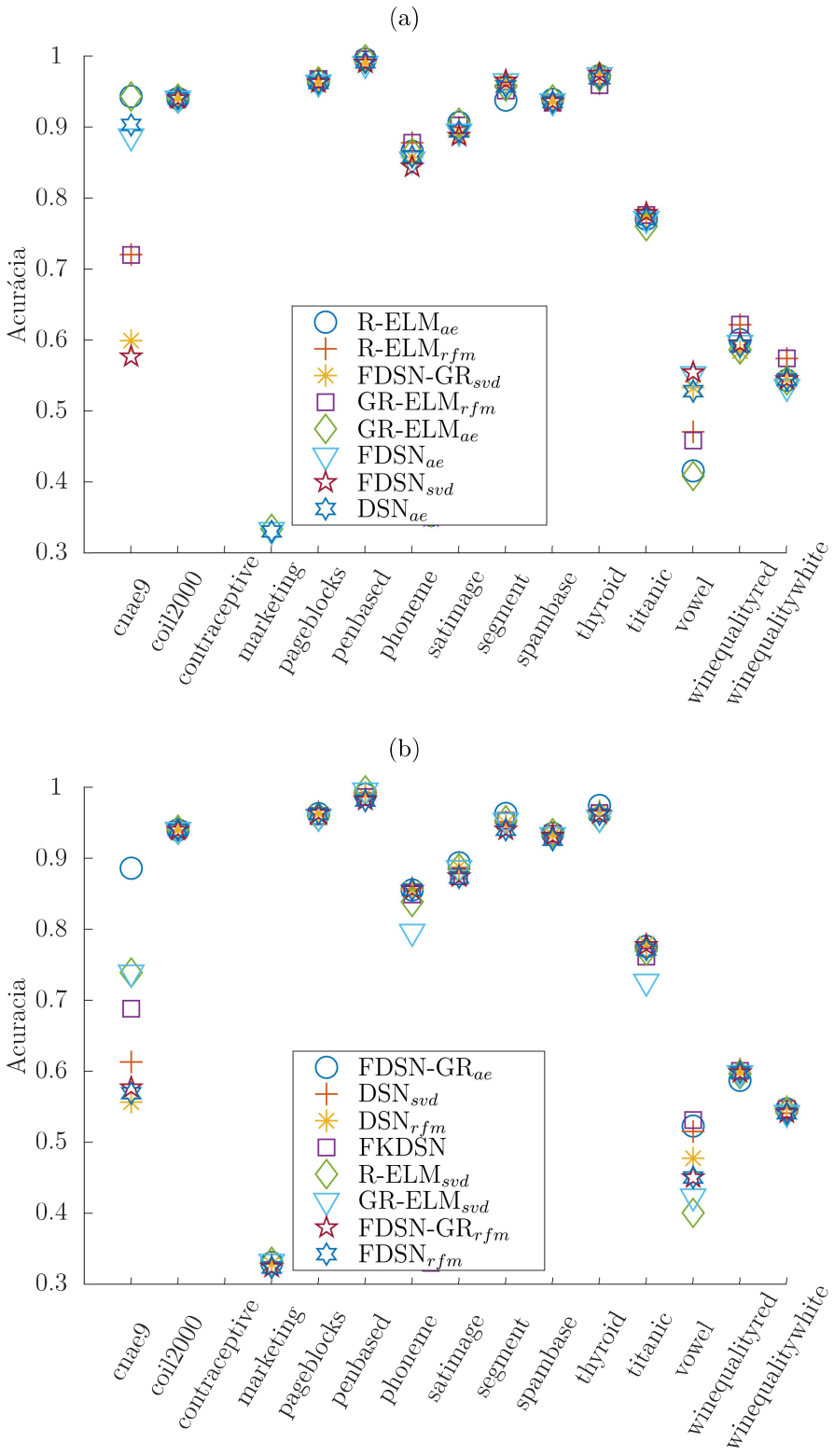
3.4.4 Análise por Teste de Hipótese

Nesta seção são apresentados os experimentos e resultados feitos com o objetivo de comparar métricas como acurácia (ou erro em casos de regressão), tempo de treinamento e memória utilizada para armazenar os métodos. Resultados envolvendo bases de dados de classificação e regressão são apresentados nas Seções 3.4.4.1 e 3.4.4.2, respectivamente.

3.4.4.1 Resultados em Bases de Dados de Classificação

Os valores de acurácia e hiperparâmetros encontrados nos experimentos realizados podem ser vistos no Apêndice A, enquanto os gráficos das acurácias médias obtidas para cada base de dados podem ser vistos na Figura 12. A Tabela 5 mostra os *ranks* médios obtidos por cada método referentes às acurácias obtidas, assim como as médias dessa métrica. É possível notar que uma métrica média maior não implica em um melhor *rank*, já que foram obtidos *outliers* que deterioraram a métrica média, mas não tanto o *rank*. Estes pontos foram possivelmente causados por sobreajuste dos modelos (*overfitting*).

Figura 12 – Acurácias obtidas nos problemas de classificação. Em (a), os métodos com os oito melhores *ranks*. Os demais estão em (b).



Fonte: Produção do próprio autor.

Tabela 5 – *Ranks* médios ($\overline{\text{Rank}}$) e métrica acurácia no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 1. Os métodos estão ordenados pelo *rank* médio, de forma crescente (do melhor para o pior).

Método	$\overline{\text{Rank}}$	Acurácia
R-ELM$_{\text{ae}}$	5,5667	0,78 (0,22)
R-ELM$_{\text{rfm}}$	6,2333	0,77 (0,21)
FDSN-GR$_{\text{svd}}$	6,7000	0,76 (0,21)
GR-ELM$_{\text{rfm}}$	6,8000	0,77 (0,21)
GR-ELM$_{\text{ae}}$	7,2667	0,78 (0,22)
FDSN$_{\text{ae}}$	7,2667	0,78 (0,20)
FDSN$_{\text{svd}}$	7,3000	0,76 (0,21)
DSN$_{\text{ae}}$	7,9000	0,78 (0,21)
FDSN-GR$_{\text{ae}}$	7,9000	0,78 (0,21)
DSN$_{\text{svd}}$	8,1000	0,76 (0,21)
DSN$_{\text{rfm}}$	9,5000	0,76 (0,21)
FKDSN	9,5667	0,77 (0,20)
R-ELM$_{\text{svd}}$	10,4000	0,76 (0,21)
GR-ELM$_{\text{svd}}$	11,2333	0,76 (0,21)
FDSN-GR$_{\text{rfm}}$	11,5667	0,75 (0,21)
FDSN$_{\text{rfm}}$	12,7000	0,75 (0,21)

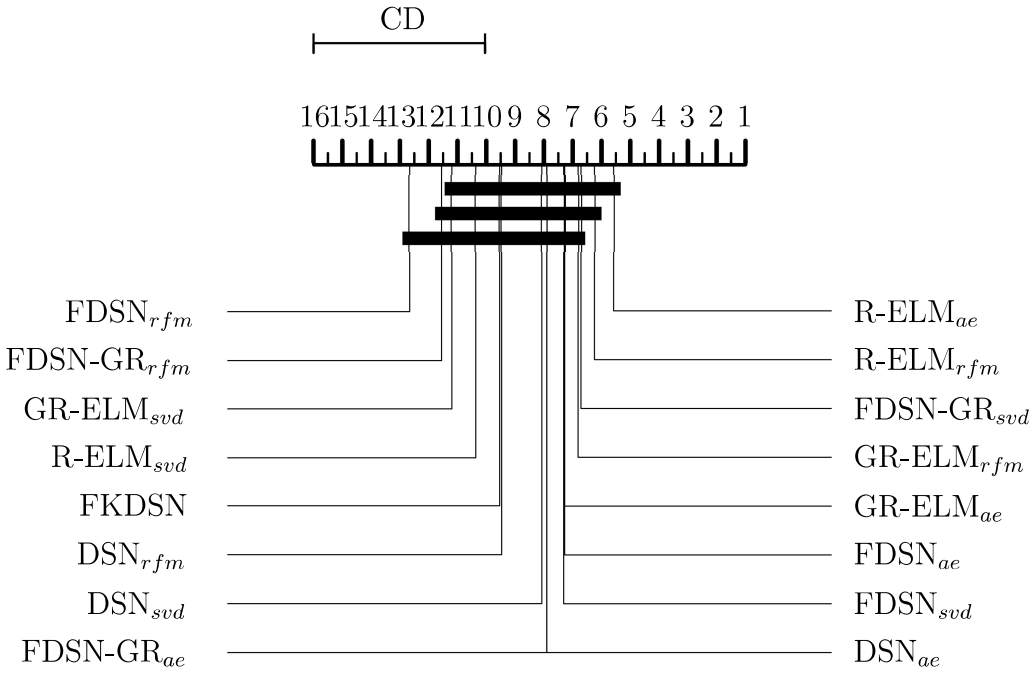
Fonte: Produção do próprio autor.

Os resultados apresentados na Tabela 5 mostram que os métodos R-ELM$_{\text{ae}}$ e R-ELM$_{\text{rfm}}$ atingiram os melhores *ranks*, seguidos pelo método FDSN-GR$_{\text{svd}}$. O método FDSN$_{\text{ae}}$ atingiu o sexto melhor *rank* para problemas de classificação, enquanto o FDSN$_{\text{rfm}}$ obteve o pior. Estes resultados indicam que os métodos baseados em FDSN, que empilham módulos pequenos, podem atingir acurácias médias próximas as acurácias dos métodos baseados em R-ELM[4], que treinam redes maiores, desde que um método de inicialização de pesos adequado seja utilizado. O método FKDSN obteve *ranks* intermediários e ainda foi considerado equivalente aos demais métodos, com exceção do FDSN$_{\text{rfm}}$ e FDSN-GR$_{\text{rfm}}$.

O teste estatístico de Friedman (DEMŠAR, 2006) foi aplicado nos resultados obtidos, considerando uma hipótese de que os métodos são equivalentes, usando um nível de significância de 5%. O teste obteve um valor-$p = 0{,}0000934$, rejeitando a hipótese nula. Assim, o teste de Nemenyi foi utilizado, cujo resultado pode ser visto na Figura 13. A distância crítica, neste caso, foi de $5,995$, ou seja, métodos cuja diferença entre *ranks* é maior que este valor não são equivalentes. O teste indica que o método FDSN$_{\text{rfm}}$ não é estatisticamente similar aos métodos com melhores *ranks*. Porém, quando combinado a uma técnica de inicialização, ele se torna similar aos métodos que treinam redes maiores, como os baseados em R-ELM e GR-ELM. Pode-se observar que as acurácias médias do método com melhor *rank* e do método FDSN$_{\text{svd}}$ são iguais, considerando 2 algarismos significativos após a vírgula.

[4] A expressão "métodos baseados em R-ELM" indica referência aos métodos R-ELM$_{\text{rfm}}$, R-ELM$_{\text{ae}}$ e R-ELM$_{\text{svd}}$. O mesmo é válido para GR-ELM, GOR-ELM, DSN, FDSN, FDSN-GR e FDSN-GOR.

Figura 13 – Teste de Nemenyi utilizando a métrica Acurácia para os experimentos com problemas de classificação.



Fonte: Produção do próprio autor.

Com a finalidade de comparar o desempenho entre métodos de inicialização de pesos, as acurácias obtidas neste experimento foram divididas em 3 grupos, chamados de “rfm”, “ae” e “svd”, onde cada grupo contém apenas os resultados dos métodos que utilizam a respectiva técnica de inicialização de pesos. Os 3 grupos foram comparados utilizando o teste estatístico de Wilcoxon unicaudal à direita. Ou seja, se a hipótese nula for rejeitada ao considerar qualquer par de grupos, a mediana das diferenças dos resultados é considerada maior que zero, e as acurácias do primeiro grupo são superiores. Ao comparar os três grupos par a par, a hipótese nula foi rejeitada ao comparar o grupo “ae” com os grupos “rfm” e “svd”, com valores de estatística de 0,0104 e 0,0081, respectivamente. Isso significa que o grupo dos métodos que utilizam a inicialização por *autoencoder* foi considerado estatisticamente superior aos demais, quando a acurácia em problemas de classificação foi considerada.

O mesmo teste foi repetido, porém dividindo os métodos em 4 grupos, com a finalidade de avaliar os algoritmos de treinamento utilizados. Tais grupos foram chamados de “R-ELM”, “GR-ELM”, “DSN” e “FDSN”. Neste caso, a hipótese nula foi rejeitada apenas ao comparar o grupo “R-ELM” com os grupos “GR-ELM” e “FDSN”, com valores de estatística de 0,0062 e 0,0432, respectivamente. Isso significa que, para os problemas de classificação analisados, R-ELM foi considerado estatisticamente superior aos métodos GR-ELM e FDSN quando considerando a métrica acurácia.

O tempo gasto para treinar cada modelo também foi analisado, onde os valores encontrados são apresentados no Apêndice A. Os *ranks* médios foram calculados e são apresentados na Tabela 6, onde os métodos baseados em FDSN obtiveram os melhores *ranks*. O teste de Friedman foi aplicado, e a hipótese nula de que os métodos são equivalentes foi rejeitada com valor-$p = 9{,}2 \times 10^{-28}$. Com isso, o teste de Nemenyi foi feito e a distância crítica de $5,9559$ é mostrada na Figura 14, que indica que os métodos baseados em FDSN e o FKDSN são estatisticamente superiores aos métodos que treinam redes maiores, como os baseados em R-ELM e GR-ELM. Vale notar que o DSN com inicialização ortogonal de pesos foi considerado estatisticamente equivalente aos métodos FDSN.

Analisando os valores de tempo de treinamento, os métodos baseados em FDSN levaram um valor médio de aproximadamente 50% do tempo gasto pelos métodos baseados em DSN e 33% dos métodos baseados em R-ELM. Destaca-se o impacto no tempo de treinamento que a inicialização utilizando *autoencoder* causa, já que em todas as 5 técnicas consideradas, o tempo de treinamento utilizando essa inicialização foi maior do que utilizando inicialização ortogonal ou aleatória.

Tabela 6 – *Ranks* médios ($\overline{\text{Rank}}$) e tempo de treinamento no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 1. Os métodos estão ordenados pelo *rank* médio, de forma crescente.

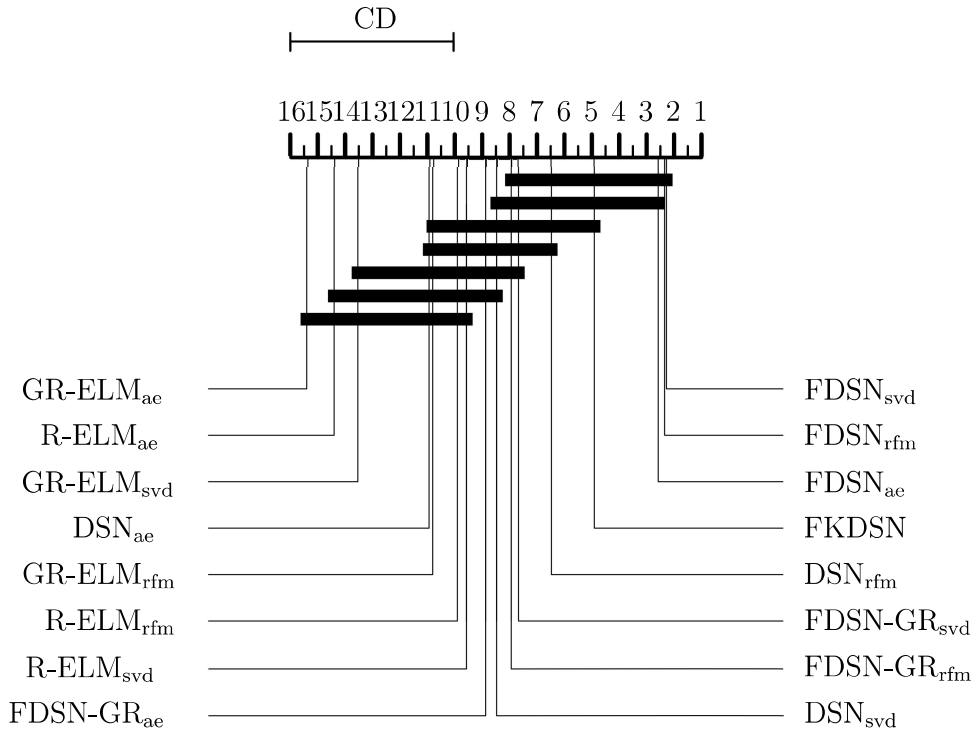
Método	$\overline{\text{Rank}}$	Tempo de treinamento (ms)
FDSN_{svd}	2,2667	20,70 (10,90)
FDSN_{rfm}	2,3333	21,80 (13,50)
FDSN_{ae}	2,5667	20,80 (13,90)
FKDSN	4,9000	30,50 (19,70)
DSN_{rfm}	6,4667	35,60 (21,50)
$\text{FDSN-GR}_{\text{svd}}$	7,6667	38,50 (15,90)
$\text{FDSN-GR}_{\text{rfm}}$	7,9333	40,40 (18,10)
DSN_{svd}	8,4667	40,20 (21,00)
$\text{FDSN-GR}_{\text{ae}}$	8,8667	42,60 (17,30)
$\text{R-ELM}_{\text{svd}}$	9,5667	54,90 (39,00)
$\text{R-ELM}_{\text{rfm}}$	9,9000	55,40 (40,00)
$\text{GR-ELM}_{\text{rfm}}$	10,8000	62,10 (45,60)
DSN_{ae}	10,9333	55,40 (33,40)
$\text{GR-ELM}_{\text{svd}}$	13,5333	128,30 (157,60)
R-ELM_{ae}	14,4000	110,60 (74,90)
$\text{GR-ELM}_{\text{ae}}$	15,4000	311,20 (353,00)

Fonte: Produção do próprio autor.

Uma análise similar foi feita com o tamanho do modelo (número de bytes que ele necessita para ser armazenado). Os métodos baseados em FDSN obtiveram os melhores *ranks*, como pode ser visto na Tabela 7[5]. O teste de Friedman foi aplicado, e a hipótese

[5] O número apresentado de bytes se refere a implementação realizada neste trabalho em MATLAB, sem muita preocupação com otimização de código e memória. Valores diferentes podem ser obtidos em

Figura 14 – Teste de Nemenyi utilizando o tempo de treinamento para os experimentos com problemas de classificação.



Fonte: Produção do próprio autor.

nula de que os métodos são equivalentes foi rejeitada. Com isso, o teste de Nemenyi foi feito e a distância crítica de $5,9559$ é mostrada na Figura 15, que indica que os métodos baseados em FDSN, embora tenham obtido *rank* muito inferior às demais técnicas, se mostraram estatisticamente equivalente às demais técnicas, exceto às baseadas em DSN e as técnicas FDSN-GR$_{ae}$ e FDSN-GR$_{rfm}$.

Embora o teste de Nemenyi tenha indicado que os métodos baseados em FDSN sejam estatisticamente equivalentes a maioria das demais técnicas testadas, a quantidade de memória necessária para armazenar a rede treinada foi de aproximadamente 24% e 26% do que foi utilizada para os métodos baseados em R-ELM e GR-ELM, respectivamente. Vale notar que o método GR-ELM inicia com uma rede completa e possui a capacidade de podar neurônios se julgar necessário. Já os métodos baseados em DSN, que não foram considerados equivalentes às demais, necessitam de seis vezes mais bytes para ser armazenada que os métodos baseados em FDSN, em média. Isto mostra que ele é mais adequado para ser utilizado em sistemas que possuem pouca memória disponível. Os métodos FDSN-GR possuem a característica de aproveitar pesos do FDSN, mas também podem podar neurônios para substituí-los nos próximos módulos, o que faz com que ele

outras implementações. Porém, como apresentado na Seção 3.3.3, o uso de memória do FDSN tende a ser menor que das demais técnicas.

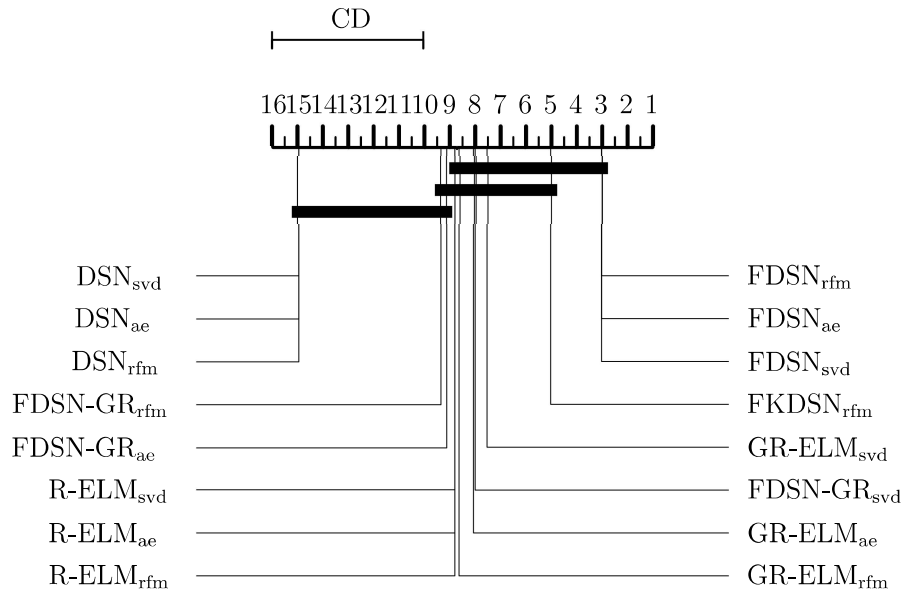
necessite de mais uso de memória. Caso a poda seja relevante, o uso de memória dele irá se aproximar do método DSN, o que explicaria o *rank* médio superior ao das técnicas FDSN.

Tabela 7 – *Ranks* médios ($\overline{\text{Rank}}$) e tamanho do modelo em kilobytes no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 1. Os métodos estão ordenados pelo *rank* médio, de forma crescente.

Método	$\overline{\text{Rank}}$	Tamanho do modelo (kilobytes)
FDSN_{rfm}	3,0000	164,55 (177,89)
FDSN_{ae}	3,0000	164,55 (177,89)
FDSN_{svd}	3,0000	164,55 (177,89)
FKDSN	5,0000	164,97 (177,89)
$\text{GR-ELM}_{\text{svd}}$	7,5000	599,95 (1650,73)
$\text{FDSN-GR}_{\text{svd}}$	7,9667	208,57 (177,22)
$\text{GR-ELM}_{\text{ae}}$	8,0333	623,88 (1645,70)
$\text{GR-ELM}_{\text{rfm}}$	8,6000	636,31 (1644,96)
$\text{R-ELM}_{\text{rfm}}$	8,8000	663,12 (1638,60)
R-ELM_{ae}	8,8000	663,12 (1638,60)
$\text{R-ELM}_{\text{svd}}$	8,8000	663,12 (1638,60)
$\text{FDSN-GR}_{\text{ae}}$	9,1333	240,48 (201,99)
$\text{FDSN-GR}_{\text{rfm}}$	9,3667	216,40 (180,10)
DSN_{rfm}	15,0000	889,06 (1663,26)
DSN_{ae}	15,0000	889,06 (1663,26)
DSN_{svd}	15,0000	889,06 (1663,26)

Fonte: Produção do próprio autor.

Figura 15 – Teste de Nemenyi utilizando o tamanho do modelo em bytes para os experimentos com problemas de classificação.



Fonte: Produção do próprio autor.

3.4.4.2 Resultados em Bases de Dados de Regressão

Os valores de aRRMSE e hiperparâmetros encontrados nos experimentos realizados podem ser vistos no Apêndice B, enquanto os gráficos das aRRMSE médias obtidas para cada base de dados podem ser vistos na Figura 16, para as oito melhores técnicas, e na Figura 17, para as demais, onde valores fora do intervalo apresentado foram ocultados para uma melhor visualização. A Tabela 8 mostra os *ranks* médios obtidos por cada método referente à métrica aRRMSE, assim como os valores médios desta métrica. É possível notar que uma métrica média menor não implica em um melhor *rank*, já que foram obtidos *outliers* que deterioraram consideravelmente a métrica média, mas não tanto o *rank*. Isto pode ser visto nas técnicas DSN_{ae}, DSN_{svd}, FDSN_{svd} e FDSN_{ae}, que obtiveram *ranks* intermediários, mas valores muito elevados de aRRMSE médio. Estes valores foram possivelmente causados por modelos sobreajustados aos dados.

Tabela 8 – *Ranks* médios ($\overline{\text{Rank}}$) e métrica aRRMSE no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 2. Os métodos estão ordenados pelo *rank* médio, de forma crescente (do melhor para o pior).

Método	$\overline{\text{Rank}}$	$\overline{\text{aRRMSE}}$	Mediana aRRMSE
DSN_{ae}	5,8293	443,67 (2803,57)	0,4022
DSN_{svd}	5,9390	22206,65 (140444,67)	0,3347
R-ELM_{ae}	6,1341	0,93 (3,44)	0,3415
FDSN_{svd}	6,8171	160,60 (1013,26)	0,3799
FKDSN	7,0488	0,39 (0,28)	0,3423
$\text{R-ELM}_{\text{svd}}$	7,0610	1,22 (4,62)	0,3513
$\text{R-ELM}_{\text{rfm}}$	7,1463	0,61 (1,33)	0,3564
$\text{FDSN-GOR}_{\text{svd}}$	7,8293	0,51 (0,79)	0,4055
FDSN_{ae}	8,6829	52,61 (329,81)	0,4490
$\text{GOR-ELM}_{\text{rfm}}$	8,7805	0,56 (1,01)	0,3626
$\text{GOR-ELM}_{\text{svd}}$	9,7317	0,43 (0,35)	0,3941
DSN_{rfm}	10,3659	0,85 (2,80)	0,4210
$\text{GOR-ELM}_{\text{ae}}$	10,5732	0,43 (0,32)	0,4032
$\text{FDSN-GOR}_{\text{ae}}$	10,7683	0,64 (1,31)	0,4374
FDSN_{rfm}	10,9878	0,92 (3,19)	0,4160
$\text{FDSN-GOR}_{\text{rfm}}$	12,3049	0,48 (0,45)	0,4156

Fonte: Produção do próprio autor.

As medianas obtidas nos testes para os métodos também são apresentadas na Tabela 8. Embora as médias das aRRMSE obtidas pelos métodos variem bastante, as medianas são valores similares, onde a maior mediana dos erros é do método FDSN_{ae}, que é cerca de 30% maior que a menor mediana, obtida pelo método DSN_{svd}.

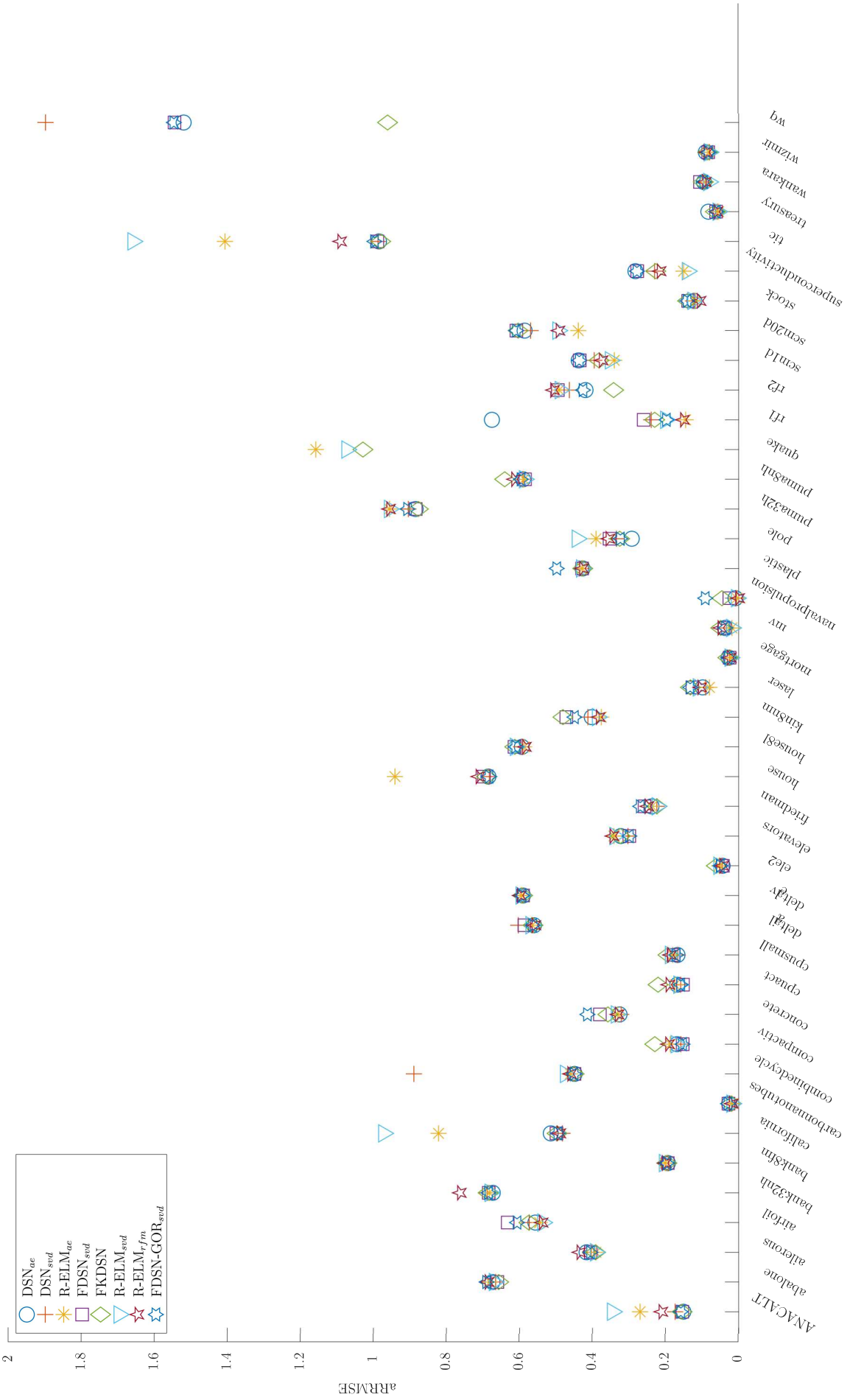
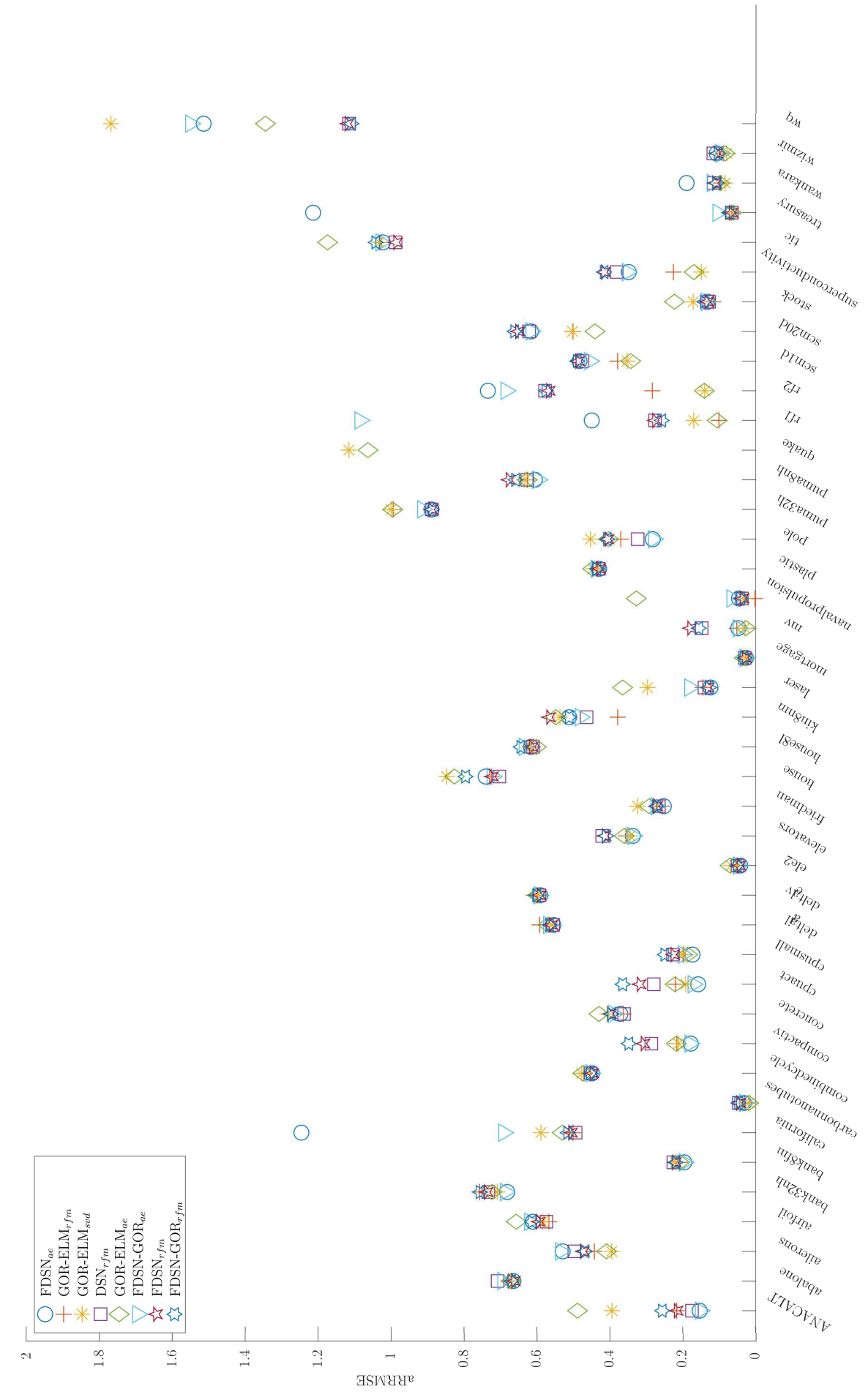


Figura 16 – Gráfico das métricas aRRMSE das técnicas com os oito melhores *ranks* obtidos nos problemas de regressão.

Fonte: Produção do próprio autor.



Fonte: Produção do próprio autor.

Figura 17 – Gráfico das métricas aRRMSE das técnicas com os oito piores *ranks* obtidos nos problemas de regressão.

Os resultados apresentados na Tabela 8 mostram que os métodos DSN_{ae} e DSN_{svd} atingiram os melhores *ranks*, seguido pelo método R-ELM_{ae}. O método FDSN_{svd} atingiu o quarto melhor *rank* para problemas de regressão, enquanto o FDSN_{rfm} obteve o segundo pior. Estes resultados indicam que métodos empilhados podem ser mais adequados para utilização em problemas de regressão, atingindo *ranks* até melhores que métodos baseados em R-ELM[6], que treinam redes com um grande número de neurônios, principalmente quando utilizados em conjunto com um método adequado de inicialização de pesos. Isto pode ser visto na técnica FDSN, que reaproveita seus pesos entre os módulos, mas essa melhora não se limita a essa técnica. De fato, todos os métodos analisados obtiveram *ranks* melhores quando combinados com inicialização de pesos, quando aplicável.

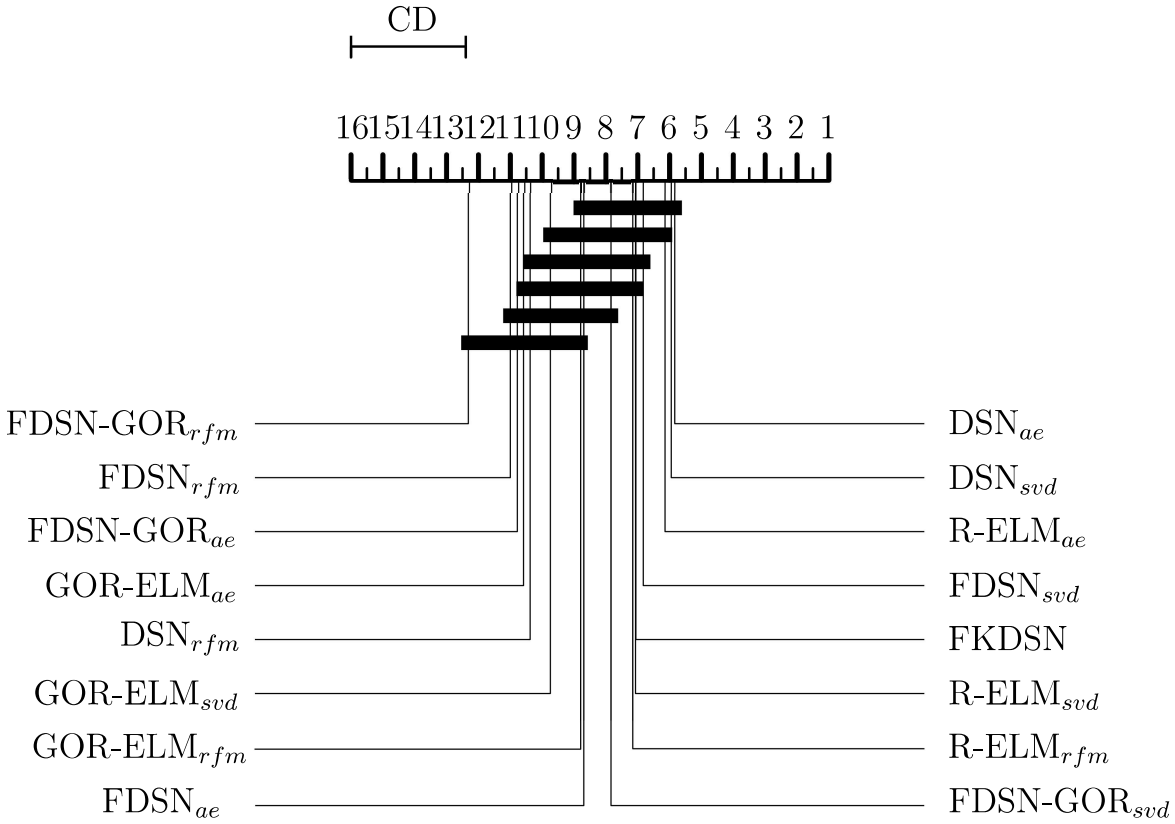
O teste estatístico de Friedman (DEMŠAR, 2006) foi aplicado nos resultados obtidos, considerando uma hipótese de que os métodos são equivalentes, usando um nível de significância de 5%. O teste obteve um valor-$p = 1{,}43 \times 10^{-17}$, rejeitando a hipótese nula. Assim, o teste de Nemenyi foi utilizado, cujo resultado pode ser visto na Figura 18. A distância crítica, neste caso, foi de 3,6025, ou seja, métodos cuja diferença entre *ranks* é maior que este valor não são equivalentes. O teste indica que o método FDSN_{rfm} não é estatisticamente similar aos 8 métodos com melhores *ranks*. Porém, quando combinado a uma técnica de inicialização, ele se torna similar a tais métodos. Os métodos FDSN, FDSN-GOR e $\text{FDSN-GOR}_{\text{ae}}$ não foram considerados equivalentes aos métodos baseados em R-ELM, exceto quando combinados com os métodos de inicialização de pesos.

Com a finalidade de comparar o desempenho entre os métodos de inicialização de pesos, as métricas aRRMSE obtidas neste experimento foram divididas em 3 grupos, chamados de "rfm", "ae" e "svd", onde cada grupo contém apenas os resultados dos métodos que utilizam a respectiva técnica de inicialização de pesos. Os 3 grupos foram comparados utilizando o teste estatístico de Wilcoxon unicaudal à esquerda. Ou seja, se a hipótese nula for rejeitada ao considerar qualquer par de grupos, a mediana das diferenças dos resultados é considerada menor que zero, e as métricas de aRRMSE do primeiro grupo são inferiores (menor erro quadrático). Ao comparar os três grupos par a par, a hipótese nula foi rejeitada ao comparar o grupo "ae" e "rfm" foram obtidos os valores de estatística de 0,0046, e ao comparar o grupo "svd" com os grupos "rfm" e "ae", com valores de estatística de $7,1 \times 10^{-6}$ e 0,0095, respectivamente. Isso significa que os grupos dos métodos que utilizam inicialização de pesos ortogonais ou por *autoencoder* foram considerados estatisticamente superiores ao grupo dos métodos com inicialização aleatória de pesos, e o grupo que utiliza inicialização ortogonal de pesos também foi considerado superior ao grupo que utiliza inicialização por *autoencoder*.

O mesmo teste foi repetido, porém dividindo os métodos em 4 grupos, com a

[6] A expressão "métodos baseados em R-ELM" indica referência aos métodos $\text{R-ELM}_{\text{rfm}}$, R-ELM_{ae} e $\text{R-ELM}_{\text{svd}}$. O mesmo é válido para GR-ELM, GOR-ELM, DSN, FDSN, FDSN-GR e FDSN-GOR.

Figura 18 – Teste de Nemenyi utilizando a métrica aRRMSE para os experimentos com problemas de regressão.



Fonte: Produção do próprio autor.

finalidade de avaliar os algoritmos de treinamento utilizados. Tais grupos foram chamados de “R-ELM”, “GOR-ELM”, “DSN” e “FDSN”. Neste caso, a hipótese nula foi rejeitada ao comparar os grupos “R-ELM” com “GOR-ELM”, “R-ELM” com “FDSN” e “DSN” com “FDSN”, com valores de estatística de $5{,}1 \times 10^{-6}$, $2{,}6 \times 10^{-4}$ e $2{,}1 \times 10^{-6}$, respectivamente. Isso significa que, para os problemas de regressão analisados, R-ELM foi considerado estatisticamente superior aos métodos GOR-ELM e FDSN, enquanto o método DSN foi considerado estatisticamente superior ao método FDSN, quando a métrica aRRMSE é considerada.

O tempo gasto para treinar cada modelo também foi analisado, onde os valores encontrados são apresentados no Apêndice B. Os *ranks* médios foram calculados e são apresentados na Tabela 9, onde os métodos baseados em FDSN obtiveram os melhores *ranks*. O teste de Friedman também foi aplicado, e a hipótese nula de que os métodos são equivalentes foi rejeitada com valor-$p < 10^{-30}$. Com isso, o teste de Nemenyi foi feito e a distância crítica de $3,6025$ é mostrada na Figura 19, que indica que os métodos baseados em FDSN e o FKDSN são estatisticamente superiores aos métodos que treinam redes maiores, como os baseados em R-ELM e GR-ELM. Vale notar que, o DSN sem

inicialização de pesos e com inicialização ortogonal foram os únicos métodos testados que foram considerados estatisticamente equivalentes aos métodos FDSN, quando tempo de processamento é levado em consideração.

Analisando os valores de tempo de treinamento, os métodos baseados em FDSN levaram um valor médio de aproximadamente 57% do tempo gasto pelos métodos baseados em DSN e 36% dos métodos baseados em R-ELM. Vale ressaltar o impacto no tempo de treinamento que a inicialização utilizando *autoencoder* causa, já que com exceção da técnica FDSN-GOR, o tempo de treinamento utilizando a inicialização por *autoencoder* foi maior do que utilizando inicialização ortogonal ou aleatória.

Tabela 9 – *Ranks* médios ($\overline{\text{Rank}}$) e tempo de treinamento no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 2. Os métodos estão ordenados pelo *rank* médio, de forma crescente.

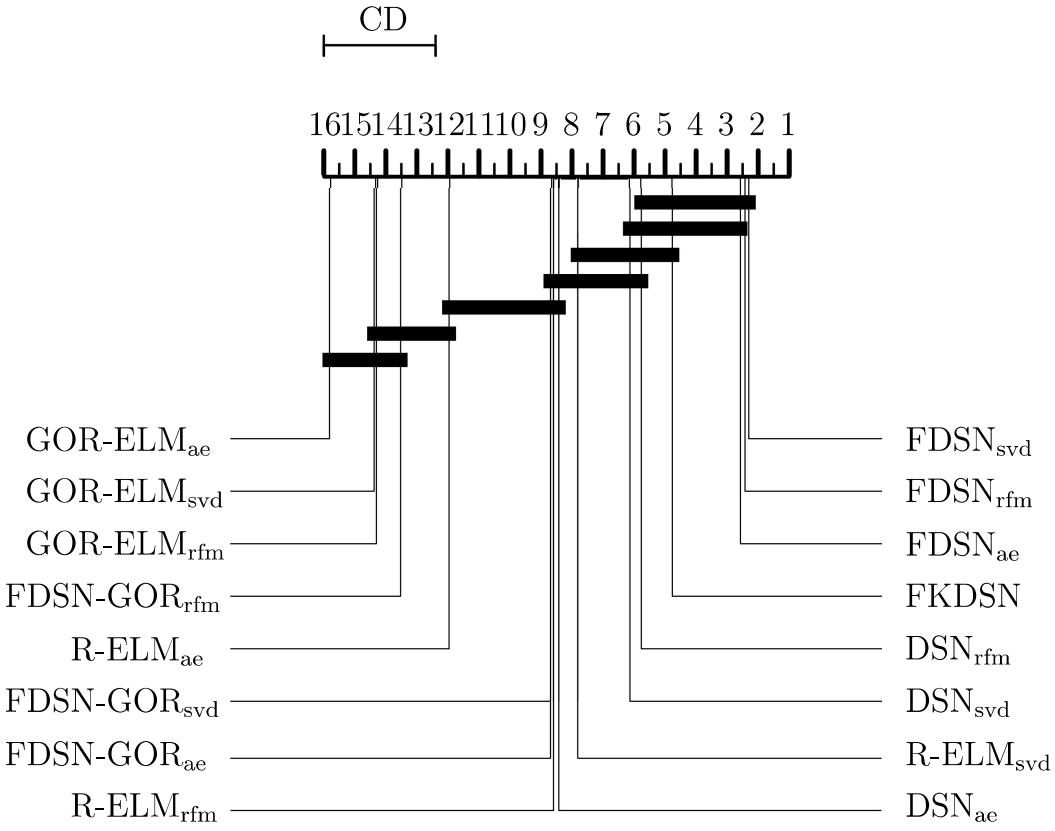
Método	$\overline{\text{Rank}}$	Tempo de treinamento (ms)
FDSN_{svd}	2,2927	41,40 (39,20)
FDSN_{rfm}	2,4146	41,80 (39,00)
FDSN_{ae}	2,5610	41,10 (39,10)
FKDSN	4,7561	57,30 (58,20)
DSN_{rfm}	5,7561	66,90 (63,10)
DSN_{svd}	6,1220	67,70 (62,80)
$\text{R-ELM}_{\text{svd}}$	7,8049	102,50 (93,10)
DSN_{ae}	8,4146	93,70 (96,20)
$\text{R-ELM}_{\text{rfm}}$	8,5854	106,30 (95,90)
$\text{FDSN-GOR}_{\text{svd}}$	8,6829	82,20 (67,50)
$\text{FDSN-GOR}_{\text{ae}}$	8,6829	82,50 (67,20)
R-ELM_{ae}	11,9512	209,10 (187,80)
$\text{FDSN-GOR}_{\text{rfm}}$	13,5122	964,60 (1392,30)
$\text{GOR-ELM}_{\text{rfm}}$	14,2927	2438,70 (2638,40)
$\text{GOR-ELM}_{\text{svd}}$	14,3659	2602,70 (2715,00)
$\text{GOR-ELM}_{\text{ae}}$	15,8049	2757,70 (2784,50)

Fonte: Produção do próprio autor.

Uma análise similar foi feita com o tamanho do modelo (número de bytes que ele necessita para ser armazenado). Os métodos baseados em FDSN obtiveram os melhores *ranks*, como pode ser visto na Tabela 10[7]. O teste de Friedman foi aplicado, e a hipótese nula de que os métodos são equivalentes foi rejeitada com valor-$p < 10^{-30}$. Com isso, o teste de Nemenyi foi feito e a distância crítica de 3,6025 é mostrada na Figura 20, que indica que o único método que foi considerado equivalente aos métodos baseados em FDSN foi o método FKDSN. Os demais foram considerados estatisticamente inferiores, quando o uso de memória é levado em consideração.

[7] O número apresentado de bytes se refere a implementação realizada neste trabalho em MATLAB, sem muita preocupação com otimização de código e memória. Valores diferentes podem ser obtidos em outras implementações. Porém, como apresentado na Seção 3.3.3, o uso de memória do FDSN tende a ser menor que das demais técnicas.

Figura 19 – Teste de Nemenyi utilizando o tempo de treinamento para os experimentos com problemas de regressão.



Fonte: Produção do próprio autor.

É possível destacar que os métodos baseados em DSN não foram considerados equivalentes a nenhum outro método, devido ao seu alto uso de memória. Assim, ao reaproveitar os pesos de uma DSN (o que dá origem ao método FDSN), o método deixaria de ter os piores *ranks* em uso de memória e passaria para os melhores.

Quanto aos valores numéricos, a quantidade de memória necessária para armazenar as redes FDSN foi de aproximadamente 24% do que foi utilizada para os métodos baseados em R-ELM e GOR-ELM, respectivamente. Vale notar que o método GOR-ELM inicia com uma rede completa e possui a capacidade de podar neurônios se julgar necessário, mas como a sua utilização média de memória foi muito próxima do R-ELM, a poda não foi aplicada. Já os métodos baseados em DSN, que obtiveram os piores *ranks*, necessitam de mais de cinco vezes mais bytes para serem armazenados que os métodos baseados em FDSN, em média. Isto mostra que o FDSN é mais adequado para ser utilizado em sistemas que possuem pouca memória disponível. Os métodos FDSN-GOR possuem a característica de aproveitar pesos do FDSN, mas também podem podar neurônios para substituí-los nos próximos módulos, o que faz com que ele necessite de mais uso de memória. Caso a poda não seja relevante, o método não podará neurônios e o seu uso de memória será próximo

ao uso do FDSN, o que foi o caso.

Tabela 10 – *Ranks* médios ($\overline{\text{Rank}}$) e tamanho do modelo em kilobytes no formato "média (desvio padrão)" dos métodos testados obtidos nas bases de dados da Tabela 2. Os métodos estão ordenados pelo *rank* médio, de forma crescente.

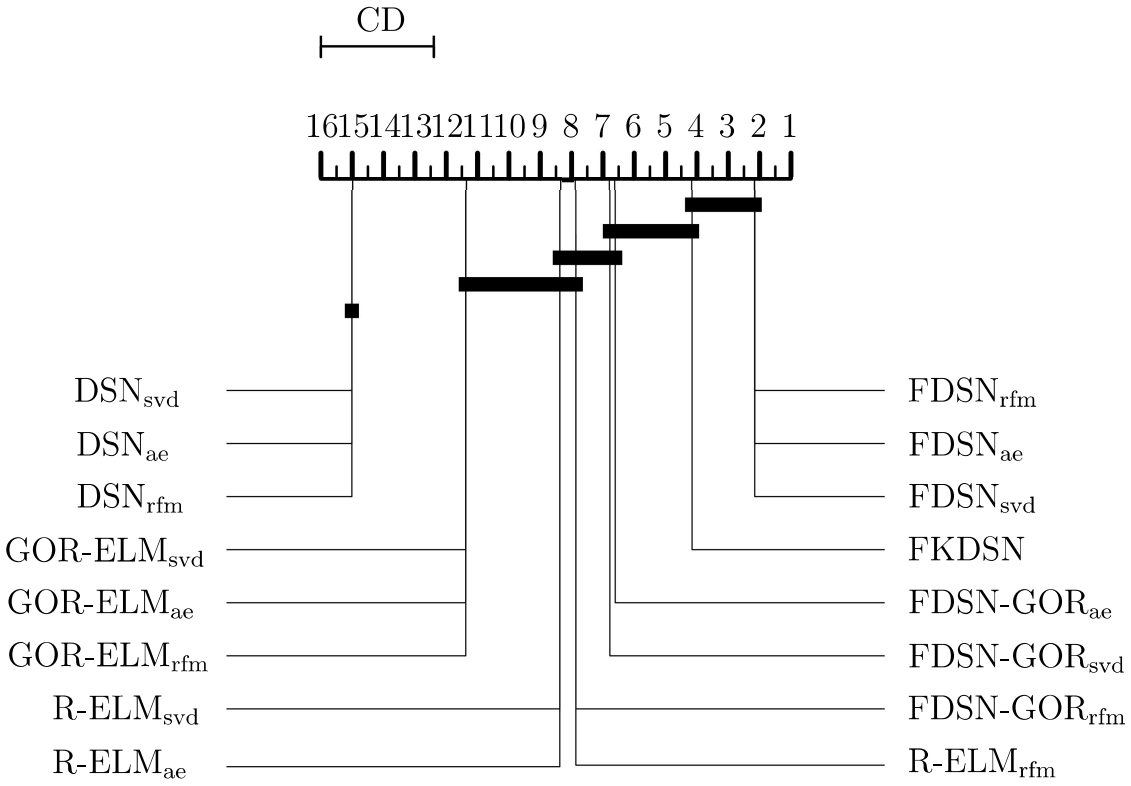
Método	$\overline{\text{Rank}}$	Tamanho do modelo (kilobytes)
FDSN_{rfm}	2,1463	78,33 (115,34)
FDSN_{ae}	2,1463	78,33 (115,34)
FDSN_{svd}	2,1463	78,33 (115,34)
FKDSN	4,1463	78,75 (115,34)
FDSN-GOR_{ae}	6,5976	99,87 (115,47)
FDSN-GOR_{svd}	6,7683	105,05 (117,10)
FDSN-GOR_{rfm}	7,8537	110,76 (111,23)
R-ELM_{rfm}	8,3659	327,08 (767,01)
R-ELM_{ae}	8,3659	327,08 (767,01)
R-ELM_{svd}	8,3659	327,08 (767,01)
GOR-ELM_{rfm}	11,3659	327,11 (767,01)
GOR-ELM_{ae}	11,3659	327,11 (767,01)
GOR-ELM_{svd}	11,3659	327,11 (767,01)
DSN_{rfm}	15,0000	421,49 (846,40)
DSN_{ae}	15,0000	421,49 (846,40)
DSN_{svd}	15,0000	421,49 (846,40)

Fonte: Produção do próprio autor.

3.5 Considerações Finais

De forma geral, os modelos propostos (FDSN e suas combinações com inicialização de pesos e FKDSN) não se destacaram consideravelmente em relação aos demais modelos para tarefas de classificação, quando a métrica acurácia é levada em consideração. Neste caso, elas foram consideradas equivalentes as técnicas com maior número de neurônios, quando treinadas com as técnicas baseadas em R-ELM e GR-ELM, principalmente quando associadas a um dos métodos de inicialização testados, no caso do FDSN. O teste estatístico de Wilcoxon foi utilizado, e seus resultados indicam que a técnica de inicialização de pesos por *autoencoder* é superior às outras técnicas comparadas quando é considerado um problema de classificação. Quanto ao tempo de treinamento, embora os melhores *ranks* foram obtidos pelos métodos baseados em FDSN e FKDSN, eles foram considerados estatisticamente similares aos outros métodos, mesmo levando 33% do tempo gasto pelas técnicas R-ELM. Quanto ao uso de memória, os métodos baseados em FDSN obtiveram os melhores *ranks*, seguidos do FKDSN, e ainda foram considerados estatisticamente equivalentes à maioria das demais técnicas, mesmo necessitando de apenas 25% da quantidade de memória da técnica R-ELM. Em termos numéricos, as técnicas propostas neste capítulo obtiveram métricas de acurácia aproximadas às demais, levaram menos tempo para serem treinadas e necessitaram menos memória para serem armazenadas, indicando que elas

Figura 20 – Teste de Nemenyi utilizando o tamanho do modelo em bytes para os experimentos com problemas de regressão.



Fonte: Produção do próprio autor.

podem ser adequadas para implementação em ambientes com recursos (como quantidade de memória e capacidade de processamento) restritos.

Já para os problemas de regressão, os métodos empilhados se destacaram. Os métodos com melhores *ranks* foram os baseados em DSN, que acabaram tendo um elevado uso de memória. Uma alternativa a este método, que tem uso reduzido de memória, é proposta neste trabalho, chamado de FDSN. Os métodos baseados em FDSN (incluindo suas combinações com inicialização de pesos e o FKDSN), foram capazes de obter bons *ranks*, obtendo valores superiores a técnicas que treinam redes maiores, como o R-ELM e GOR-ELM, principalmente quando utilizados em conjunto com um método de inicialização de pesos. O teste estatístico de Wilcoxon foi utilizado, e seus resultados indicam que a técnica de inicialização de pesos ortogonais é superior às outras técnicas comparadas quando é considerado um problema de regressão. Quanto ao tempo de treinamento, os melhores *ranks* foram obtidos pelos métodos baseados em FDSN e FKDSN, levando cerca de 57% do tempo gasto pelos métodos baseados em DSN e 36% dos métodos baseados em R-ELM,

e ainda foram considerados estatisticamente superiores a maioria dos demais métodos comparados. Quanto ao uso de memória, os métodos baseados em FDSN obtiveram os melhores *ranks*, sendo considerados estatisticamente superiores a todos os outros métodos testados, com exceção do FKDSN, que obteve o quarto melhor *rank*, necessitando apenas de aproximadamente 24% da memória necessária para os métodos baseados em R-ELM e GOR-ELM. Em termos numéricos, as mesmas conclusões obtidas nos testes com problemas de classificação com relação ao uso das técnicas propostas em ambientes com recursos computacionais limitados (memória e capacidade de processamento) podem ser feitas para regressão, com a diferença em que é possível atingir os melhores *ranks* com as técnicas empilhadas.

4 Estudo de Caso de Métodos Offline

Neste capítulo é apresentado um estudo de caso dos métodos *offline* aplicados a duas bases de dados reais de predição de séries temporais de produção de energias renováveis, como energia eólica e energia solar. O estudo de caso foi baseado no trabalho de Severiano et al. (2021) e, portanto, segue a metodologia destes autores.

Segundo Severiano et al. (2021), o tema de energias renováveis ganhou popularidade na década de 2010 em diante. Porém, esse tipo de energia traz uma elevada incerteza com relação à sua produção, já que depende de fatores externos que envolvem fatores climáticos, como a incidência solar em placas geradoras ou a velocidade do vento que movimenta turbinas, apresentando volatilidade e aleatoriedade nas quantidade produzidas.

A incerteza de produção se torna um problema na integração com os grandes sistemas de geração e distribuição de energia elétrica de um país (no Brasil, seria o Sistema Interligado Nacional), pois este deve manter a estabilidade da distribuição de energia elétrica. O risco de uma instabilidade pode ser reduzido se existir uma ferramenta capaz de predizer com baixo erro a quantidade de energia que será produzida em instantes de tempo futuros, e o sistema pode se ajustar com antecedência.

Assim, ferramentas robustas de predição destas grandezas são importantes para auxiliar tarefas relacionadas ao comércio, no contexto de geração e distribuição de energia elétrica. Estas ferramentas podem ser técnicas de aprendizado de máquina, que obtém conhecimento de um conjunto de dados previamente disponível e o utiliza para predizer grandezas em instantes de tempo futuros, ou então abordagens estatísticas, onde o primeiro tipo costuma obter melhores resultados na literatura (SEVERIANO et al., 2021).

Usualmente, tais conjuntos de dados podem ser compostos por séries temporais, onde a quantidade de energia pode ser amostrada em intervalos de tempo e disponibilizadas como um sinal de única dimensão, que será analisado seja por técnicas estatísticas ou de inteligência artificial. No caso da utilização de técnicas de aprendizado de máquina, tais séries temporais podem ser analisadas como quaisquer base de dados com características, e é possível realizar um pré-processamento, ou analisar as bases diretamente. É necessário, porém, realizar uma adaptação para transformar a série temporal em amostras com características e um alvo de predição.

Considerando que a série temporal é representada por $x(t)$, uma possível adaptação seria construir o conjunto $[x(t-k)\ x(t-k+1) \cdots x(t)]$, onde k é um parâmetro de "atraso", como características de entrada de uma técnica de aprendizado de máquinas que tentará predizer $x(t+l)$, onde l é um parâmetro que controla o horizonte de predição. Assim, a série temporal pode ser tratada como um conjunto de dados padrão e ser analisada por

uma técnica de aprendizado de máquina, como uma rede neural.

4.1 Metodologia

Nesta seção são apresentados detalhes da metodologia proposta por Severiano et al. (2021) para testar métodos de predição em duas bases de dados de geração de energia renovável.

4.1.1 Bases de dados e Métricas Utilizadas

Neste estudo de caso, duas bases de dados de energias renováveis são utilizadas. Ambas as bases são públicas e são compostas de séries temporais extraídas de sistemas reais de produção de energia solar e eólica. A base de dados de energia solar foi obtida do repositório *United States National Renewable Energy Laboratory*[1] (NREL) e tem detalhes apresentados na Tabela 11. Já a base de dados de geração de energia eólica foi disponibilizada pela *Global Energy Forecasting Competition 2012*[2] (GEF-Com 2012) e tem detalhes apresentados na Tabela 12. Nesta base, por ser utilizada em competição, as medições foram normalizadas de -1 a 1 para omitir possíveis detalhes geográficos.

Tabela 11 – Detalhes da base de dados de Energia Solar.

Base de dados - Energia Solar	
Variável	*Global Horizontal Irradiance* (GHI)
Unidade	W/m²
Frequência de Amostragem	15 minutos
Intervalo de Tempo	01/06/2010 a 31/07/2011
Num. Séries Temporais	16
Num. Amostras	36051

Fonte: Adaptado de (SEVERIANO et al., 2021).

Tabela 12 – Detalhes da base de dados de Energia Eólica.

Base de dados - GEFCom 2012	
Variável	Potência Eólica
Unidade	Normalizado*
Frequência de Amostragem	1 hora
Intervalo de Tempo	01/06/2010 a 31/07/2011
Num. Séries Temporais	7
Num. Amostras	8761

Fonte: Adaptado de (SEVERIANO et al., 2021).

[1] ⟨https://midcdmz.nrel.gov/apps/sitehome.pl?site=OAHURS1M⟩
[2] ⟨https://www.kaggle.com/c/GEF2012-wind-forecasting/overview⟩

As métricas utilizadas para avaliar o desempenhos das técnicas nestas bases de dados foram a *Root Mean Square Error* (RMSE) e a métrica *Symmetric Mean Absolute Percentage Error* (SMAPE), que podem ser calculadas através das Equações 4.1 e 4.2, respectivamente, onde $y_j^{(t)}$ é o elemento referente a j-ésima das d dimensões do alvo da base de dados $\boldsymbol{y}^{(t)}$ e $\hat{y}_j^{(t)}$ é o elemento da dimensão j da predição $\hat{\boldsymbol{y}}^{(t)}$ de um método para uma amostra no instante de tempo t.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(\frac{\sum_{j=1}^{d}\left(y_j^{(t)} - \hat{y}_j^{(t)}\right)^2}{d}\right)} \tag{4.1}$$

$$\text{SMAPE} = 100\% \cdot \frac{1}{n}\sum_{t=1}^{n}\left(\frac{1}{d}\sum_{j=1}^{d}\frac{|y_j^{(t)} - \hat{y}_j^{(t)}|}{|y_j^{(t)}| + |\hat{y}_j^{(t)}|}\right) \tag{4.2}$$

Vale notar que as bases de dados utilizam informações simultâneas de várias séries temporais (obtidas de estações de geração de energia geograficamente próximas). Isto significa que, a cada instante de tempo, são previstas amostras de n séries temporais distintas, com base em amostras passadas destas mesmas séries. Assim, os cálculos das métricas RMSE e SMAPE foram adaptados para avaliar a qualidade de predições de dados com n dimensões.

Um breve experimento foi realizado para avaliar o comportamento das duas bases de dados utilizadas ao longo do tempo. Primeiramente, foi obtida a componente principal (utilizando PCA) da base de dados a partir de todas as amostras disponíveis. E com essa componente, foi possível projetar amostras da base de dados para apenas uma dimensão. Para analisar o comportamento desta componente principal ao longo da base de dados, lotes sequenciais de tamanho 100 foram criados a partir da base de dados e projetados para uma dimensão utilizando a componente principal encontrada, o que resulta em um vetor de 100 números por lote. Os *boxplots* dos vetores dos lotes localizados aproximadamente de 10% em 10% da base de dados são apresentados na Figura 21. É possível ver que há uma variação da distribuição da componente principal ao longo do tempo para ambas as bases, o que caracteriza o chamado *concept drift*.

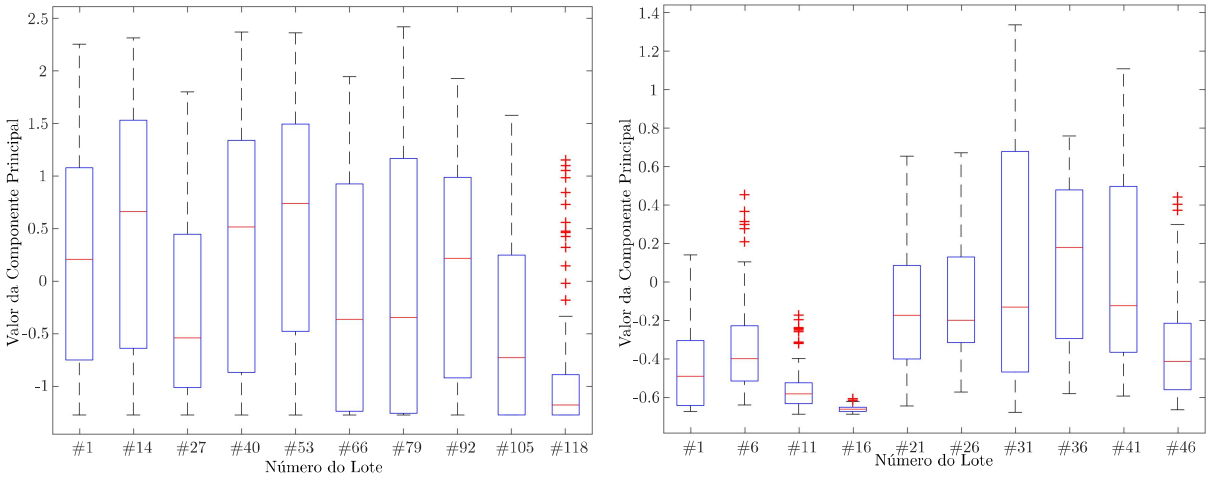
4.1.2 Métodos Utilizados e Detalhes de Implementação

Os métodos utilizados neste estudo de caso foram os mesmos utilizados por Severiano et al. (2021). Foram aplicados os métodos *Vector Autoregressive* (VAR), *Multilayer Perceptron* (MLP), *Fuzzy Information Granule Time Series* (FIG-FTS) e o método *Evolving Multivariate Fuzzy Times Series* (e-MVFTS), no qual o último foi proposto em (SEVERIANO et al., 2021), onde podem ser encontrados mais detalhes sobre estas técnicas. Para comparação com as técnicas propostas neste trabalho, os métodos FDSN_{rfm}, FDSN_{ae},

Figura 21 – Análise de Componentes Principais nas Bases de Dados *Wind* e *Solar Energy*.

(a) Base Solar Energy

(b) Base Wind Energy



Fonte: Produção do próprio autor.

FDSN_{svd} e FKDSN, como definidos na Seção 3.4.1.3, foram adicionados ao *framework* disponibilizado pelos autores.

Os métodos e-MVFTS, VAR e FIG-FTS são projetados para lidar com os dados no formato de série temporal (isto é, sua entrada é um vetor a ser processado). Já as técnicas MLP e as propostas neste trabalho lidam com dados no formato entrada-saída (isto é, dois vetores distintos são utilizados). Portanto, a série temporal teve seu formato adaptado para estas técnicas como se fosse um problema de regressão comum, como citado no início deste capítulo. Seguindo a metodologia de Severiano et al. (2021), as técnicas consideram um "atraso" de duas amostras (no caso, cada amostra seria um instante de tempo que tem um certo número de séries temporais, de acordo com a base) e um horizonte de predição de uma amostra. No caso das técnicas MLP, FKDSN e as baseadas em FDSN, elas têm um número de saídas e de características iguais a 1 e 2 vezes o número de séries temporais das bases de dados, respectivamente. Como exemplo, o "atraso" de 2 na base de Energia Solar indica 32 características como entrada das técnicas, e uma saída com 16 dimensões.

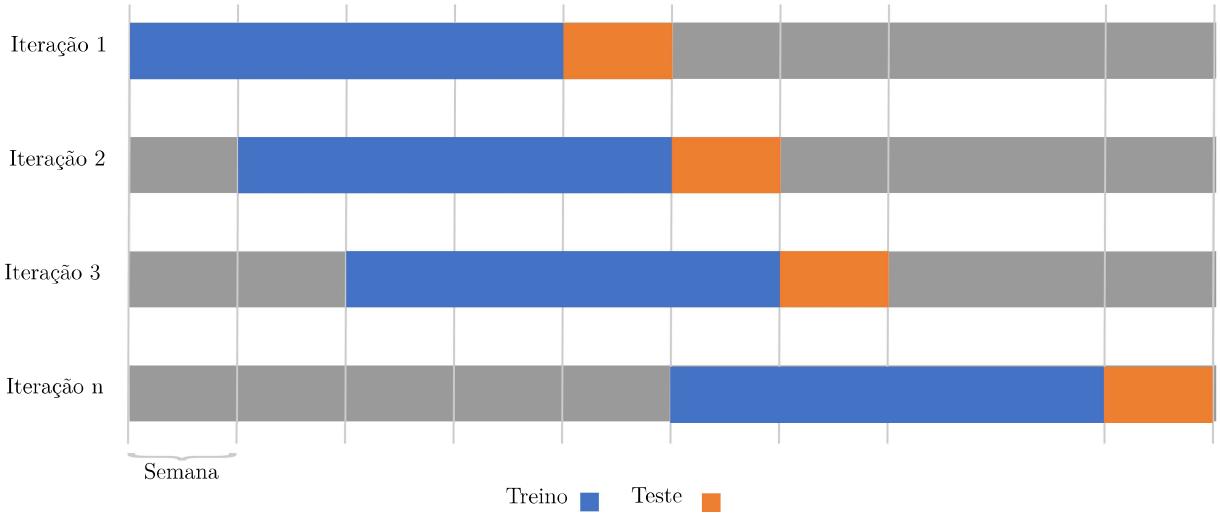
Excepcionalmente neste capítulo, os experimentos foram executados em um computador com sistema operacional Ubuntu 20.04.2 LTS, com processador Intel® Core™ i7-8750H (6 núcleos a 2.2GHz) com 16 GB de memória RAM. Com exceção das técnicas propostas neste trabalho, todos os códigos envolvidos nos testes feitos por Severiano et al. (2021) estão disponíveis publicamente[3,4] na linguagem de programação Python. Assim, as técnicas propostas neste trabalho foram implementadas na linguagem de programação Python e incluídas no *framework* de testes disponibilizado pelos autores. Os códigos utili-

[3] ⟨https://minds.eng.ufmg.br/download_resources⟩
[4] ⟨https://github.com/cseveriano/spatio-temporal-forecasting⟩

zados neste trabalho podem ser consultados em ⟨https://doi.org/10.17632/y5ytz6kydf.1⟩

O experimento foi conduzido utilizando o conceito de janela deslizante. Assim, um subconjunto da série temporal é escolhido como conjunto de treino (ou treino e validação), e uma determinada quantidade de amostras após essa janela são consideradas como amostras de teste. Todas as técnicas avaliadas são treinadas com a janela de treino (caso necessário, a janela pode ser dividida em treino e validação), e testadas na janela de teste. As métricas são calculadas através das Equações 4.1 e 4.2, e a janela "desliza" para instantes de tempo futuros (com sobreposição), como mostra a Figura 22, onde amostras de 4 semanas consecutivas são utilizadas para treino, e amostras da semana seguinte para teste. Isto é feito até que todas as possíveis janelas deslizantes sejam analisadas.

Figura 22 – Exemplo de uma Análise por Janela Deslizante.



Fonte: Adaptado de (SEVERIANO et al., 2021).

Após processar todas as janelas, cada método analisado terá um vetor de métricas como resultado. As métricas apresentadas são baseadas em uma única execução da janela deslizante para os métodos e-MVFTS, VAR e FIG-FTS, pois estes sempre retornam o mesmo resultado para diferentes execuções na mesma base de dados. Já os métodos MLP, FDSN_{rfm}, FDSN_{ae}, FDSN_{svd} e FKDSN, por terem uma natureza aleatória, foram executados 10 vezes e as médias das métricas das 10 execuções foram consideradas. A função utilizada para estimar o tamanho de um objeto (classificador/regressor) em Python está disponível na internet[5].

4.1.3 Especificação dos Hiperparâmetros

Quanto aos hiperparâmetros empregados nos testes deste capítulo, os valores foram os mesmos do trabalho de Severiano et al. (2021). O método VAR não possui parâmetros

[5] ⟨https://code.activestate.com/recipes/546530/⟩

além do "atraso" da base de dados. O método e-MVFTS teve seus hiperparâmetros fixados em limite de variância $r_0 = 0,001$ e distância cumulativa $\alpha = 0,6$. Os hiperparâmetros do MLP foram obtidos de um *grid* de hiperparâmetros definido pelos autores, e a rede resultante possui duas camadas ocultas, sendo a primeira com 512 elementos e a segunda com 64, ambas com função de ativação do tipo ReLU. A camada de saída possui função de ativação sigmoide e a rede toda foi treinada com o otimizador Adam, usando 300 épocas de treinamento, com tamanho de lote de 64 amostras.

Já para os métodos FKDSN e os baseados em FDSN, treinados com RK-ELM e R-ELM, respectivamente, uma abordagem similar a empregada na Seção 3.4.1.4 foi feita, seguindo os passos para uma comparação justa entre os métodos. Neste trabalho os hiperparâmetros foram definidos usando um *grid*, um método de busca e uma divisão do conjunto de dados inteiro em 60% como amostras de treino e o restante para validação, da mesma forma que os hiperparâmetros da MLP foram definidos originalmente. O método de busca utilizado foi o 5-fold para os 4 métodos propostos neste trabalho, e o *grid* de busca continha o parâmetro de regularização C, que era escolhido do conjunto $\{2^{-20}; 2^{-19}; \ldots; 2^{20}\}$, número de módulos S do conjunto $\{1, 2, \ldots, 10\}$, e número de neurônios ocultos por módulo L do conjunto $\{10, 50, 100\}$. Para os métodos baseados em FDSN, foram consideradas as funções de ativação sigmoide, *hard-limit*, gaussiana e multiquádrica, definidas pelas Equações 3.40, 3.41, 3.42 e 3.43, respectivamente. Já para o método FKDSN, foi considerado o *kernel* gaussiano com hiperparâmetro γ escolhido do conjunto $\{2^{-20}; 2^{-19}; \ldots; 2^{20}\}$.

4.2 Experimentos e Resultados

Nesta seção são apresentados os resultados obtidos no *framework* disponibilizado por Severiano et al. (2021), considerando as bases de dados *Solar Energy* e *Wind Energy*.

4.2.1 Base de Dados *Solar Energy*

Os valores da RMSE, SMAPE, tempo de treinamento e memória necessária para armazenar os métodos obtidos no experimento considerando a base de dados *Solar Energy* podem ser vistos na Tabela 13. Os hiperparâmetros encontrados pela validação cruzada realizada nesta base de dados podem ser vistos no Apêndice C. Os resultados apresentados mostram que os métodos propostos neste trabalho foram superiores aos métodos testados por Severiano et al. (2021) quando as métricas de memória utilizada, SMAPE e RMSE são consideradas. No caso do tempo de treinamento, o método VAR obteve o melhor valor, enquanto os métodos FDSN_{rfm} e FDSN_{svd} atingiram um tempo aproximadamente 60% maior. Outro ponto a ser destacado é que os resultados das técnicas e-MVFTS, VAR, MLP e FIG-FTS foram aproximadamente iguais aos apresentados em (SEVERIANO et

al., 2021).

Com relação à métrica RMSE, o melhor valor foi obtido pelo método FDSN_{rfm}, seguido do FKDSN e do VAR, onde os três valores foram próximos, com a RMSE do método VAR sendo 1,3% maior que a do método FDSN_{rfm}. O *boxplot* das métricas obtidas nas janelas deslizantes pode ser visto na Figura 23, onde as três melhores medianas dos *boxplots* ocorreram na mesma ordem das médias. É possível notar que alguns *outliers* foram obtidos nos métodos MLP e FDSN_{ae}, o que pode ter prejudicado a RMSE média destes métodos. Também é possível ver, na Figura 24, o gráfico da métrica RMSE ao longo das janelas deslizantes, onde as técnicas aparentam seguir um padrão similar de crescimento ou decrescimento do valor da métrica. Isto ocorre pois a cada janela deslizante testada, um novo modelo é criado e treinado com as 4 semanas anteriores. Assim, as técnicas não aproveitam conhecimentos antigos e sempre aprendem do mesmo conjunto de dados. Se a janela de treino representar bem a janela de testes, a tendência é que haja um decrescimento na métrica, e se ele não for um bom representante, a tendência é um aumento na métrica para todos os modelos.

Já com relação à métrica SMAPE, na qual o método proposto por Severiano et al. (2021) se destacou nesta base de dados, o melhor valor obtido nos testes realizados foi o do método FDSN_{svd}, seguido pelo FDSN_{ae} e e-MVFTS. Ou seja, dois dos métodos propostos neste trabalho atingiram métricas melhores que a técnica e-MVFTS. Da mesma forma que com a métrica RMSE, os valores médios das 3 melhores técnicas foram similares, com a SMAPE do método e-MVFTS sendo 3,6% maior que a do FDSN_{svd}. O *boxplot* das métricas obtidas nas janelas deslizantes pode ser visto na Figura 25, onde as três melhores medianas dos *boxplots* ocorreram na mesma ordem das médias e também atingiram valores similares, além de não ter a presença de pontos considerados *outliers*.

Já para a métrica de memória necessária para armazenar os modelos, os melhores valores foram obtidos pelas técnicas FDSN_{ae}, seguido de FKDSN e FDSN_{svd}. Vale notar que FDSN_{ae} ocupa cerca de 15% da memória ocupada pelas técnicas VAR e e-MVFTS, obtendo valores similares de RMSE e SMAPE que estas técnicas. Já os demais métodos propostos nesta tese ocupam aproximadamente metade da memória ocupada pelas técnicas VAR e e-MVFTS.

4.2.2 Base de Dados *Wind Energy*

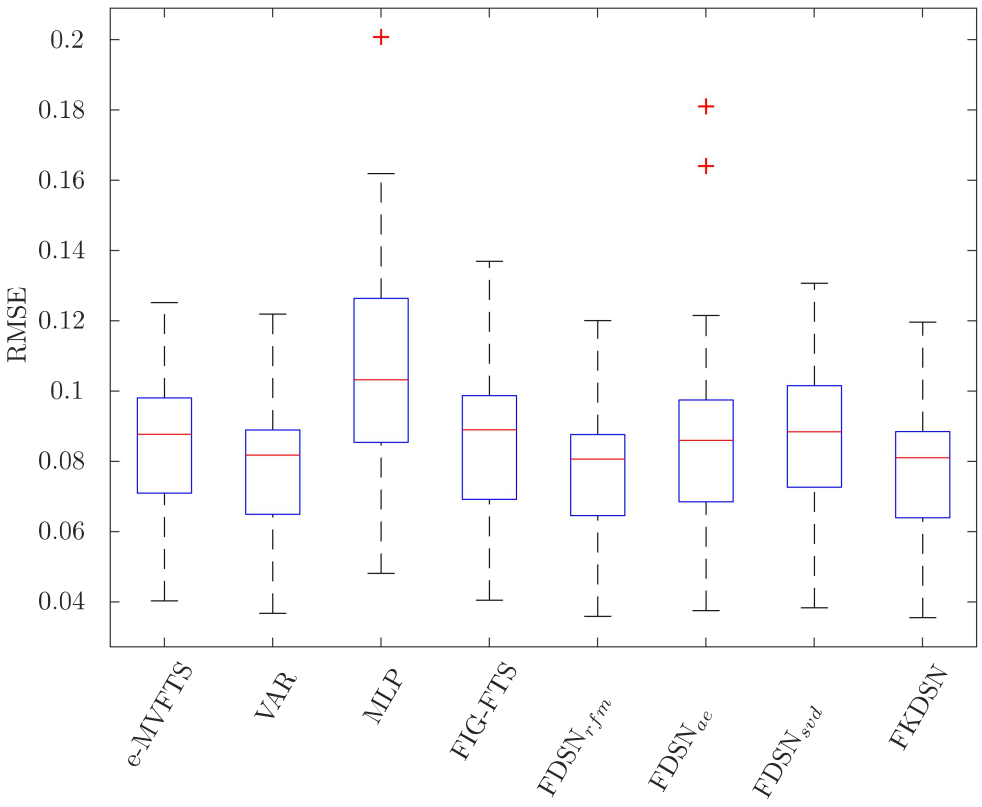
Os valores da RMSE, SMAPE, tempo de treinamento e memória necessária para armazenar os métodos obtidos no experimento considerando a base de dados *Wind Energy* podem ser vistos na Tabela 14. Os hiperparâmetros encontrados pela validação cruzada realizada nesta base de dados podem ser vistos no Apêndice C. Os resultados apresentados mostram que os métodos propostos neste trabalho foram superiores aos métodos testados por Severiano et al. (2021), com exceção do método VAR, quando as métricas SMAPE e

Tabela 13 – Métricas obtidas nos testes realizados com o *framework* de Severiano et al. (2021) usando a base de dados *Solar Energy*

Método	Tempo Gasto (s)	RMSE	SMAPE	Memória (kB)
e-MVFTS	275.1237	0.0854 (0.0192)	21.2244 (3.9193)	466.5091
VAR	**2.8247**	0.0778 (0.0171)	25.5301(3.0819)	469.5000
MLP	275.7640	0.1060 (0.0305)	26.8517 (3.6858)	1747.8062
FIG-FTS	81.3255	0.0862 (0.0208)	26.6996(4.0626)	17281.8745
$FDSN_{rfm}$	4.7148	**0.0768 (0.0173)**	22.2618 (3.1421)	274.4145
$FDSN_{ae}$	4.4528	0.0860 (0.0262)	20.5068 (2.3832)	**66.9766**
$FDSN_{svd}$	10.0588	0.0868 (0.0196)	**20.4811 (2.3429)**	274.3828
FKDSN	5.7667	0.0771 (0.0173)	22.3923 (2.9858)	221.6641

Fonte: Produção do próprio autor

Figura 23 – *Boxplots* das RMSE obtidas nos experimentos com a base de dados *Solar Energy*

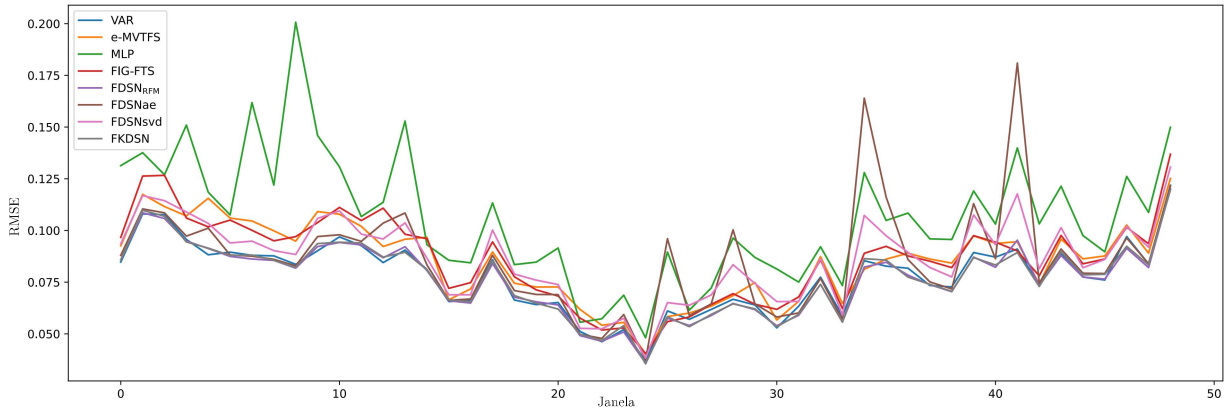


Fonte: Produção do próprio autor.

RMSE são consideradas. No caso do tempo de treinamento, o método VAR obteve o melhor valor, enquanto os métodos $FDSN_{rfm}$ e $FDSN_{ae}$ atingiram um tempo aproximadamente 100% maior. Assim como ocorrido na base de dados *Solar Energy*, os resultados das técnicas e-MVFTS, VAR, MLP e FIG-FTS foram aproximadamente iguais aos apresentados em (SEVERIANO et al., 2021).

Com relação a métrica RMSE, o melhor valor foi obtido pelo método VAR, seguido

Figura 24 – Gráfico da RMSE obtidas nas janelas deslizantes do experimento com a base de dados *Solar Energy*



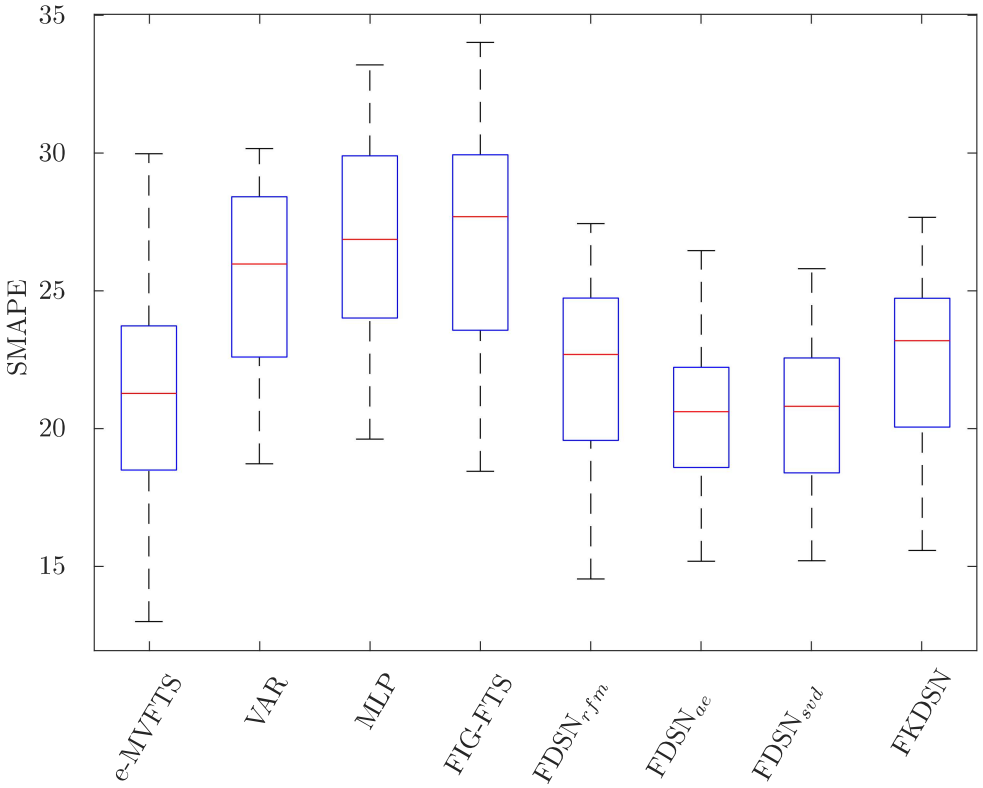
Fonte: Produção do próprio autor.

do FKDSN e do FDSN$_{ae}$, onde os três valores foram próximos, com a RMSE do método FDSN$_{ae}$ sendo 3,3% maior que a do método VAR. O *boxplot* das métricas obtidas nas janelas deslizantes pode ser visto na Figura 26, onde as três melhores medianas dos *boxplots* ocorreram na mesma ordem das médias. É possível notar que alguns *outliers* foram obtidos em todos os métodos testados. Neste caso, porém, a maioria dos *outliers* foram inferiores ou muito próximos aos quartis que formam os *boxplots*. Também é possível ver, na Figura 27, o gráfico da métrica RMSE ao longo das janelas deslizantes, onde as técnicas aparentam seguir um padrão similar de crescimento ou decrescimento do valor da métrica, de forma similar ao ocorrido na Seção 4.2.1.

Já com relação à métrica SMAPE, o melhor valor obtidos nos testes realizados foi o do método VAR, seguido pelos métodos FKDSN e FDSN$_{rfm}$. Ou seja, os métodos propostos neste trabalho só foram inferiores do que o método VAR, obtendo melhores resultados que os métodos e-MVFTS e FIG-FTS. Da mesma forma que com a métrica RMSE, os valores médios das 3 melhores técnicas foram similares, com a SMAPE do método FDSN$_{rfm}$ sendo 1,8% maior que a do VAR. O *boxplot* das métricas obtidas nas janelas deslizantes pode ser visto na Figura 28, onde as três melhores medianas dos *boxplots* foram obtidas pelos métodos VAR, FDSN$_{rfm}$ e FKDSN, também com valores similares. Vale ressaltar que o segundo e terceiro colocados na média das SMAPE foram o terceiro e segundo colocado na mediana, respectivamente. Isto pode ter ocorrido devido aos *outliers* que podem ser vistos no *boxplot*, que ocorreram em todas as técnicas, prejudicando o valor de suas respectivas médias.

Já para a quantidade de memória necessária para armazenar os modelos, os melhores valores foram obtidos pelas técnicas FDSN$_{ae}$, seguido de FKDSN e FDSN$_{svd}$ (empatado com FDSN$_{rfm}$). Vale notar que FDSN$_{ae}$ ocupa cerca de 10% da memória ocupada pelas

Figura 25 – *Boxplots* das SMAPE obtidas nos experimentos com a base de dados *Solar Energy*



Fonte: Produção do próprio autor.

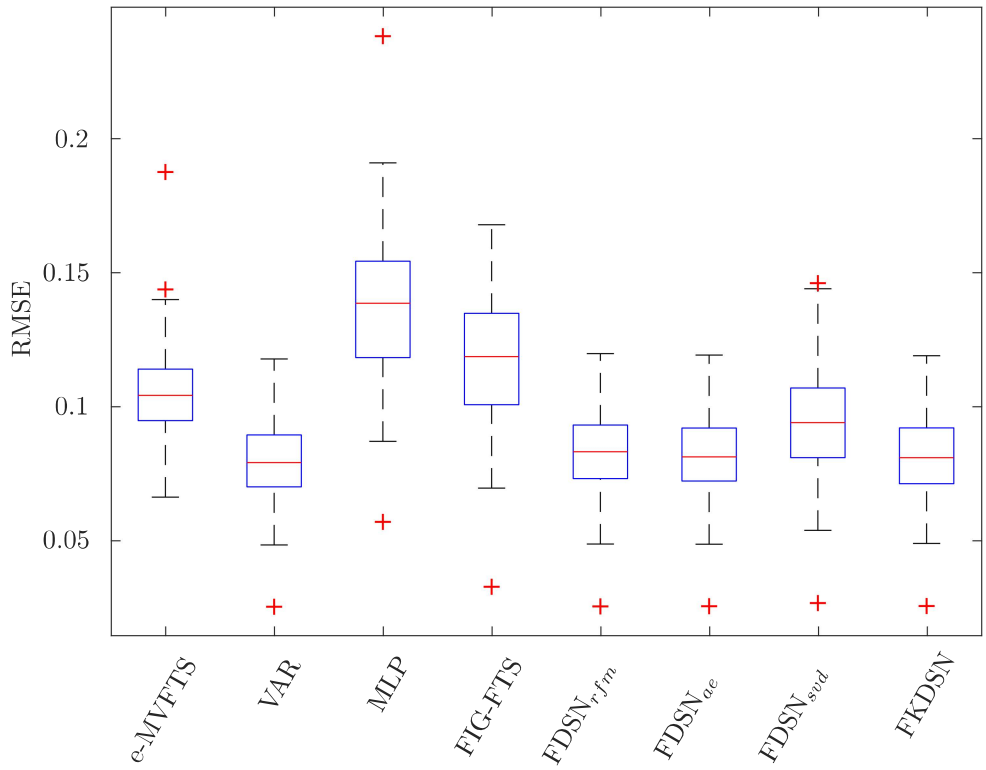
técnicas VAR e 1% do e-MVFTS, obtendo valores similares de RMSE e SMAPE aos obtidos por estas técnicas. Já os demais métodos propostos nesta tese ocupam aproximadamente 20% da memória ocupada pelo VAR e 2% da memória ocupada pelo e-MVFTS.

Tabela 14 – Métricas obtidas nos testes realizados com o *framework* de Severiano et al. (2021) usando a base de dados *Wind Energy*

Método	Tempo Gasto (s)	RMSE	SMAPE	Memória (kB)
e-MVFTS	292.1506	0.1061 (0.0201)	33.6979 (11.7129)	0.9670
VAR	**0.7570**	**0.0789 (0.0165)**	**28.0303 (8.8259)**	0.0918
MLP	213.0365	0.1385 (0.0341)	35.4312 (8.4680)	1.7549
FIG-FTS	202.6340	0.1179 (0.0257)	32.2817 (6.4839)	74.2629
$FDSN_{rfm}$	1.4881	0.0824 (0.0182)	28.5418 (8.7769)	0.0203
$FDSN_{ae}$	1.6270	0.0815 (0.0176)	28.5440 (8.8030)	**0.0096**
$FDSN_{svd}$	1.7880	0.0937 (0.0245)	30.3469 (8.7901)	0.0203
FKDSN	1.7792	0.0810 (0.0175)	28.3394 (8.8064)	0.0198

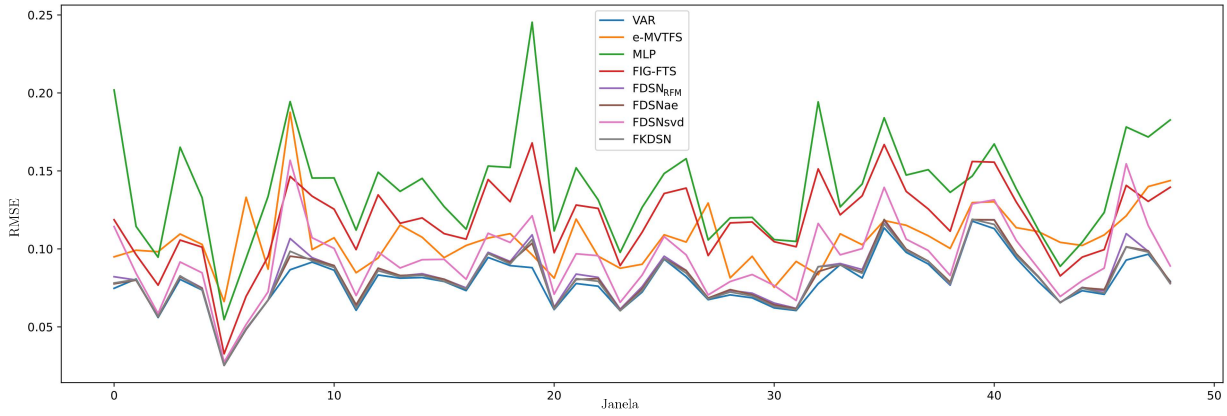
Fonte: Produção do próprio autor

Figura 26 – *Boxplots* das RMSE obtidas nos experimentos com a base de dados *Wind Energy*



Fonte: Produção do próprio autor.

Figura 27 – Gráfico da RMSE obtidas nas janelas deslizantes do experimento com a base de dados *Solar Energy*

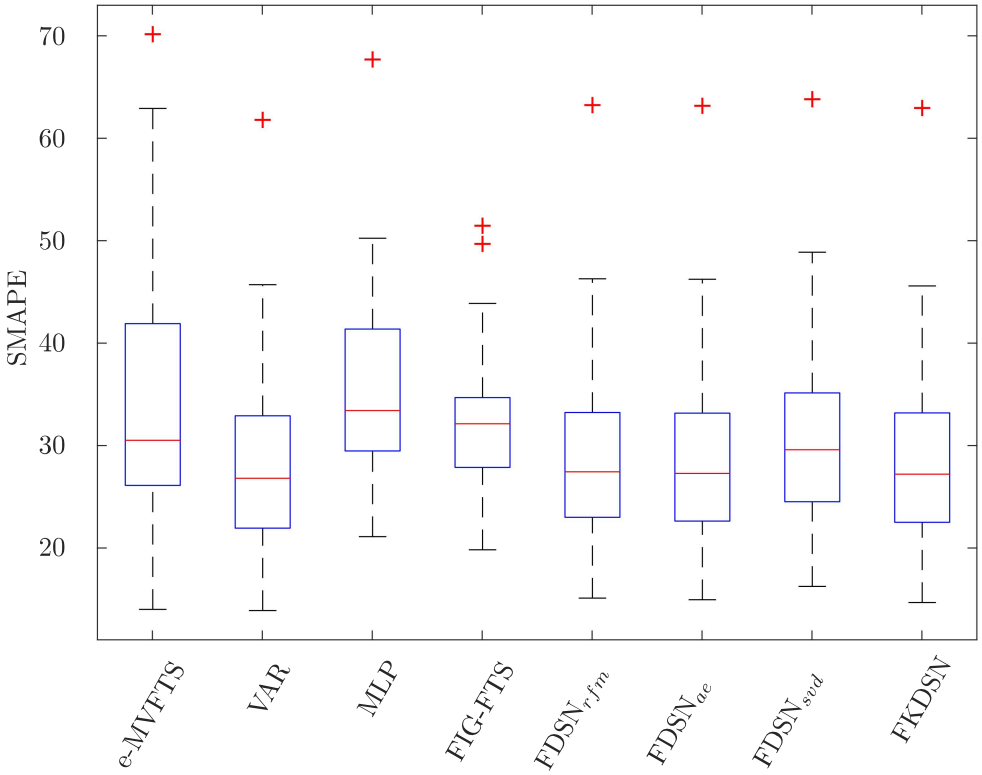


Fonte: Produção do próprio autor.

4.3 Considerações Finais

Os três métodos *offline* propostos neste trabalho foram testados no *framework* publicado por Severiano et al. (2021) e obtiveram resultados satisfatórios utilizando o

Figura 28 – *Boxplots* das SMAPE obtidas nos experimentos com a base de dados *Wind Energy*



Fonte: Produção do próprio autor.

mesmo procedimento de testes feito pelos autores. Na base de dados *Solar Energy*, métricas melhores do que as obtidas pelos demais métodos foram alcançadas, tanto em memória utilizada quanto métricas de erro. Já na base *Wind Energy*, o método VAR obteve os melhores resultados, exceto no uso de memória, o que já havia ocorrido na publicação original de Severiano et al. (2021). De qualquer forma, os métodos propostos obtiveram baixos tempos de treinamento, baixo uso de memória, e métricas similares ou até melhores que todos os métodos comparados nas duas bases de dados, o que pode indicar que eles são adequados para utilização em bases de dados de problemas reais e em situações com uso restrito de memória.

Vale destacar que os testes estatísticos aplicados no Capítulo 3 não foram utilizados neste capítulo pois eles não foram originalmente empregados na publicação de Severiano et al. (2021), além de que apenas duas bases de dados foram testadas ($D = 2$), o que significa que a distância crítica que diferencia as técnicas assumiria um valor elevado, como mostra a Equação 3.36, que consideraria todas as técnicas similares. Além disso, segundo Demšar (2006), alguns testes requerem suposições acerca da distribuição dos dados quando poucas bases de dados são utilizadas, e caso essa suposição não seja atendida, o teste pode falhar. Assim, optou-se por não utilizar testes estatísticos neste experimento.

5 Propostas considerando métodos *Online*

Neste capítulo é apresentada uma variação do método FDSN. Esta técnica é chamada de *Online Sequential Fast Deep Stacked Network* (OSFDSN) e é um método *online* de aprendizado de máquina. Ou seja, ele é capaz de adaptar seus parâmetros e/ou sua arquitetura de acordo com a chegada de novos dados. Em seguida, experimentos são apresentados com o objetivo de avaliar o desempenho desta técnica em bases de dados públicas.

5.1 *Online Sequential Fast Deep Stacked Network*

Nas Seções 2.2.1 e 2.2.2 foram apresentados dois algoritmos baseados em ELM que são capazes de lidar com novos lotes de dados à medida que eles vão sendo disponibilizados para o treinamento, o que torna possível obter um modelo inicial antes do acesso a todos os dados (quando possível), e também uma adaptação a possíveis mudanças nas distribuições destes. Além desses, outros métodos capazes de se adaptarem a novos dados e lidar com mudanças na distribuição dos dados (o chamado *concept drift*) foram apresentados nas Seções 2.2.3, 2.2.4, 2.2.5 e 2.2.6. A capacidade desses algoritmos de "atualizarem" seus parâmetros para incluir informações novas no conhecimento já adquirido pelos modelos é uma característica necessária a um algoritmo sequencial (MINKU, 2011; POLIKAR et al., 2001).

É possível notar similaridades entre os modelos sequenciais, como o OSR-ELM e o treinamento de uma FDSN. Ambos constroem um modelo inicial e vão adicionando informações na arquitetura inicial, onde a primeira técnica adiciona informações extraídas de novos dados e o segundo de dados já conhecidos, que é o conjunto fixo de amostras (porém, após um processamento em módulos anteriores).

Já o método *Online Sequential Fast Deep Stacked Network* (OSFDSN), proposto neste trabalho, envolve, de certa forma, uma combinação das técnicas FDSN com um algoritmo de treinamento sequencial, como o OSR-ELM. Ou seja, é um modelo empilhado (um *ensemble* de módulos) que tem a capacidade de adaptar a sua arquitetura quando um novo lote de dados fica disponível.

Além disso, seguindo a ideia dos métodos OWE, DOER e OEOA, as saídas dos módulos de uma OSFDSN são combinadas através de pesos para compor a saída final do *ensemble*, o que pode ser considerado um mecanismo passivo de detecção de *drift*, permitindo o método a se adaptar a conceitos diferentes. O mecanismo empregado no OSFDSN tem origem nos métodos DOER e OEOA, que são especializados em tarefas de

regressão. Portanto, o método OSFDSN também foca em tarefas de regressão. Da mesma forma que o Learn^{++}.NSE, é considerado que a saída de módulos intermediários podem contribuir para uma melhor predição em ambientes com *concept drift*.

O OSFDSN tem treinamento similar a uma FDSN: o primeiro módulo é treinado com os dados iniciais, e os próximos módulos são treinados sequencialmente utilizando entrada e saída do módulo anterior como entrada do atual, reutilizando alguns cálculos para acelerar a etapa de treinamento do método e um algoritmo baseado em ELM. No caso do OSFDSN, um algoritmo *online* como o OSR-ELM é necessário. Porém, duas principais diferenças devem ser destacadas. A primeira é que o método OSFDSN inicia criando apenas um módulo, e pode adicionar um novo módulo apenas quando receber um novo lote de dados, enquanto o FDSN cria um número fixo de módulos com os mesmos dados. A segunda diferença é que, como já mencionado, o OSFDSN utiliza uma soma ponderada da saída de seus módulos (incluindo os intermediários) ao invés de apenas utilizar a saída do último módulo da rede, como considera o FDSN.

Com estas características, o OSFDSN será capaz de reter conhecimento (conceito) antigo, nos módulos iniciais, e os últimos módulos irão reter conhecimentos mais recentes. E, através de diferentes ponderações (em instantes de tempo distintos), é possível dar um peso maior aos primeiros modelos caso um conceito antigo volte a ocorrer no futuro. Assim, os membros do *ensemble* terão pesos diretamente relacionados com os seus respectivos erros no lote de dados mais recente.

É possível dividir o treinamento do OSFDSN em duas etapas. A primeira (etapa de inicialização) envolve treinar o módulo inicial do método utilizando os dados iniciais $\boldsymbol{D}^1 = \{\boldsymbol{X}^1, \boldsymbol{y}^1\}$ (que é armazenado para uso posterior) de treinamento, de uma forma muito similar ao que é feito no FDSN. Vale notar que o peso inicial do primeiro módulo é igual a 1, ou seja, 100% da saída da rede é dada pela saída do primeiro (e único) módulo. Já a segunda (etapa online/de atualização) possui 3 passos, onde o primeiro envolve esperar por um novo conjunto de dados $\boldsymbol{D}^t = \{\boldsymbol{X}^t, \boldsymbol{y}^t\}$, com N amostras de n características e um alvo, o segunda passo avalia se o modelo irá apenas atualizar o seu módulo mais recente ou se irá adicionar um completamente novo, e no terceiro passo o algoritmo irá calcular os erros dos módulos no lote $\boldsymbol{D}^t$ e atualizar os pesos de cada módulo (tais pesos se referem a ponderação das saídas dos módulos). Finalmente, o lote armazenado é substituído pelo mais recente $\boldsymbol{D}^t$ e o procedimento volta para o passo 1 da etapa de atualização.

No segundo passo da etapa de atualização, o OSFDSN decide entre duas abordagens: atualizar o módulo mais recente ou adicionar um novo módulo. Isto pode ser a critério do usuário, onde ele pode escolher adicionar um módulo sempre que um novo lote chegar, de forma similar ao feito na técnica Learn^{++}.NSE ou nunca adicionar novos módulos, e o algoritmo funcionará como um único módulo treinado com OSR-ELM. Porém, neste trabalho é proposta uma forma "gulosa" (no sentido computacional) entre as opções, onde

o algoritmo irá considerá-las (atualizar o módulo atual ou adicionar um novo) utilizando $\boldsymbol{D}^t$, mas avaliar o melhor desempenho em cima do conjunto de validação (que será o lote armazenado $\boldsymbol{D}^{t-1}$), escolhendo aquele que obtém o melhor valor de uma métrica (por exemplo, RMSE). No caso da adição de novos módulos, candidatos com diferentes conjuntos de hiperparâmetros podem ser testados.

No terceiro passo da etapa de atualização, o OSFDSN checa o desempenho de todos os módulos no lote $\boldsymbol{D}^t$ (o mais recente), usando uma métrica de erro como a raiz do erro quadrático médio (RMSE) ou o erro quadrático médio (MSE) em tarefas de regressão de uma saída. Por exemplo, a métrica MSE do j-ésimo módulo pode ser calculada utilizando a Equação 5.1, onde $\hat{y}_i^j$ é a saída do j-ésimo módulo referente a i-ésima amostra do lote mais recente.

$$\text{MSE}_j = \sum_{i=1}^{N} (y_i - \hat{y}_i^j)^2 \tag{5.1}$$

O método constrói um vetor de erros $\mathbf{Err} = [Err_1, \ldots, Err_k]$ de acordo com as métricas obtidas em todos os k módulos (se for MSE, $Err_j = MSE_j$). Com este vetor, os pesos dos módulos são calculados usando a Equação 5.2, de forma similar ao que é feito nos métodos DOER e OEOA. Vale ressaltar que após calcular todos os j pesos w_j, eles são normalizados para que a sua soma seja igual a 1. Segundo Soares e Araújo (2015a), essa estratégia de atualização de pesos atribui peso maior aos módulos com melhor desempenho, enquanto reduz o impacto negativo dos módulos com os piores desempenhos no lote mais recente.

$$w_j = \exp\left(-\frac{\text{Err}_j - \text{median}(\mathbf{Err})}{\text{median}(\mathbf{Err})}\right) \tag{5.2}$$

Entre as vantagens de utilizar o OSFDSN estão o fato de que ele é um método que se atualiza utilizando lotes e se aproveita das vantagens do FDSN e dos algoritmos baseados em ELM, podendo adaptar sua arquitetura se julgar necessário (de acordo com o desempenho nos conceitos mais recentes, lidando com *concept drift*) com baixo custo de memória e tempo reduzido de treinamento. Uma desvantagem de se utilizar o OSFDSN é a impossibilidade de podar módulos, devido ao fato de ser uma abordagem empilhada (para processar o k-ésimo módulo é necessário ter a informação de todos os módulos anteriores), sendo possível podar apenas o último módulo da rede. Além disso, se um número máximo de módulos não for definido, o método pode crescer indefinidamente ao longo do tempo. Vale notar, porém, que devido a construção do FDSN, a adição de um módulo tem reduzido impacto computacional (SILVA et al., 2020a) devido ao reaproveitamento de pesos.

O Algoritmo 4 resume como um modelo OSFDSN é treinado, e o Algoritmo 5 mostra como a saída de um modelo OSFDSN é obtida. A Figura 29 apresenta uma ilustração de um modelo OSFDSN. Vale notar que OSFDSN com um número máximo de módulos $S = 1$ é uma SLFN única treinada com o método baseado no ELM escolhido.

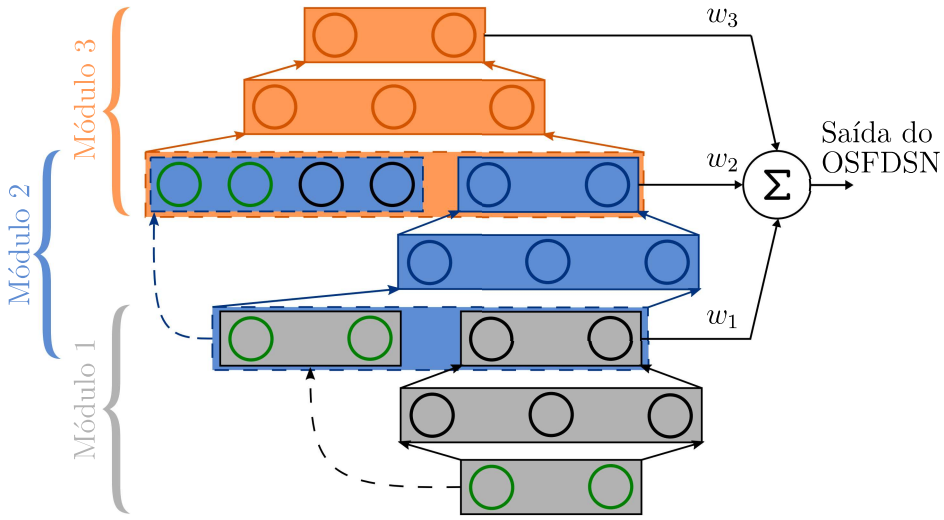
Algoritmo 4: Algoritmo de treino do OSFDSN

Requer: Um algoritmo *online* baseado em ELM (OSR-ELM, por exemplo) e seus hiperparâmetros; o número máximo de módulos S; o primeiro lote de dados $\boldsymbol{D}^1 = (\boldsymbol{X}^1, \boldsymbol{y}^1)$; A métrica de erro Err.
Etapa de Inicialização:
    t=1; $\boldsymbol{D}^V = \boldsymbol{D}^1$;
    Treina o primeiro módulo usando o método baseado em ELM escolhido e $\boldsymbol{D}^1$.
Etapa de atualização:
repita
    Passo 1:
        Espera o novo lote $\boldsymbol{D}^{t+1} = (\boldsymbol{X}^{t+1}, \boldsymbol{y}^{t+1})$ chegar.
    Passo 2:
        Copia o modelo atual e atualiza a cópia usando a etapa de atualização do algoritmo online escolhido e o lote $\boldsymbol{D}^{t+1}$. Estima a saída do módulo atualizado com relação ao lote de validação $\boldsymbol{D}^V$ e obtém a métrica Err_{upd}.
        Se *o número atual de módulos for menor que* S, **faça:**
            Copia o modelo atual e adiciona um novo módulo ao modelo copiado usando a abordagem do FDSN e $\boldsymbol{D}^{t+1}$. Estima a saída do novo módulo quanto ao lote de validação $\boldsymbol{D}^V$ e obtém a métrica Err_{new}. Nesse passo, diferentes novos módulos podem ser treinados (por exemplo, utilizando hiperparâmetros diferentes) caso diferentes cópias sejam feitas. A métrica a ser armazenada Err_{new} sempre é a melhor métrica obtida entre as opções.
        se não: Faça $Err_{new} = \infty$ (pior caso da métrica Err)
        Atualiza o modelo atual com a opção que obteve a melhor métrica entre Err_{new} or Err_{upd} (por exemplo, o menor MSE), descartando as demais opções.
    Passo 3:
        Calcula o erro de todos os módulos com respeito ao lote mais recente $\boldsymbol{D}^{t+1}$ e constrói o vetor $\mathbf{Err} = [Err_1, \ldots, Err_k]$. Calcula os pesos de todos os módulos usando a Equação 5.2 e os normaliza, de modo que $\sum_{i=1}^{k} w_i = 1$.
    Salva o lote atual como lote de validação: $\boldsymbol{D}^V = \boldsymbol{D}^{t+1}$; $t = t + 1$;
até *que o método seja terminado*;

Algoritmo 5: Algoritmo de predição do OSFDSN

Requer: Modelo OSFDSN com k módulos e dados de entrada $\boldsymbol{X}^t$
Ache, para todos os k módulos, o vetor de saídas $\boldsymbol{y}_k^t$ referente ao conjunto de entrada $\boldsymbol{X}^t$.
Obtém a soma ponderada dos vetores $\boldsymbol{y}_k^t$ de todos os módulos: $\hat{\boldsymbol{y}}^t = \sum_{i=1}^{k} w_i \boldsymbol{y}_i^t$.
Retorna a saída estimada $\hat{\boldsymbol{y}}$

Figura 29 – Exemplo de arquitetura do OSFDSN. Retângulos preenchidos pela mesma cor representam camadas do mesmo módulo (uma SLFN), e círculos na cor verde representam os dados de entrada originais. É possível notar que a entrada e saída do primeiro módulo são usadas para construir a entrada do segundo módulo. Considerando que cada módulo tem uma saída $y_k \in \mathbb{R}$ referente a uma amostra $\boldsymbol{x}_k$, a saída do método OSFDSN é dada por $\hat{y} = \sum_{i=1}^{k} w_i y_i$, onde $\sum_{i=1}^{k} w_i = 1$.



Fonte: Produção do próprio autor.

5.2 Experimentos Considerando Métodos *Online*

Nesta seção é feita uma avaliação do método OSFDSN em diferentes situações. Na Seção 5.2.1 são apresentados os conjuntos de dados, métricas e testes estatísticos utilizados, especificação dos parâmetros dos métodos e alguns detalhes de implementação e execução. Na Seção 5.2.2 é feita uma análise das técnicas propostas considerando bases de dados *online* com diferentes tamanhos de lotes, isto é, as amostras chegam em conjuntos com diferentes tamanhos. Na Seção 5.2.3 são feitos experimentos e comparações das técnicas propostas em problemas de regressão com saídas unidimensionais, utilizando métricas e testes de hipótese para avaliação.

5.2.1 Metodologia dos Experimentos

Exceto quando explicitado, os conjuntos de dados de problemas de regressão utilizados para avaliação das técnicas *online* propostas neste trabalho são os apresentados na Tabela 15. Todas os conjuntos de dados são públicos e foram obtidos de problemas reais, disponíveis em repositórios como o Keel (ALCALÁ-FDEZ et al., 2011), UCI (DUA; GRAFF, 2017) e *Weka Datasets* (HALL et al., 2009), e já foram utilizados em trabalhos que envolvem métodos *online* de aprendizado de máquina.

Tabela 15 – Bases de dados de problemas de regressão utilizados nos experimentos *online*, ordenados pelos seus respectivos tamanhos. As colunas mostram o número de amostras N da base, número de atributos d e uma indicação de um trabalho em que elas foram utilizadas em uma abordagem de aprendizado *online*.

Conjunto	#Amostras	#Atributos	Referência
debutanizer	1836	7	(SOARES; ARAÚJO, 2015a) (SOARES; ARAÚJO, 2015b) (SOARES; ARAÚJO, 2016)
FriedmanGRA	2000	10	(SOARES; ARAÚJO, 2015a) (SOARES; ARAÚJO, 2015b) (SOARES; ARAÚJO, 2016)
FriedmanGnRG	2000	10	(SOARES; ARAÚJO, 2015a) (SOARES; ARAÚJO, 2015b) (SOARES; ARAÚJO, 2016)
FriedmanLA	2000	10	(SOARES; ARAÚJO, 2015a) (SOARES; ARAÚJO, 2015b) (SOARES; ARAÚJO, 2016)
Hyperplane	2000	10	(SOARES; ARAÚJO, 2015a) (SOARES; ARAÚJO, 2015b) (SOARES; ARAÚJO, 2016)
quake	2175	3	*Weka Datasets*
Abalone	4177	8	(IKONOMOVSKA; GAMA; DZEROSKI, 2014)
Winequality	4898	11	(IKONOMOVSKA; GAMA; DZEROSKI, 2014)
Wind	6574	14	(IKONOMOVSKA; GAMA; DZEROSKI, 2014)
Bank8fm	8192	8	(DUARTE; GAMA; BIFET, 2016)
Kin8nm	8192	8	(DUARTE; GAMA; BIFET, 2016)
Puma32h	8192	32	(DUARTE; GAMA; BIFET, 2016)
SRU1	10000	5	(SOARES; ARAÚJO, 2015a) (SOARES; ARAÚJO, 2015b) (SOARES; ARAÚJO, 2016)
SRU2	10000	5	(SOARES; ARAÚJO, 2015a) (SOARES; ARAÚJO, 2015b) (SOARES; ARAÚJO, 2016)
Pol	15000	48	(DUARTE; GAMA; BIFET, 2016)
Elevators	16599	18	(DUARTE; GAMA; BIFET, 2016)
House_16H	22784	16	(DUARTE; GAMA; BIFET, 2016)
MV	40768	10	(DUARTE; GAMA; BIFET, 2016)

Fonte: Produção do próprio autor.

Vale notar que foram considerados apenas bases de regressão de saída única devido a restrições de implementação das técnicas incluídas nos experimentos. No caso, a base SRU possui duas saídas, mas cada saída foi considerada separadamente nas bases SRU1 e SRU2.

As métricas e os testes estatísticos utilizados são os mesmos apresentados na Seção 3.4.1.2. Como métricas de avaliação, foram utilizados o RMSE, o tempo de treinamento e a quantidade de memória necessária para armazenar os modelos. Já os testes de Friedman e Nemenyi foram utilizados nas análises estatísticas dos resultados.

5.2.1.1 Detalhes da Implementação e Ambiente de Execução

Todos os métodos utilizados nos testes desta seção foram implementados no software MATLAB 9.4 e executados em um computador com sistema operacional Ubuntu 18.04.3 LTS com processador Intel® Core™ i9-9900KF (8 núcleos a 3.6 GHz) com 64 GB de memória RAM. Nenhum teste foi realizado com a ajuda de placas de processamento gráfico.

Como o teste agora envolve métodos *online*, a metodologia de um experimento deve ser adequada a bases de dados cujas amostras chegam ao longo do tempo. O esquema de avaliação escolhido foi o chamado de *Prequential Evaluation Method* (ou "*Test-then-train*"), usado em diversos trabalhos de métodos *online*, como (SOARES; ARAÚJO, 2015a; SOARES; ARAÚJO, 2015b; SOARES; ARAÚJO, 2016). Essa abordagem considera que a base de dados terá lotes de tamanho fixo m (no caso deste trabalho, as bases de dados foram divididas sequencialmente), que serão enviados ao algoritmo de treinamento um por um. Todos os lotes são normalizados para que as suas m amostras tenham média zero e desvio padrão igual a um. Quando o algoritmo recebe o lote $\boldsymbol{D}^t = \{\boldsymbol{X}^t, \boldsymbol{y}^t\}$, o método prediz uma saída $\hat{\boldsymbol{y}}^t$ a partir de $\boldsymbol{X}^t$, que é comparada com a $\boldsymbol{y}^t$ através de uma métrica (quanto ao OSFDSN, esses passos são feitos entre as linhas 8 e 10 do Algoritmo 4). Esta métrica é armazenada, e $\boldsymbol{D}^t$ é utilizado para que o método atualize seus parâmetros, adquirindo novo conhecimento. Após processar todos os lotes, a métrica final de um método em uma base de dados é a média das métricas obtidas em cada um dos lotes recebidos. Todos os experimentos foram repetidos 10 vezes com diferentes sementes aleatórias iniciais, obtidas a partir de uma semente aleatória única. Vale notar que as bases de dados são divididas antes das repetições, então em todas as execuções as mesmas condições de treinamento são fornecidas aos métodos.

Exceto quando explicitado, 11 métodos foram considerados nos experimentos deste capítulo, através da combinação de arquiteturas, algoritmos de treinamento e técnicas de inicialização de pesos. Tais métodos são apresentados no Quadro 6. Vale ressaltar que entre os métodos considerados, estão o OWE, DOER, OEOA e Learn^{++}.NSE, que possuem implementação em MATLAB disponibilizada[1], e duas variantes do OSFDSN. A primeira delas, chamada de OSFDSN-v, é similar ao descrito no Algoritmo 4 com uma etapa inicial de validação para definir os hiperparâmetros, onde são considerados o método OSR-ELM para treinar uma SLFN com L neurônios ocultos. Na etapa de inicialização, o OSFDSN-v

[1] ⟨https://sites.google.com/site/symonesoares/home/codes-data-sets⟩

utiliza validação 5-*fold* no primeiro lote para determinar o melhor valor de C. Caso ele considere a possibilidade de adicionar um novo módulo ao modelo, a validação *hold-out* é utilizada para definir o melhor valor de C do algoritmo OSR-ELM, usando o lote $\boldsymbol{D}^t$ como conjunto de treino e $\boldsymbol{D}^V$ como conjunto de validação (neste caso, cada módulo pode possuir um valor de C). Após atualizar o modelo (linha 14 do Algoritmo 4), seja atualizando o último módulo ou adicionando um novo, os pesos de saída[2] do OSFDSN também serão atualizados. Este procedimento pode ser visto no Algoritmo 4. Uma variante do OSFDSN similar ao OSFDSN-v também foi considerada, onde a validação cruzada para definir o valor de C é feita apenas no primeiro lote, permanecendo fixo (nos módulos seguintes) enquanto o método existir. Este método é chamado de OSFDSN-v-i. Os códigos utilizados neste trabalho podem ser consultados em ⟨https://doi.org/10.17632/y5ytz6kydf.1⟩

Quadro 6 – Métodos considerados nos experimentos realizados.

Nome Associado	Arquitetura	Algoritmo de Treinamento	Inicialização de Pesos
OSFDSN-v_{rfm}	OSFDSN	OSFDSN-v + OSR-ELM	rfm
OSFDSN-v_{ae}			ae
OSFDSN-v_{svd}			svd
OSFDSN-v-i_{rfm}		OSFDSN-v-i + OSR-ELM	rfm
OSFDSN-v-i_{ae}			ae
OSFDSN-v-i_{svd}			svd
Learn^{++}.NSE	Learn^{++}.NSE	SIMPLS	-
OWE	OWE		-
DOER	DOER	OSELM	-
OEOA	OEOA		-
kNN	-	kNN	-

Fonte: Produção do próprio autor.

5.2.1.2 Especificação dos Hiperparâmetros

Os métodos OWE e Learn^{++}.NSE usam como método base (isto é, cada um dos membros dos seus ensembles é treinado usando este método) o algoritmo *Straightforward Implementation of Partial Least Square Regression* (SIMPLS) (JONG, 1993), que é uma abordagem alternativa a regressão por mínimos quadrados parciais. Já DOER e OEOA utilizam o algoritmo OS-ELM como método base. Eles possuem um número máximo de modelos no ensemble igual a $S = 15$, e utilizam um lote de tamanho $m = 20$ (exceto nos testes da Seção 5.2.2, que consideram diferentes valores para m). Os métodos que utilizam o OS-ELM usam $L = 20$ neurônios e função de ativação sigmoide em cada membro do ensemble. Embora OWE, DOER e OEOA sejam métodos que analisam amostra por amostra, eles armazenam uma janela de amostras de tamanho m para que elas sejam

[2] Para evitar confusão, "pesos de saída" se referem aos pesos que multiplicam as saídas dos módulos para encontrar a saída final do método OSFDSN.

utilizadas para retreinar os métodos, caso necessário. Todos estes parâmetros foram definidos nas implementações originais disponibilizadas[3].

Com relação ao método OSFDSN, os mesmos valores de m e L utilizados pela implementação do OWE, DOER e OEOA foram considerados, exceto quando explicitado. Como o i-ésimo módulo do OSFDSN depende do módulo anterior $(i-1)$, a substituição de módulos intermediários não é possível, o que limita a uma substituição de apenas do último módulo ou que todo o modelo seja retreinado. Nenhuma dessas duas abordagens é desejada para um método sequencial, já que no primeiro caso há uma perda de informação de conceitos recentes, e no segundo caso toda a informação já aprendida é descartada. Ou seja, se o número máximo de módulos das técnicas baseadas em OSFDSN for limitado, todos os conceitos que aparecem depois do modelo atingir o número máximo obrigatoriamente estariam "incorporados" no último módulo, o que também pode não ser uma boa abordagem para lidar com conceitos antigos e/ou recorrentes. Assim, limitar o valor S do OSFDSN pode afetar consideravelmente o seu desempenho em situações não estacionárias. Portanto, os métodos baseados em OSFDSN irão possuir $S = \infty$. Isto, porém, pode afetar o uso de memória do método e o seu tempo de treinamento, já que não há um limite para o tamanho dos *ensembles*, mesmo que o custo de memória para adicionar um novo módulo seja pequeno devido ao aproveitamento de pesos da técnica. Vale ressaltar que as demais técnicas consideradas possuem a capacidade de substituir módulos e possuem limitação de módulos, conforme as propostas de seus respectivos autores.

O método OSFDSN utiliza o algoritmo OSR-ELM, que necessita de um hiperparâmetro de regularização C, escolhido do conjunto $\{2^{-20}, 2^{-19}, \ldots, 2^{20}\}$, e uma função de ativação, escolhida entre as funções sigmoide, *hard-limit*, gaussiana e multiquádrica, ambos escolhidos na etapa de validação cruzada. Vale destacar que o OSR-ELM é uma versão mais robusta do OS-ELM (HUYNH; WON, 2011; SHAO; ER, 2016), e por isso foi escolhida como algoritmo de treinamento do OSFDSN. No caso do OSFDSN-v-i, a função é definida apenas no primeiro lote e utilizada ao longo de todo o modelo. Já no OSFDSN-v, a função de ativação de um módulo adicionado pode mudar, a depender do novo módulo adicionado, que é escolhido através de validação cruzada. Assim, o *grid* de hiperparâmetros testado nas etapas de validação cruzada é o mesmo utilizado nos experimentos do Capítulo 3. Quanto ao método kNN, o valor de k foi escolhido entre os valores $\{3, 5, 7, 9, 11, 13, 15\}$, também utilizando validação cruzada (no primeiro lote).

5.2.2 Análise considerando tamanhos de lotes distintos

Nesta seção são apresentados experimentos dos métodos OSFDSN-v_{rfm}, OSFDSN-v-i_{rfm}, OSFDSN-v_{ae}, OSFDSN-v_{svd}, OSFDSN-v-i_{ae} e OSFDSN-v-i_{svd}, descritos na Seção 5.2.1.1,

[3] ⟨https://sites.google.com/site/symonesoares/home/codes-data-sets⟩

para diferentes tamanhos de lotes de amostras, sempre com 20 neurônios ocultos por módulo.

Os resultados de quatro bases de dados da Tabela 15 foram selecionados para este experimento. Duas das bases, *FriedmanGRA* e *Hyperplane*, são bases de dados sintéticas com duas mil amostras e com ocorrência de *concept drift*. A primeira delas possui *drift* do tipo global, recorrente e abrupto, enquanto a segunda possui *drift* gradual e não recorrente, além de ruído (SOARES; ARAÚJO, 2015a). Já as outras duas bases, *Elevators* e *SRU1*, são bases de dados de problemas reais, sem informações sobre *concept drift*.

Cada base de dados escolhida foi dividida em lotes de tamanho fixo, e testada pelos 6 métodos citados. Os tamanhos de lote utilizados foram 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300, 350, 400, 450 e 500 amostras. Desta forma, é possível analisar o desempenho das técnicas em métricas distintas nestes tamanhos. Vale lembrar que, quanto maior o tamanho do lote, menor a capacidade de adaptação dos modelos a variações mais bruscas nos conceitos da base de dados, menor o tempo de treinamento (menos lotes processados) e menor o uso de memória das técnicas testadas (menos possibilidades de adição de novos módulos).

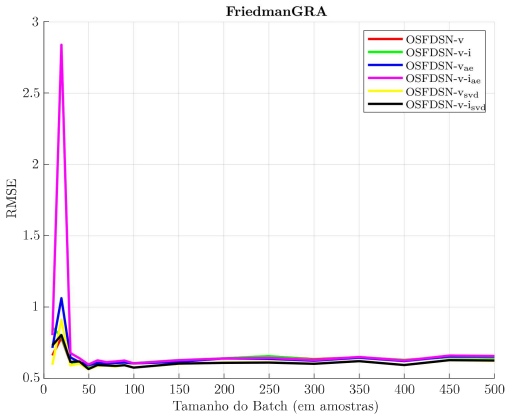
As métricas de RMSE, tempo de treinamento e número de bytes utilizados para armazenar tais técnicas são apresentadas nas Figuras 30, 32 e 33, respectivamente. Os métodos OSFDSN-v_{rfm}, OSFDSN-v-i_{rfm}, OSFDSN-v_{ae}, OSFDSN-v-i_{ae}, OSFDSN-v_{svd} e OSFDSN-v-i_{svd} são representados pelas cores vermelha, verde, azul, rosa, amarela e preta, respectivamente.

Com relação ao RMSE, é possível ver dois comportamentos distintos. O primeiro, que ocorreu nas bases *FriedmanGRA* e *Elevators*, em que os maiores valores de erro são obtidos utilizando tamanhos de lotes pequenos, enquanto os erros decrescem com o aumento do número de amostras. O segundo comportamento é observado nas bases *Hyperplane* e *SRU1*, que envolve um erro elevado com lotes muito pequenos, porém há uma tendência de crescimento da métrica com o aumento do tamanho do lote.

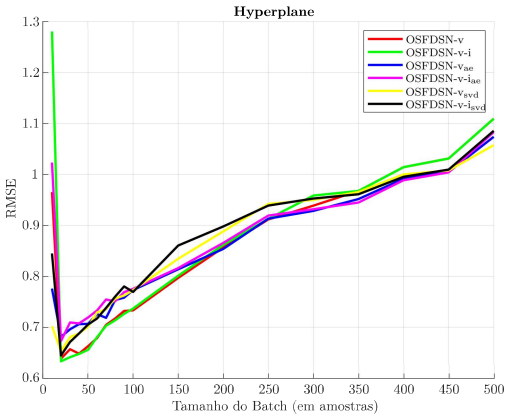
É possível que esses dois comportamentos sejam causados pelas naturezas distintas das bases de dados. Considerando a base de dados *Hyperplane*, que possui *drift* gradual e, portanto, o seu conceito varia ao longo do tempo, processar mais amostras por vez fez com que o método não se adaptasse a tempo, elevando a métrica de erro. Já na base de dados *FriedmanGRA*, que possui *drift* abrupto e recorrente, não houve uma grande variação na métrica de erro a partir de 50 amostras por lote. O *drift* desta base de dados é abrupto mas ocorre apenas duas vezes ao longo da base de dados, o que pode significar que as técnicas foram capazes de se adaptar logo que o *drift* ocorreu. Comportamentos similares aos encontrados nas bases *Hyperplane* e *FriedmanGRA*, que são sintéticas e cujo *drift* é conhecido, foram obtidos nas bases de dados *SRU1* e *Elevators*, respectivamente.

Figura 30 – Métrica RMSE obtida nos métodos propostos para diferentes tamanhos de lote[4].

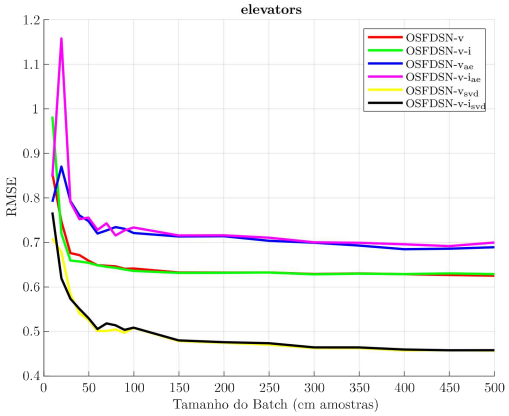
(a) Base FriedmanGRA



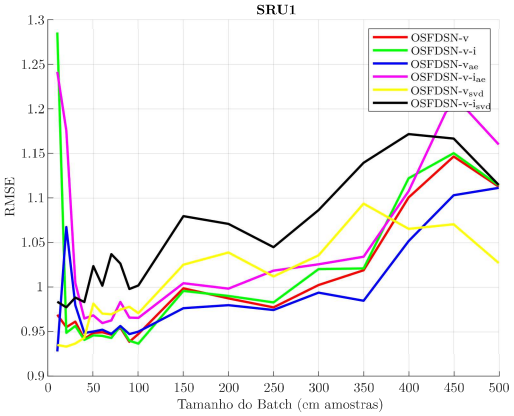
(b) Base Hyperplane



(c) Base Elevators



(d) Base SRU1



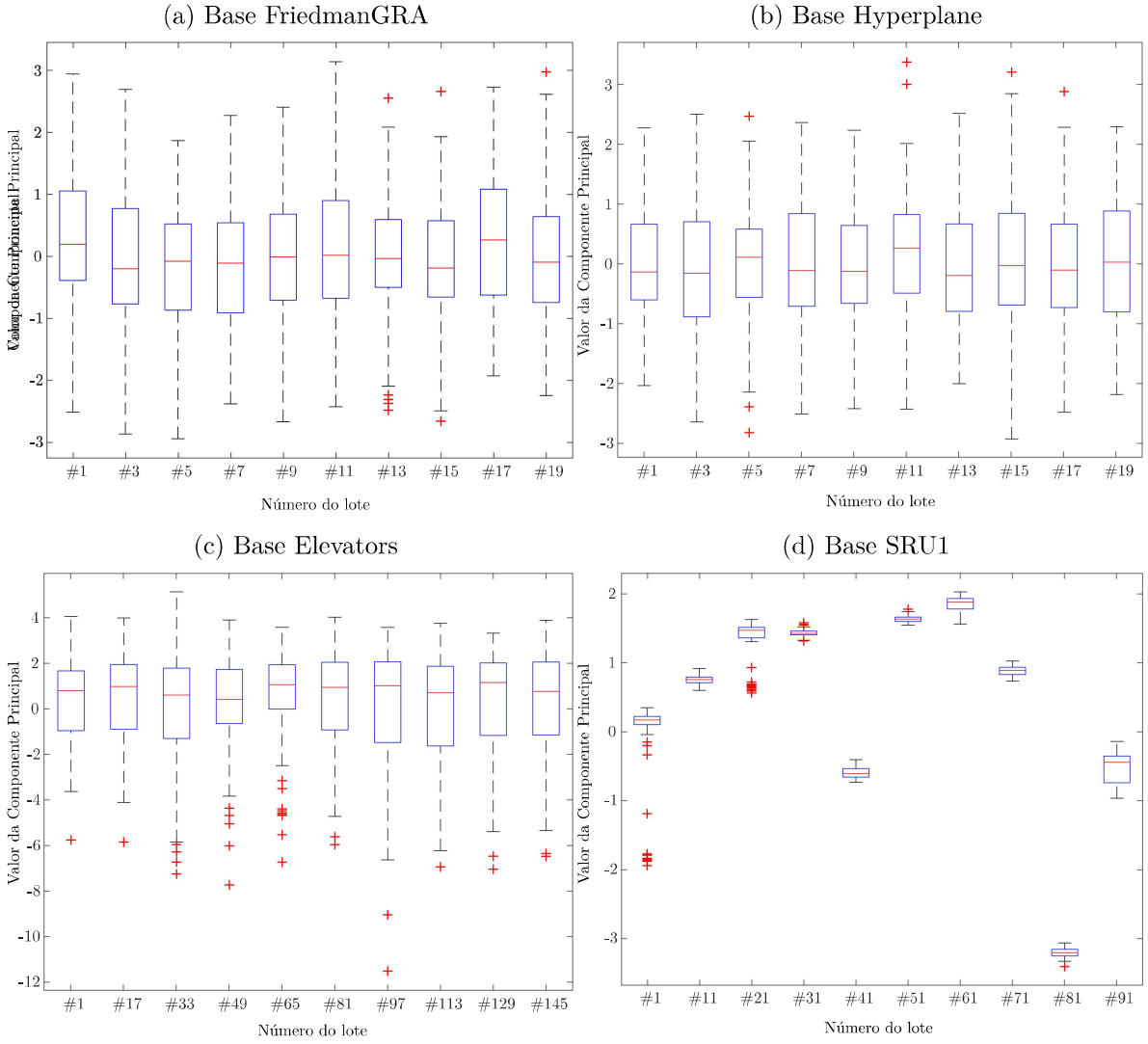
Fonte: Produção do próprio autor

O fato de usar lotes pequenos envolve a criação de modelos com pouca informação aprendida, já que eles devem extrair o seu conhecimento desse pequeno número de amostras, que podem não representar bem a base de dados e obter modelos inadequados devido à pequena quantidade de amostras. Assim, erros elevados foram obtidos em ambos os comportamentos. Já a utilização de lotes grandes proporciona muitas amostras para cada modelo (incluindo o modelo inicial) extrair informações, além dos dados possivelmente representarem melhor o problema abordado. Isto os torna mais robustos (aprendem melhor o conceito), mas também reduzem a sua velocidade de adaptação a novos conceitos. Numa situação prática, esperar um lote chegar por inteiro pode levar um longo período de tempo, sendo que o método não se atualizaria neste período.

Como dois comportamentos distintos foram observados, um experimento foi feito para avaliar o quão dinâmicas as bases de dados analisadas são. Para isso, 10 lotes igualmente espaçados (isto é, o primeiro, o lote localizado aproximadamente em 10% do total de amostras da base de dados, o próximo em 20%, e assim por diante) e com

100 amostras distintas foram analisados utilizando Análise de Componentes Principais. Assim como no Capítulo 4, um experimento foi realizado para analisar as características dinâmicas da base de dados. A componente principal da base de dados foi extraída utilizando todos os dados, e cada um dos lotes foi projetado para a base vetorial composta apenas desta componente, o que resulta em um vetor de 100 números por lote. Os *boxplots* da componente principal de cada lote são apresentados na Figura 31.

Figura 31 – Análise de Componentes Principais nas bases de dados analisadas.



Fonte: Produção do próprio autor

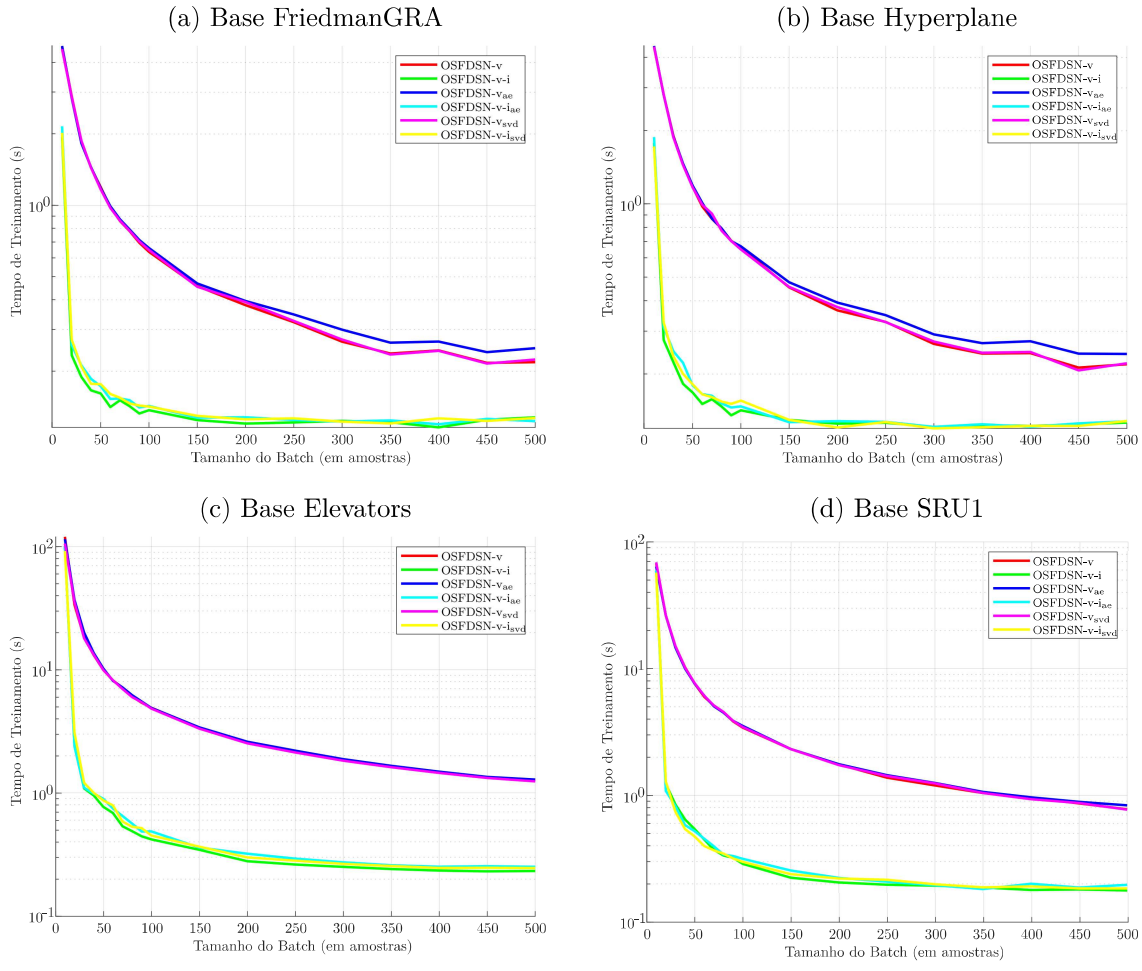
Com relação as bases de dados FriedmanGRA e Elevators, é possível notar que as componentes principais dos lotes permanecem aproximadamente com as mesmas distribuições, como mostram as Figuras 31a e 31c, respectivamente. Os métodos analisados necessitaram de um valor mínimo de amostras para aprender sobre tais bases de dados, e o erro permaneceu constante caso o tamanho do lotes aumentasse.

Já para as bases de dados Hyperplane e SRU1, é possível notar que a componente principal da base Hyperplane obteve um comportamento similar ao das bases FriedmanGRA

e Elevators, enquanto a componente principal da base SRU1 passou por diversas alterações em sua distribuição. Como a base SRU aparenta variar suas características de forma acentuada, a tendência é que as técnicas apresentem erros maiores quando utilizadas em conjunto com lotes maiores, como foi o caso apresentado na Figura 30d. É possível, também, que a base Hyperplane apresente características mais dinâmicas em outras componentes principais, ou ainda utilizando outras formas de redução de dimensionalidade.

No que diz respeito ao tempo de treinamento dos métodos e número de bytes ocupados pelos modelos, o mesmo comportamento foi obtido para as 4 bases de dados com pequenas oscilações, como pode ser visto na Figura 32. Quanto menos amostras por lote, e consequentemente mais lotes, maior o número de atualizações dos pesos e possibilidades de adição de novos módulos ao *ensemble* dos métodos. Quanto mais módulos, mais processamento e mais bytes são requeridos pelo modelo. Assim, uma espécie de decaimento cujo comportamento é similar a uma exponencial negativa foi obtido nas duas métricas, caindo rapidamente com o aumento do tamanho do lote.

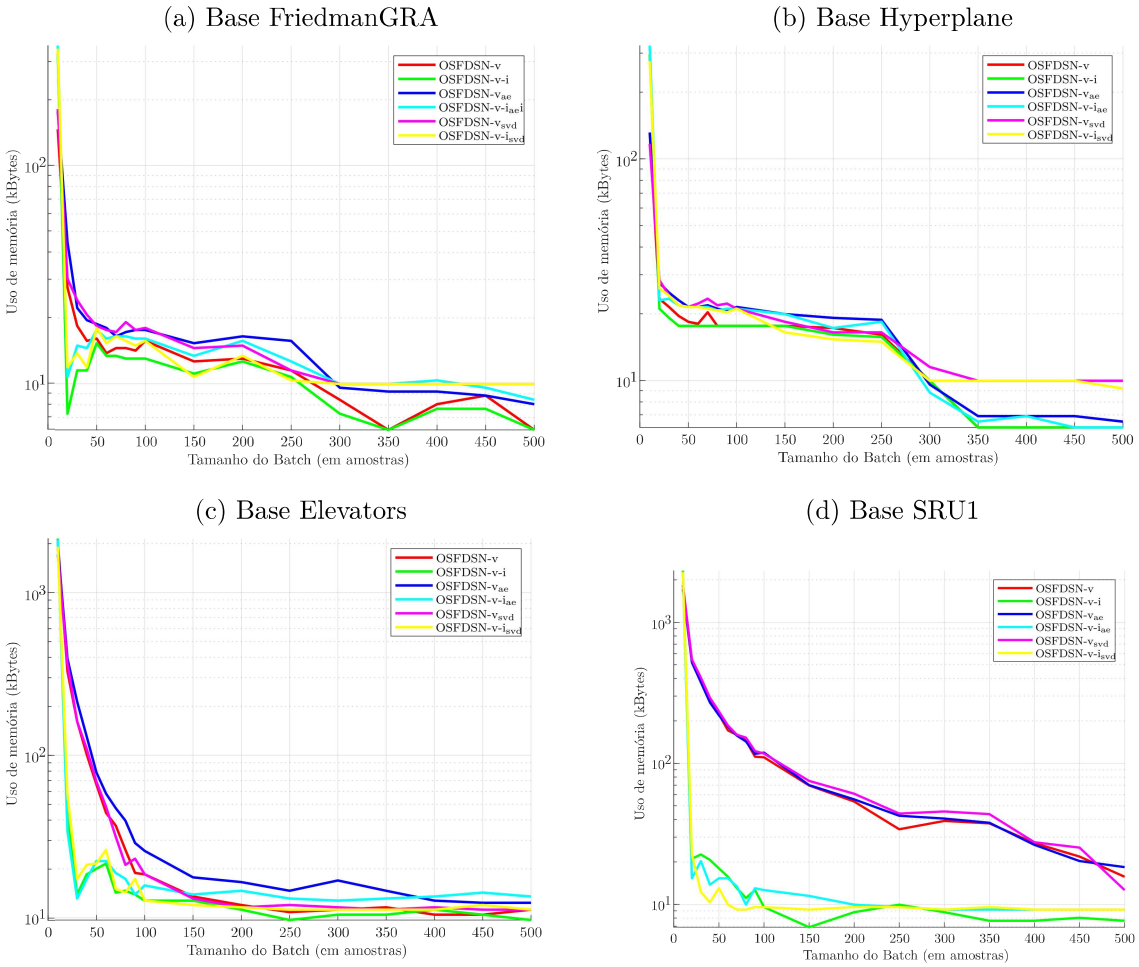
Figura 32 – Tempo de treinamento dos métodos propostos para diferentes tamanhos de lote. Para melhor visualização, o eixo vertical está em escala logarítmica.



Fonte: Produção do próprio autor

Numericamente, os métodos OSFDSN-v levaram cerca de 5 segundos para processar as bases sintéticas de 2000 amostras divididas em lotes de tamanho 10 e cerca de 1 segundo para processar estas mesmas bases, mas divididas em lotes de 50 amostras. Após 150 amostras por lote, houve pouca variação de tempo de processamento. Já para aa bases *Elevators* e *SRU1*, cerca de 100 e 60 segundos, respectivamente, foram necessários quando lote de tamanho 10 foi considerado, e 10 e 7,5 segundos, respectivamente, para as mesmas bases divididas em lotes de tamanho 50.

Figura 33 – Número de bytes para armazenar os métodos propostos para diferentes tamanhos de lote. Para melhor visualização, o eixo vertical está em escala logarítmica.



Fonte: Produção do próprio autor

Considerando o uso de memória, os métodos ocuparam cerca de 350 *kilobytes* nas bases de dados sintéticas de 2000 amostras com tamanho de lote 10, mas quando este valor aumenta para 20 a 30 amostras, o tamanho dos modelos se reduz a cerca de 25 *kilobytes*, permanecendo próximo a este valor para lotes maiores. Já para as bases com mais amostras, um comportamento similar foi obtido, onde um tamanho de lote próximo a 30 atinge um uso de memória de cerca de 10% do uso de memória da mesma técnica na

mesma base de dados, mas dividida em lotes de 10 amostras. Vale notar que o tamanho do modelo é diretamente proporcional ao número de módulos adicionados aos métodos.

Nos testes realizados, foi possível observar dois comportamentos distintos com relação ao erro quadrático médio nas bases de dados. Em um deles foi observado que o tamanho do lote não altera muito a métrica, desde que um número mínimo de amostras seja utilizado, e no outro foi notado que lotes de tamanho maior implicam em um maior erro. No que diz respeito ao tempo de treinamento e uso de memória, o comportamento observado era esperado, pois quanto mais amostras por lote, e consequentemente um número menor de lotes, menos intervenções na arquitetura dos métodos baseados no OSFDSN são feitas, o que reduz o tempo de treinamento e uso de memória das técnicas.

5.2.3 Análise por Teste de Hipótese

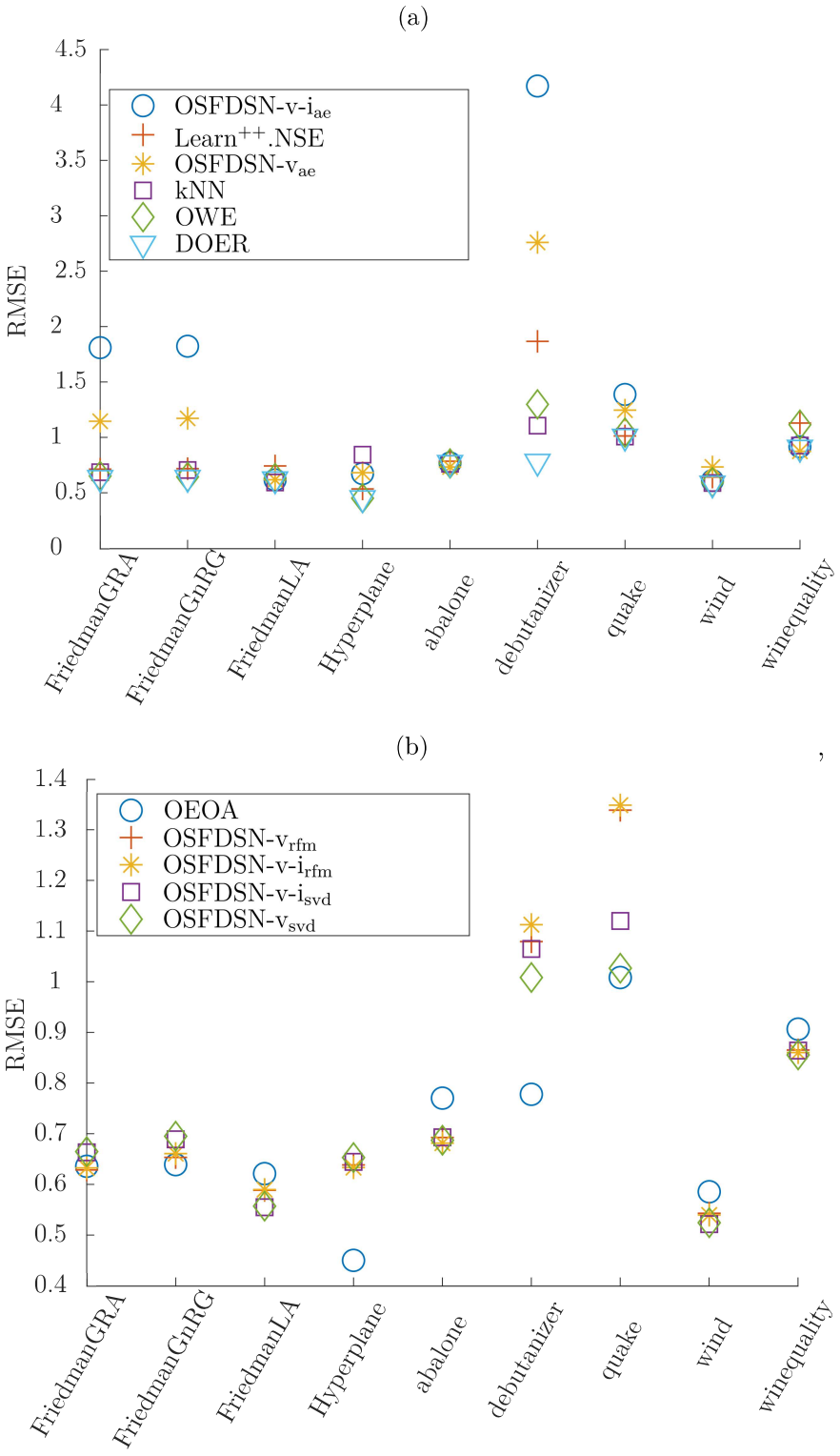
5.2.3.1 Considerando Bases Menores e os métodos DOER e OEOA

Nesta seção são apresentados os resultados dos métodos citados na Seção 5.2.1.1 nas bases de dados de até 8000 amostras da Tabela 15. Restringindo a estas bases, os métodos DOER e OEOA foram incluídos nos testes. A utilização destas técnicas em bases de dados com grandes quantidades de amostras é dificultada devido ao elevado tempo de processamento das implementações dos métodos, mesmo em computadores modernos e, portanto, os testes com elas foram limitados a bases pequenas. Devido a implementação do DOER e OEOA utilizarem apenas a função de ativação sigmoide, os métodos baseados em OSFDSN foram limitados a esta função neste teste.

Os valores de RMSE e os hiperparâmetros encontrados nos experimentos realizados podem ser vistos no Apêndice D. Destaca-se que não é possível indicar um conjunto único de hiperparâmetros para as execuções dos métodos, devido a natureza dinâmica dos mesmos. Ou seja, o hiperparâmetro de regularização obtido nos módulos do método OSFDSN-v podem ser diferentes devido a utilização do k-fold. Já para os métodos OSFDSN-v-i, embora utilizem apenas um conjunto de hiperparâmetros em todos os membros do ensemble, eles podem ser diferentes em diferentes execuções (com sementes aleatórias distintas). Assim, apenas o número médio de módulos destas técnicas é apresentado na Tabela 40 do Apêndice D. Os hiperparâmetros utilizados nas demais técnicas foram os disponibilizados nas implementações fornecidas.

A Tabela 16 mostra os *ranks* médios obtidos por cada método referentes à métrica RMSE, assim como os valores médios desta métrica. É possível notar que uma métrica média menor não implica em um melhor *rank*, já que podem ter sido obtidos *outliers* que deterioraram a métrica média, mas não o *rank*. Os gráficos das RMSE obtidas podem ser vistos na Figura 34, onde valores fora do intervalo apresentado foram ocultados para uma melhor visualização.

Figura 34 – RMSE obtidas nos experimentos com problemas de regressão *online* para bases de dados de até 8000 amostras.



Fonte: Produção do próprio autor.

Tabela 16 – *Ranks* médios ($\overline{\text{Rank}}$) e métrica RMSE no formato "média (desvio padrão)" dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras. Os métodos estão ordenados pelo *rank* médio, de forma crescente.

Método	$\overline{\text{Rank}}$	RMSE	Mediana do RMSE
OSFDSN-v$_{\text{svd}}$	4,0000	0,74 (0,17)	0,6871
OSFDSN-v-i$_{\text{svd}}$	4,1667	0,76 (0,20)	0,6889
OEOA	4,1667	0,71 (0,16)	0,6390
OSFDSN-v-i$_{\text{rfm}}$	4,3333	0,78 (0,26)	0,6532
OSFDSN-v$_{\text{rfm}}$	4,3889	0,78 (0,25)	0,6608
DOER	4,5000	0,71 (0,16)	0,6357
OWE	6,3333	0,80 (0,27)	0,6517
kNN	8,1111	0,83 (0,19)	0,7615
OSFDSN-v$_{\text{ae}}$	8,3333	1,11 (0,62)	0,8741
Learn^{++}.NSE	8,4444	0,90 (0,38)	0,7422
OSFDSN-v-i$_{\text{ae}}$	9,2222	1,42 (1,08)	0,9158

Fonte: Produção do próprio autor.

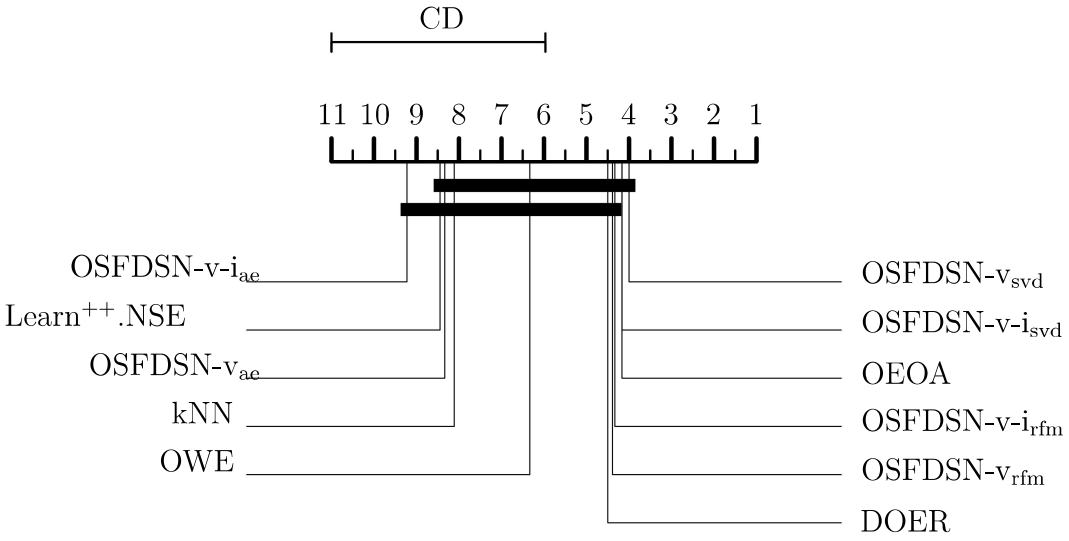
Os resultados apresentados na Tabela 16 mostram que, para problemas de regressão *online*, os métodos baseados em OSFDSN com inicialização ortogonal de pesos atingiram os melhores *ranks*, seguido pelos métodos baseados em OSFDSN com inicialização aleatória de pesos (empatados com o OSFDSN-v-i$_{\text{svd}}$) e pelo método OEOA. Este resultado pode indicar que os métodos empilhados propostos neste trabalho também podem ser adequados para utilização em problemas de regressão *online*. No caso deste experimento, a utilização de métodos de inicialização de pesos melhorou o *rank* médio, quando o método ortogonal é utilizado, mas também piorou consideravelmente, quando a inicialização por *autoencoder* foi utilizada. As demais técnicas utilizadas como referência obtiveram *ranks* intermediários.

Vale notar que as médias dos RMSE não foram muito distantes um dos outros. O mesmo aconteceu com as medianas, que também podem ser conferidas na Tabela 16. A maior mediana, obtida pelo método OSFDSN-v-i$_{\text{ae}}$, foi cerca de 48% maior que a menor mediana, obtida pelo OEOA.

O teste estatístico de Friedman (DEMŠAR, 2006) foi aplicado nos resultados obtidos, considerando uma hipótese de que os métodos são equivalentes, usando um nível de significância de 5%. O teste obteve um valor-$p = 6{,}78 \cdot 10^{-5}$, rejeitando a hipótese. Assim, o teste de Nemenyi foi utilizado, cujo resultado pode ser visto na Figura 35. A distância crítica, neste caso, foi de 5,0328, ou seja, métodos cuja diferença de *ranks* é maior que este valor não são equivalentes. O teste indica que o método OSFDSN-v-i$_{\text{ae}}$ não é estatisticamente similar ao método OSFDSN-v$_{\text{svd}}$, que tem melhor *rank*. Os demais foram considerados estatisticamente equivalentes.

Com a finalidade de comparar o desempenho entre métodos de inicialização de pesos, as métricas RMSE obtidas neste experimento foram divididas em 3 grupos, chamados de "rfm", "ae" e "svd", onde cada grupo contém apenas os resultados dos métodos que

Figura 35 – Teste de Nemenyi utilizando a métrica RMSE dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras.



Fonte: Produção do próprio autor.

utilizam a respectiva técnica de inicialização de pesos. Os 3 grupos foram comparados utilizando o teste estatístico de Wilcoxon unicaudal à esquerda. Ou seja, se a hipótese nula for rejeitada apenas ao considerar qualquer par de grupos, a mediana das diferenças dos resultados é considerada menor que zero, e as métricas de RMSE do primeiro grupo são inferiores (menor erro quadrático). Ao comparar os três grupos par a par, a hipótese nula foi rejeitada ao comparar os grupos "svd" e "rfm" com o grupo "ae", com valores de estatística de 0,0006 e 0,0001, respectivamente. Isso significa que os grupos dos métodos que utilizam inicialização de pesos ortogonais ou pesos aleatórios foram considerados estatisticamente superiores ao grupo dos métodos com inicialização por *autoencoder*.

O mesmo teste foi repetido, porém dividindo os métodos em 2 grupos, com a finalidade de avaliar as variantes do algoritmo proposto. Tais grupos foram chamados de "OSFDSN-v" e "OSFDSN-v-i". Neste caso, a hipótese nula foi rejeitada ao comparar os dois grupos, com valor de estatística de 0,0367, respectivamente. Isso significa que, para os problemas de regressão analisados, OSFDSN-v foi considerado estatisticamente superior aos métodos OSFDSN-v-i.

O tempo gasto para treinar cada modelo também foi analisado, onde os valores obtidos podem ser encontrados no Apêndice D. Os *ranks* médios foram calculados e são apresentados na Tabela 17, onde os métodos *online* propostos obtiveram os melhores *ranks*. O teste de Friedman foi aplicado, e a hipótese nula de que os métodos são equivalentes foi rejeitada com valor-$p = 2{,}83 \times 10^{-13}$. Devido a ausência de uma etapa de treinamento no método kNN (ele só precisa armazenar as novas amostras), a análise estatística o desconsidera. Com isso, o teste de Nemenyi foi feito e a distância crítica de 4,5158 é

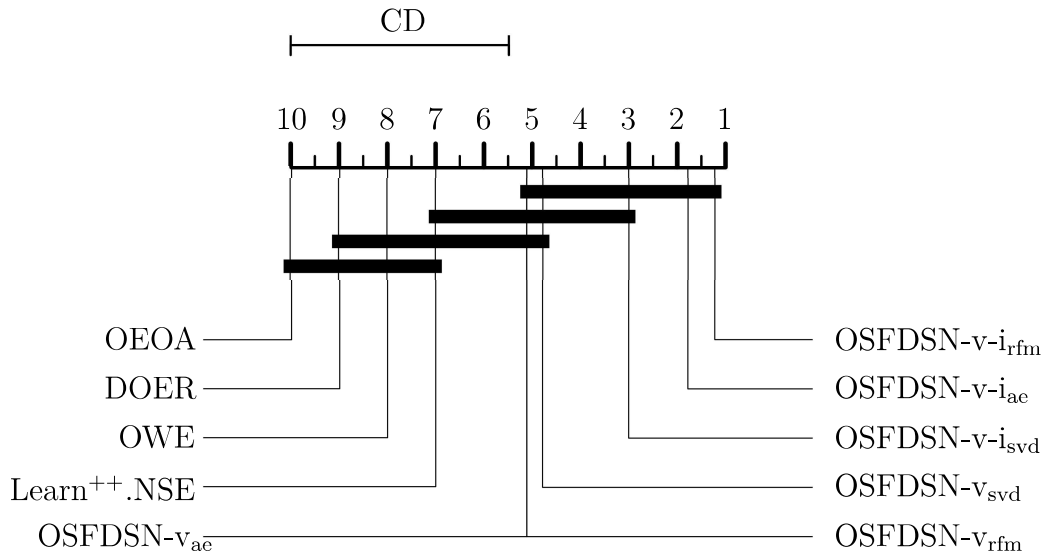
mostrada na Figura 36, que indica que os métodos baseados em OSFDSN-v-i propostos foram considerados estatisticamente equivalentes apenas aos métodos OSFDSN-v, sendo mais rápidos do que os demais, com exceção do método OSFDSN-v-i_{svd} que foi considerado similar ao método Learn^{++}.NSE.

Tabela 17 – *Ranks* médios ($\overline{\text{Rank}}$) e tempo de treinamento no formato "média (desvio padrão)" dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras. Os métodos estão ordenados pelo *rank* médio, de forma crescente.

Método	$\overline{\text{Rank}}$	Tempo de treinamento (s)
OSFDSN-v-i_{rfm}	1,2222	0,39 (0,20)
OSFDSN-v-i_{ae}	1,7778	0,42 (0,24)
OSFDSN-v-i_{svd}	3,0000	0,46 (0,25)
OSFDSN-v_{svd}	4,7778	4,64 (2,52)
OSFDSN-v_{rfm}	5,1111	4,70 (2,59)
OSFDSN-v_{ae}	5,1111	4,64 (2,53)
Learn^{++}.NSE	7,0000	15,09 (11,83)
OWE	8,0000	278,10 (220,16)
DOER	9,0000	2926,53 (1575,34)
OEOA	10,0000	2936,33 (1575,90)

Fonte: Produção do próprio autor.

Figura 36 – Teste de Nemenyi utilizando o tempo de treinamento dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras.



Fonte: Produção do próprio autor.

Analisando os valores de tempo de treinamento, os métodos baseados em OSFDSN-v-i atingiram um tempo de treinamento de cerca de 10% do tempo médio dos métodos baseados em OSFDSN-v, 2,6% do tempo gasto pelo Learn^{++}.NSE e 0,14% do tempo gasto pelo método OWE. Os demais métodos, levaram, em média, 10 vezes mais tempo que o

OWE, o que pode se tornar um problema se a aplicação lidar com dados em alta frequência (ou em simulações com bases de dados com muitos lotes).

Uma análise similar foi feita com o tamanho do modelo (número de bytes que ele necessita para ser armazenado). De forma similar ao ocorrido nos testes que envolvem tempo de treinamento, os métodos baseados em OSFDSN-v-i obtiveram os melhores *ranks*, como pode ser visto na Tabela 18[5]. O teste de Friedman foi aplicado, e a hipótese nula de que os métodos são equivalentes foi rejeitada com valor-$p = 6{,}29 \cdot 10^{-11}$. Com isso, o teste de Nemenyi foi feito e a distância crítica de 5,0328 é mostrada na Figura 37, que indica que o método OSFDSN-v-i_{rfm}, que obteve melhor *rank*, só foi considerado estatisticamente equivalente aos métodos OSFDSN-v-i e ao OWE, sendo considerado estatisticamente menor do que os demais.

Tabela 18 – *Ranks* médios ($\overline{\text{Rank}}$) e tamanho do modelo em kilobytes no formato "média (desvio padrão)" dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras. Os métodos estão ordenados pelo *rank* médio, de forma crescente.

Método	$\overline{\text{Rank}}$	Tamanho do modelo (kilobytes)
OSFDSN-v-i_{rfm}	1,2778	18,37 (8,28)
OSFDSN-v-i_{ae}	2,0556	20,54 (10,03)
OSFDSN-v-i_{svd}	3,6667	25,48 (11,02)
OWE	3,8889	27,68 (1,52)
OSFDSN-v_{rfm}	6,3333	74,96 (47,06)
Learn^{++}.NSE	6,5556	46,85 (10,28)
DOER	7,4444	64,22 (9,60)
OSFDSN-v_{ae}	7,8889	82,11 (48,77)
OEOA	8,0000	64,30 (9,71)
OSFDSN-v_{svd}	8,2222	84,58 (49,26)
kNN	10,6667	293,46 (216,96)

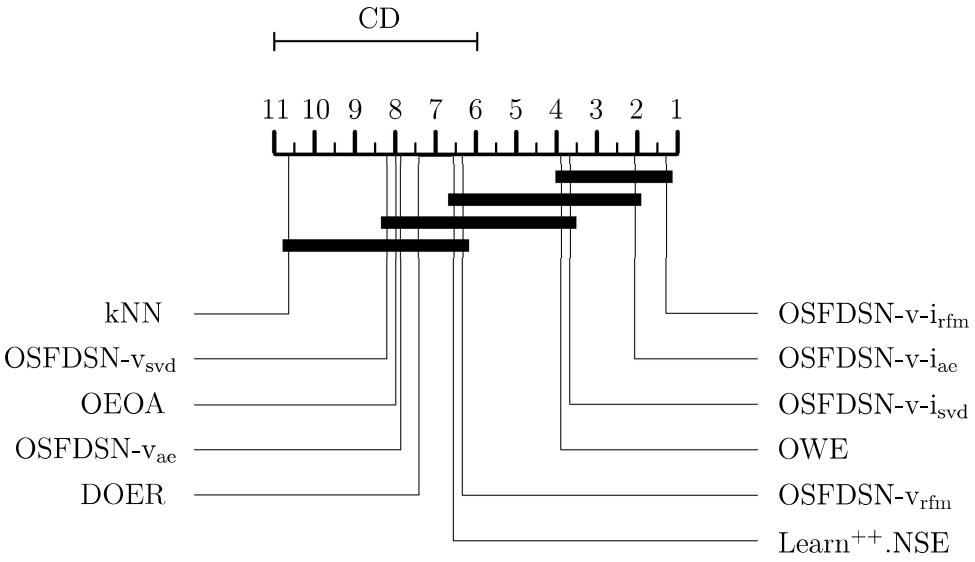
Fonte: Produção do próprio autor.

Vale destacar que os métodos baseados em OSFDSN-v e OSFDSN-v-i não possuem limitação de número de módulos e, portanto, poderiam ter crescido indefinidamente. O método OSFDSN-v-i obteve os melhores *ranks* nestes testes com bases de dados de até 8000 amostras, mas o seu desempenho pode ser inferior em testes com bases de dados com mais amostras. O método OSFDSN-v, mesmo lidando com "poucos" lotes, utilizou uma quantidade elevada de memória em comparação as demais técnicas, só precisando de menos memória que o método kNN.

Quanto aos valores numéricos, a quantidade de memória necessária para armazenar as redes OSFDSN-v-i treinada foi, em média, de aproximadamente 80%, 50%, 33% e 25%

[5] O número apresentado de bytes se refere a implementação realizada neste trabalho em MATLAB, sem muita preocupação com otimização de código e memória. As implementações dos métodos OWE, DOER e OEOA foram obtidas diretamente do site dos autores dos mesmos. Valores diferentes podem ser obtidos em outras implementações.

Figura 37 – Teste de Nemenyi utilizando o tamanho do modelo em bytes dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras.



Fonte: Produção do próprio autor.

do que foi utilizada para os métodos OWE, Learn^{++}.NSE, DOER/OEOA e OSFDSN-v, respectivamente. O método kNN, cujo "treinamento" é armazenar toda a base de dados, usou cerca de 15 vezes mais memória do que a utilizada pelos métodos OSFDSN-v-i.

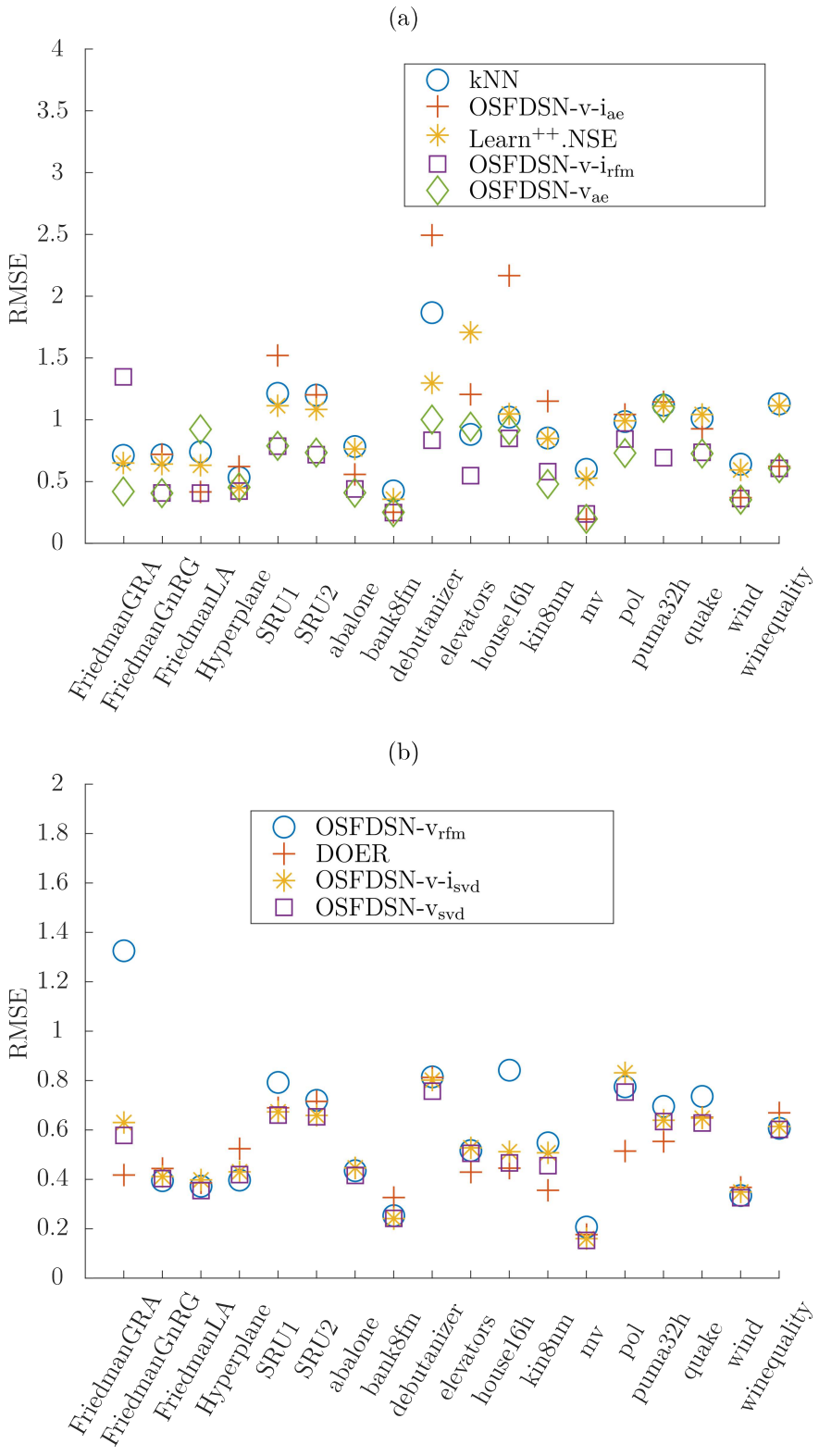
5.2.3.2 Considerando Todas as Bases Levantadas

Nesta seção são apresentados os resultados dos métodos citados na Seção 5.2.1.1 em todas as bases de dados da Tabela 15. Como o teste inclui bases de dados com mais de 40 mil amostras, as técnicas descritas na Seção 5.2.1.1, com exceção da DOER e OEOA, foram testadas. A função de ativação dos métodos baseados no OSFDSN-v e OSFDSN-v-i foram escolhidas entre as quatro funções de ativação citadas na Seção 5.2.1.2 nas suas respectivas etapas de validação cruzada.

Os valores de RMSE e hiperparâmetros encontrados nos experimentos realizados podem ser vistos no Apêndice E. Assim como citado na Seção 5.2.3.1, apenas o número médio de módulos dos métodos propostos é apresentado na Tabela 44 do Apêndice E, pois os mesmos podem ter conjuntos diferentes nas várias execuções, já hiperparâmetros utilizados nas demais técnicas foram os disponibilizados nas implementações fornecidas.

A Tabela 19 mostra os *ranks* médios obtidos por cada método referentes à métrica RMSE, assim como os seus valores médios. É possível notar que uma média menor não implica em um melhor *rank*, já que podem ter ocorridos *outliers* que deterioraram a métrica média, mas não o *rank*. Os gráficos das RMSE obtidas podem ser vistos na Figura 38, onde valores fora do intervalo apresentado foram ocultados para uma melhor visualização.

Figura 38 – RMSE obtidas nos experimentos com problemas de regressão *online* para todas as bases de dados da Tabela 15.



Fonte: Produção do próprio autor.

Tabela 19 – *Ranks* médios ($\overline{\text{Rank}}$) e métrica RMSE no formato "média (desvio padrão)" dos métodos testados nas bases de dados da Tabela 15. Os métodos estão ordenados pelo *rank* médio, de forma crescente.

Método	$\overline{\text{Rank}}$	RMSE	Mediana da RMSE
OSFDSN-v_{svd}	1,6389	0,50 (0,17)	0,4852
OSFDSN-v-i_{svd}	3,3889	0,53 (0,17)	0,5189
kNN	3,4444	0,49 (0,16)	0,4450
OSFDSN-v_{rfm}	3,7778	0,60 (0,26)	0,5765
OSFDSN-v_{ae}	4,6944	0,64 (0,27)	0,6691
OSFDSN-v-i_{rfm}	4,9167	0,62 (0,26)	0,5952
OWE	7,3889	0,89 (0,33)	0,9213
OSFDSN-v-i_{ae}	7,8056	1,30 (1,44)	0,9860
Learn^{++}.NSE	7,9444	0,91 (0,32)	0,8683

Fonte: Produção do próprio autor.

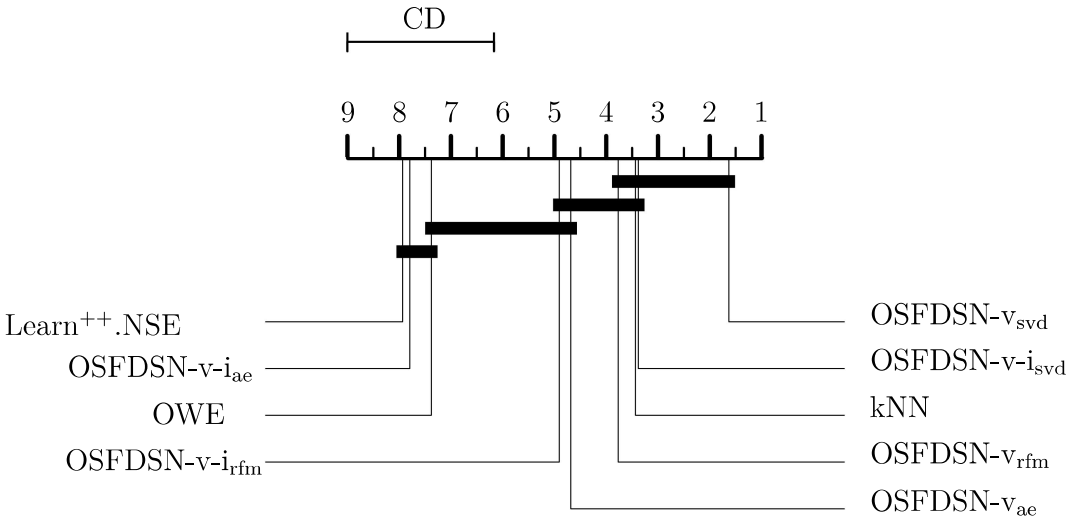
Os resultados apresentados na Tabela 19 mostram que, para problemas de regressão *online*, os métodos OSFDSN-v_{svd} e OSFDSN-v-i_{svd} atingiram os melhores *ranks*, seguido pelo método kNN e, em seguida, pelo OSFDSN-v_{rfm} e OSFDSN-v_{ae}. Assim como na Seção 5.2.3.1, os resultados considerando todas as bases de dados indicam que os métodos empilhados propostos neste trabalho podem ser adequados para utilização em regressão *online* em bases de dados maiores, principalmente quando utilizados em conjunto com método ortogonal de inicialização de pesos. Da mesma forma, a utilização do método por *autoencoder* piora o *rank* das técnicas, quando comparado com a inicialização aleatória. Quanto as técnicas utilizadas para comparação, o Learn^{++}.NSE, OWE e kNN obtiveram o nono, sétimo e terceiro melhor *rank*, respectivamente.

Vale notar que as médias das métricas não foram muito distantes uma das outras. O mesmo aconteceu com as medianas, que também podem ser conferidas na Tabela 19. A maior mediana, obtida pelo método OSFDSN-v-i_{ae}, foi cerca de 120% maior que a menor mediana, obtida pelo método kNN.

O teste estatístico de Friedman (DEMŠAR, 2006) foi aplicado nos resultados obtidos, considerando uma hipótese de que os métodos são equivalentes, usando um nível de significância de 5%. O teste obteve um valor-$p = 3{,}18 \cdot 10^{-17}$, rejeitando a hipótese nula. Assim, o teste de Nemenyi foi utilizado, cujo resultado pode ser visto na Figura 39. A distância crítica, neste caso, foi de 2,8317, ou seja, métodos cuja diferença de *ranks* médios é maior que este valor não são equivalentes. O teste indica que o método OSFDSN-v_{svd}, que obteve o melhor *rank*, só é equivalente aos métodos OSFDSN-v-i_{svd}, kNN e OSFDSN-v_{rfm}, sendo estatisticamente melhor que os demais, incluindo os métodos OWE e Learn^{++}.NSE.

Com a finalidade de comparar o desempenho entre métodos de inicialização de pesos, as métricas RMSE obtidas neste experimento foram divididas em 3 grupos, chamados de "rfm", "ae" e "svd", onde cada grupo contém apenas os resultados dos métodos que utilizam a respectiva técnica de inicialização de pesos. Os 3 grupos foram comparados

Figura 39 – Teste de Nemenyi utilizando a métrica RMSE dos métodos testados em todas as bases de dados da Tabela 15.



Fonte: Produção do próprio autor.

utilizando o teste estatístico de Wilcoxon unicaudal à esquerda. Ou seja, se a hipótese nula for rejeitada ao considerar qualquer par de grupos, a mediana das diferenças dos resultados é considerada menor que zero, e as métricas de RMSE do primeiro grupo são inferiores (menor erro quadrático). Ao comparar os três grupos par a par, a hipótese nula foi rejeitada ao comparar os grupos "svd" e "rfm" com o grupo "ae", com valores de estatística de 8×10^{-7} e 0,0001, respectivamente, e ao comparar os métodos "svd" e "rfm", com valor de estatística 0,0001. Isso significa que os grupos dos métodos que utilizam inicialização de pesos ortogonais ou pesos aleatórios foram considerados estatisticamente superiores ao grupo dos métodos com inicialização por *autoencoder*, e a inicialização por pesos ortogonais também foi considerada superior à inicialização aleatória.

O mesmo teste foi repetido, porém dividindo os métodos em 2 grupos, com a finalidade de avaliar as variantes do algoritmo proposto. Tais grupos foram chamados de "OSFDSN-v" e "OSFDSN-v-i". Neste caso, a hipótese nula foi rejeitada ao comparar os dois grupos, com valor de estatística de 5×10^{-9}. Isso significa que, para os problemas de regressão analisados, OSFDSN-v foi considerado estatisticamente superior aos métodos OSFDSN-v-i.

O tempo gasto para treinar cada modelo também foi analisado, onde os valores obtidos podem ser encontrados no Apêndice E. Os *ranks* médios foram calculados e são apresentados na Tabela 20, onde os métodos OSFDSN-v-i obtiveram os melhores *ranks*, seguido do método Learn^{++}.NSE, os métodos OSFDSN-v e o método OWE. O teste de Friedman foi aplicado, e a hipótese nula de que os métodos são equivalentes foi rejeitada com valor-$p = 6{,}59 \times 10^{-21}$. Devido a ausência de uma etapa de treinamento

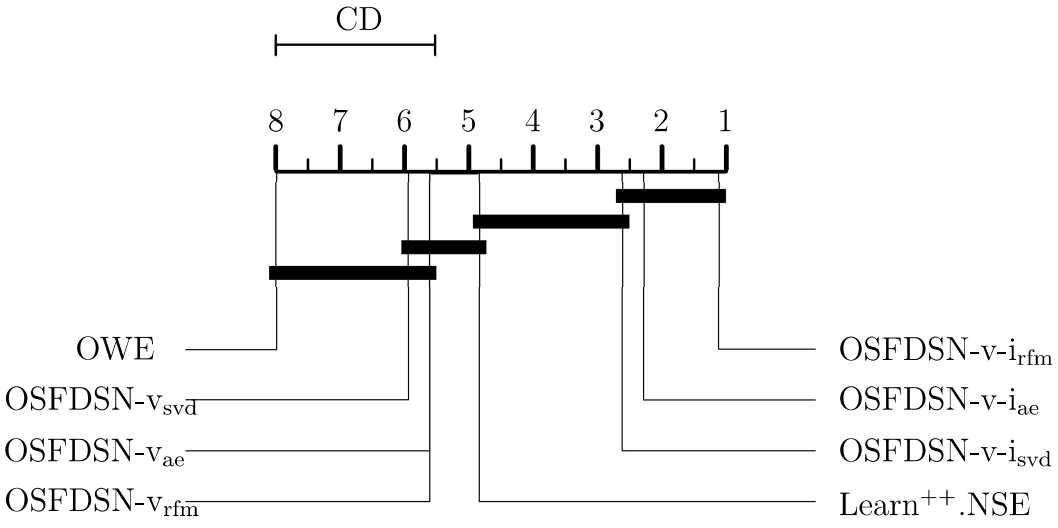
no método kNN (ele só precisa armazenar as novas amostras), a análise estatística o desconsidera. Com isso, o teste de Nemenyi foi feito e a distância crítica de 2,4748 é mostrada na Figura 40, que indica que os métodos baseados em OSFDSN-v-i propostos foram considerados estatisticamente mais rápidos do que os demais, com exceção do método OSFDSN-v-i_{svd}, que foi considerado equivalente também ao método Learn^{++}.NSE. O método OWE, que obteve os piores *ranks*, foi considerado equivalente aos métodos OSFDSN-v.

Tabela 20 – *Ranks* médios ($\overline{\text{Rank}}$) e tempo de treinamento no formato "média (desvio padrão)" dos métodos testados em todas as bases de dados da Tabela 15. Os métodos estão ordenados pelo *rank* médio, de forma crescente.

Método	$\overline{\text{Rank}}$	Tempo de treinamento (s)
OSFDSN-v-i_{rfm}	1.1111	2.06 (4.26)
OSFDSN-v-i_{ae}	2.2778	2.62 (4.81)
OSFDSN-v-i_{svd}	2.6111	3.04 (7.61)
Learn^{++}.NSE	4.8333	66.87 (87.41)
OSFDSN-v_{rfm}	5.6111	56.77 (62.44)
OSFDSN-v_{ae}	5.6111	56.94 (62.68)
OSFDSN-v_{svd}	5.9444	57.34 (63.76)
OWE	8.0000	1249.83 (1653.44)

Fonte: Produção do próprio autor.

Figura 40 – Teste de Nemenyi utilizando o tempo de treinamento dos métodos testados em todas as bases de dados da Tabela 15.



Fonte: Produção do próprio autor.

Analisando os valores de tempo de treinamento, os métodos baseados em OSFDSN-v-i atingiram um tempo de treinamento de cerca de 4% dos tempo médio dos métodos baseados em OSFDSN-v, 3% do tempo gasto pelo Learn^{++}.NSE e 0,16% do tempo gasto pelo método OWE. Comparando com os resultados obtidos na Seção 5.2.3.1, o método

OSFDSN-v obteve uma piora no tempo gasto, devido à quantidade de módulos adicionados ao longo da chegada das amostras. Quanto mais módulos os modelos OSFDSN-v e OSFDSN-v-i possuírem, mais processamento eles exigem.

Uma análise similar foi feita com o tamanho do modelo (número de bytes que ele necessita para ser armazenado). De forma similar ao ocorrido nos testes que envolvem tempo de treinamento, os métodos baseados em OSFDSN-v-i obtiveram os melhores *ranks*, como pode ser visto na Tabela 21[6]. O teste de Friedman foi aplicado, e a hipótese nula de que os métodos são equivalentes foi rejeitada com valor-$p = 4{,}48 \cdot 10^{-23}$. Com isso, o teste de Nemenyi foi feito e a distância crítica de 2,8317 é mostrada na Figura 41, que indica que o método OSFDSN-v-i_{rfm}, que obteve melhor *rank*, só foi considerado estatisticamente equivalente aos métodos OSFDSN-v-i e ao OWE, sendo considerado estatisticamente menor do que os demais.

Tabela 21 – *Ranks* médios ($\overline{\text{Rank}}$) e tamanho do modelo em kilobytes no formato "média (desvio padrão)" dos métodos testados em todas as bases de dados da Tabela 15. Os métodos estão ordenados pelo *rank* médio, de forma crescente.

Método	$\overline{\text{Rank}}$	Tamanho do modelo (kilobytes)
OSFDSN-v-i_{rfm}	1.4444	30.26 (45.75)
OSFDSN-v-i_{ae}	2.7778	47.66 (57.97)
OSFDSN-v-i_{svd}	3.0556	47.98 (87.89)
OWE	3.1111	29.37 (4.78)
Learn^{++}.NSE	4.9444	75.38 (41.00)
OSFDSN-v_{rfm}	6.2222	283.35 (239.82)
OSFDSN-v_{ae}	6.9444	296.68 (254.35)
OSFDSN-v_{svd}	7.8333	308.85 (256.05)
kNN	8.6667	1259.33 (1575.86)

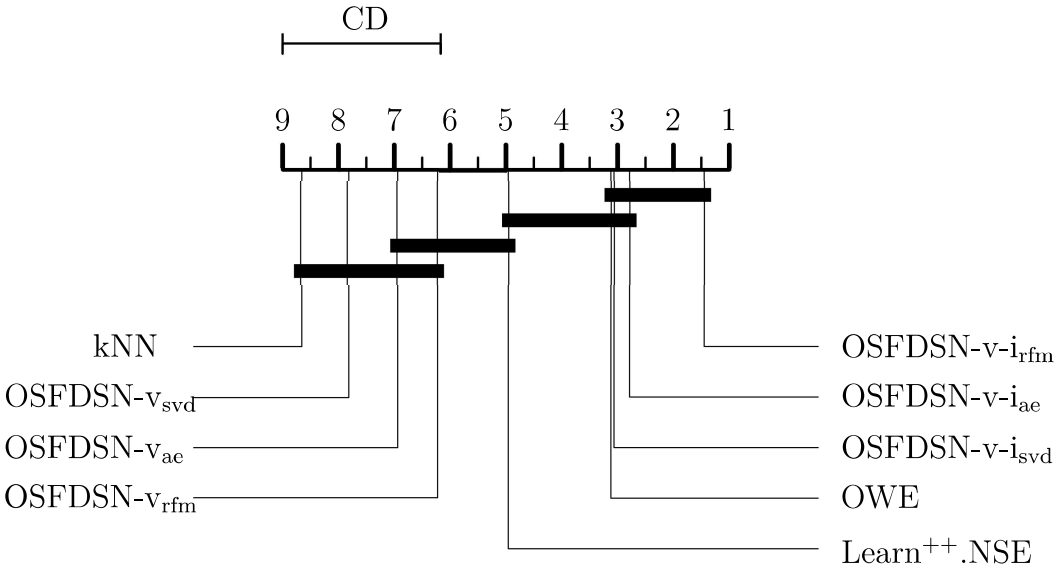
Fonte: Produção do próprio autor.

Assim como destacado na Seção 5.2.3.1, os métodos OSFDSN-v e OSFDSN-v-i não possuem limitação de número de módulos e, portanto, poderiam ter crescido indefinidamente. Embora, na média, os métodos OSFDSN-v-i precisam de mais memória, os *ranks* médios dos mesmos foram melhores do que do método OWE. Já os métodos OSFDSN-v lidaram com muitos lotes nesse teste e, consequentemente, utilizaram uma quantidade elevada de memória, sendo maior que o dos métodos OWE e Learn^{++}.NSE. Vale lembrar que esses valores dependem do número de lotes nos métodos baseados no OSFDSN, cuja tendência é crescer a medida em que mais lotes sejam processados.

Quanto aos valores numéricos, a quantidade de memória necessária para armazenar as redes OWE e OSFDSN-v-i_{rfm} foram praticamente iguais, sendo a segunda 3% superior à primeira, enquanto os métodos OSFDSN-v-i_{svd} e OSFDSN-v-i_{ae} utilizam cerca de 160%

[6] O número apresentado de bytes se refere a implementação realizada neste trabalho em MATLAB, sem muita preocupação com otimização de código e memória. A implementação do método OWE foi obtida diretamente do site dos autores. Valores diferentes podem ser obtidos em outras implementações.

Figura 41 – Teste de Nemenyi utilizando o tamanho do modelo em bytes dos métodos testados nas bases de dados da Tabela 15 de até 8000 amostras.



Fonte: Produção do próprio autor.

da memória utilizada pelo OWE. Já o método Learn^{++}.NSE e os métodos baseados em OSFDSN-v utilizam cerca de 250% e 1000% da memória utilizada pelo OWE, respectivamente. De qualquer forma, todos os métodos utilizaram menos memória, em média, do que o kNN, que apenas armazena as amostras recebidas.

5.3 Considerações Finais

Com relação à métrica RMSE, os métodos propostos neste trabalho obtiveram valores similares aos métodos comparados, como DOER, OEOA, kNN, OWE e Learn^{++}.NSE para as bases de dados de até 8000 amostras. Levando em consideração todas as bases de dados, mas excluindo os métodos DOER e OEOA (devido ao seu elevado tempo de processamento), os métodos propostos, quando utilizados em conjunto com a inicialização ortogonal de pesos, obtiveram melhores métricas e melhores *ranks* que as demais técnicas. O teste estatístico de Wilcoxon foi utilizado, e seus resultados indicam que a técnica de inicialização de pesos ortogonais é superior às outras técnicas comparadas quando é considerado um problema de regressão, quando todas as bases de dados foram analisadas.

Os experimentos realizados mostram que os métodos DOER e OEOA (considerando a implementação disponibilizada pelos autores) levam, em média, 1 segundo para processar cada amostra das bases de dados consideradas. Assim, os métodos DOER e OEOA não foram inclusos nos testes com as bases de dados com muitas amostras, devido à grande quantidade de tempo necessária para executá-los múltiplas vezes. Isto se torna uma

limitação de tais técnicas, que não são capazes de funcionar em aplicações onde são geradas mais de uma amostra por segundo. Já o método OWE processa, em média, 10 amostras por segundo, enquanto as técnicas propostas neste trabalho são capazes de processar um número muito maior de amostras por segundo, principalmente se a variante OSFDSN-v-i for considerada. Vale lembrar que o tempo de processamento destas técnicas tende a ser proporcional ao número de lotes que ela recebe, devido ao número ilimitado de módulos que estas técnicas podem possuir.

Quanto à quantidade de memória gasta pelas técnicas, os métodos propostos possuem a desvantagem de serem ilimitados com relação ao número de módulos e, consequentemente, irão consumir mais memória ao longo do tempo com a chegada dos lotes. Mesmo com essa desvantagem, a utilização de memória destas técnicas foi inferior a maioria das técnicas comparadas, principalmente quando considerada a variante OSFDSN-v-i. Das técnicas comparadas, o método OWE utilizou menos memória, em média, mas obteve tempos de treinamento superiores e métricas inferiores.

No geral, o método OSFDSN-v-i com inicialização ortogonal de pesos foi a técnica *online* proposta que obteve um melhor compromisso entre tempo de treinamento, memória utilizada e métrica de erro. Tanto nos testes com bases até 8 mil amostras quanto nos experimentos que incluíram todas as bases de dados, o *rank* desta técnica quanto à métrica RMSE só foi superado pela técnica OSFDSN-v, que tem tempo de treinamento superior e necessita de mais memória para ser armazenada. Vale destacar que, caso o treinamento do OSFDSN seja feito em sistemas com restrições de memória, é possível adaptar o Algoritmo 4 para copiar apenas o último módulo, ao invés dos modelos completos, embora isso dependa de um maior cuidado na implementação do mesmo.

6 Conclusão e Trabalhos Futuros

Este trabalho considera a hipótese de que a combinação de modelos de aprendizado de máquina que utilizam o princípio de empilhamento com algoritmos rápidos de treinamento, como os baseados em ELM, podem resultar em um modelo cujo treinamento é relativamente rápido e que necessita de pouca memória para ser armazenado enquanto mantém a qualidade dos resultados, tanto para tarefas consideradas *offline* quanto *online*.

Duas principais abordagens foram propostas neste trabalho para avaliar tal hipótese: os métodos FDSN e OSFDSN. Ambas são baseadas no método DSN, que tem objetivo de empilhar modelos pequenos para atingir desempenhos comparáveis a modelos maiores. Porém, DSN possui desvantagens relacionadas ao seu uso de memória e tempo de treinamento. As abordagens FDSN e OSFDSN utilizam algoritmos rápidos de treinamento baseados em ELM, juntamente com uma abordagem de reaproveitamento de pesos. O emprego destas abordagens resultou num procedimento recursivo de realizar o treinamento dos módulos, o que tornou a construção da arquitetura DSN computacionalmente menos custosa, tanto em termos de tempo quanto de memória.

Entretanto, a reutilização de pesos pode prejudicar o desempenho da rede, caso estes não sejam adequados para o problema tratado. Para evitar tal problema, foi proposta a utilização de métodos mais complexos de inicialização de pesos, baseados em *autoencoders* e pesos ortogonais. Tais métodos foram propostos na literatura no contexto de ELM e obtêm um novo conjunto de pesos de forma rápida, a partir de pesos aleatórios inicialmente obtidos, que usualmente resulta em melhor desempenho. Também foi proposta uma variante do FDSN, chamada de FKDSN, que utiliza o conceito de *kernels* e não depende de pesos puramente aleatórios.

Os métodos FDSN e FKDSN foram apresentados no Capítulo 3 e foram testados em diversas bases de dados de problemas supervisionados de aprendizado, com o objetivo de avaliar a hipótese levantada de que a combinação de um algoritmo rápido de treinamento de redes neurais com uma arquitetura empilhada pode obter bons resultados com uso de memória e tempo de treinamento reduzidos para tarefas de aprendizado de máquina *offline*. Nas tarefas de classificação, 15 bases de dados multi-classe e a métrica de avaliação acurácia foram utilizadas. Para regressão, 41 bases de dados e a métrica aRRMSE foram utilizadas, contemplando conjuntos com uma e múltiplas saídas. Diversos experimentos foram realizados para avaliar o comportamento de tais métodos. O primeiro grupo de experimentos envolveu bases de dados de tamanhos distintos, onde foi observado que a acurácia dos modelos sobre os conjuntos de teste melhorava à medida que o número de amostras de treinamento por classe aumentava, convergindo para o valor teórico esperado.

O segundo grupo de experimentos avaliou o compromisso entre número de módulos e número de neurônios por módulo, onde os resultados obtidos mostraram que há uma melhora nas métricas ao aumentar as duas grandezas, sempre atingindo uma região de saturação após determinado valor dos hiperparâmetros. Um dos testes realizados mostrou que o DSN e o FDSN com 30 módulos de 100 neurônios foram capazes de obter métricas melhores do que uma rede única que contém 3000 neurônios.

O terceiro grupo de experimentos do Capítulo 3 avaliou o desempenho dos métodos em problemas de aprendizado de máquina, considerando métricas de erro, tempo de treinamento e uso de memória, analisando os resultados através dos testes estatísticos de Friedman e de Nemenyi. Considerando o problema de classificação e a métrica acurácia, os testes estatísticos mostraram que os métodos propostos - FKDSN, FDSN e suas variantes - se mostraram equivalentes aos demais métodos, onde o FDSN apresentou melhores resultados quando combinado com um método complexo de inicialização de pesos (por exemplo, com pesos ortogonais). Além disto, os métodos propostos obtiveram o menor tempo de treinamento e utilizaram a menor quantidade de memória dentre as técnicas comparadas. Assim, o FKDSN e o FDSN combinado com pesos ortogonais podem substituir redes SLFN maiores em ambientes restritos, pois possuem desempenhos estatisticamente similares e necessitam, em média, cerca de 20% da memória necessária para armazenar a rede SLFN em questão. Vale destacar que testes estatísticos indicaram que o método de inicialização por *autoencoder* foi estatisticamente superior aos demais métodos testados para problemas de classificação.

Já para problemas de regressão, os desempenhos dos métodos empilhados foram melhores. Os resultados de aRRMSE do FDSN foram tão bons quanto dos demais métodos testados, e considerado estatisticamente equivalente a eles, independente do método de inicialização de pesos. De forma semelhante ao observado nos problemas de classificação, foi o método com treinamento mais rápido, que consome menos memória e a utilização de pesos ortogonais ou *kernels* trouxe uma melhora para o desempenho das técnicas propostas. Neste caso, testes estatísticos indicaram que o método de inicialização com pesos ortogonais foi estatisticamente superior aos demais métodos testados.

Com relação às limitações das técnicas FDSN e FKDSN, os experimentos indicaram que a combinação das mesmas com um dos métodos de inicialização de pesos tende a melhorar os resultados de problemas de classificação e de regressão e, portanto, é recomendado que elas sejam combinadas com tais técnicas. Além disso, foi observado que os valores das métricas das técnicas *offline* propostas tendem a saturar após determinada combinação de número de módulos e neurônios por módulo, o que significa que um aumento desnecessário nessas grandezas pode demandar um uso desnecessário de recursos computacionais, mesmo que a adição de um novo módulo nas técnicas propostas caracterizem um baixo incremento na necessidade de recursos computacionais. Com relação aos resultados, a técnica FDSN

obteve resultados similares à técnica DSN, mas foi considerada estatisticamente inferior quando ambas são treinadas nas mesmas condições. Contudo, a técnica proposta requer quantidades consideravelmente inferiores de memória e tempo para treinamento do que as requeridas pela técnica DSN.

No Capítulo 4 foi feito um estudo de caso das técnicas *offline* propostas (FDSN e FKDSN) em bases de dados públicas de problemas reais, no caso a predição de geração de energias renováveis. Os métodos propostos foram utilizados seguindo uma metodologia proposta em um trabalho da literatura, e atingiram boas métricas de erro, mantendo um bom compromisso entre qualidade de predição, uso de memória e tempo de treinamento. Assim, os quatro primeiros objetivos específicos desta tese, que tratam de propor, treinar e avaliar um método de aprendizado de máquina empilhado com treinamento rápido, baixo uso de memória e com qualidade comparável a outros métodos da literatura, foram atingidos.

O método OSFDSN foi apresentado no Capítulo 5 e foi testado em diversas bases de dados de problemas supervisionados de aprendizado de máquina *online*, que lidam com dados de forma sequencial, com o objetivo de avaliar a hipótese levantada de que é possível adaptar a técnica FDSN para lidar com tarefas de aprendizado de máquina *online*, obtendo bons resultados com uso de memória e tempo de treinamento reduzidos. Ele é uma versão modificada do FDSN: cada módulo é treinado utilizando um algoritmo *online* e baseado em ELM, novos módulos só podem ser adicionados quando um novo lote de dados chega para o algoritmo e todos os módulos possuem uma parcela de contribuição na saída da técnica. Para lidar com ambientes não estacionários, em que as distribuições estatísticas dos dados se alteram ao longo do tempo (efeito chamado de *concept drift*), a contribuição de cada módulo na saída da técnica é atualizada para cada lote, de forma que os módulos com menor erro tenham maior peso na predição final do OSFDSN.

Dois grupos de experimentos foram realizados para avaliar o comportamento do OSFDSN. O primeiro grupo de experimentos avalia a importância do tamanho do lote no desempenho da técnica, onde os resultados mostraram que esta grandeza tem relação direta com as características dinâmicas das bases de dados, levando em consideração que lotes pequenos podem trazer pouca informação útil (e aumentar o tempo de processamento da técnica), e lotes grandes podem impactar na capacidade do método de se adaptar a mudanças bruscas na distribuição dos dados. Já o segundo grupo de experimentos compara o método proposto com abordagens similares propostas na literatura, que também lidam com regressão *online*. Para isso, foram utilizadas bases de dados sintéticas e de problemas reais que já foram aplicadas na literatura para testar métodos de regressão *online*. As duas variantes da técnica OSFDSN obtiveram os melhores *ranks* dos experimentos quando utilizadas em conjunto com a inicialização ortogonal de pesos e comparadas a outras técnicas de regressão *online* propostas na literatura. Com respeito ao tempo de

treinamento, os métodos propostos utilizaram um tempo consideravelmente inferior aos das demais técnicas, principalmente se tratando da variante OSFDSN-v-i. Novamente, testes estatísticos indicaram que o método de inicialização com pesos ortogonais é estatisticamente superior aos demais métodos testados para problemas de regressão *online*.

A técnica OSFDSN possui a limitação de que não é possível descartar nenhum módulo intermediário que poderia conter informações ultrapassadas. Embora isso possa ser importante na presença de conceitos recorrentes, isso traz desvantagens para o método no caso da aplicação de um limite de número de módulos. Portanto, os métodos OSFDSN não possuem tal limite e podem requerer uma grande quantidade de memória após receberem uma grande quantidade de dados. Mesmo com essa desvantagem, resultados satisfatórios foram obtidos nos testes realizados, principalmente na variante OSFDSN-v-i.

Nos Capítulos 4 e 5, foram feitas análises com relação ao *concept drift* de algumas bases de dados. Esta abordagem envolveu analisar a projeção de lotes distintos na componente principal da base de dados inteira. Porém, esta abordagem não se mostrou muito efetiva para avaliar a ocorrência de tal efeito nas diferentes bases de dados utilizadas.

Assim, todos os objetivos específicos desta tese foram atingidos. A hipótese levantada neste trabalho considera ser possível combinar uma arquitetura do tipo DSN e algoritmos baseados em ELM, resultando em uma rede com desempenho estatisticamente equivalente a outros métodos e tempo de treinamento e uso de memória reduzidos. Esta hipótese foi validada através dos métodos propostos neste trabalho, que foram avaliados por diversos experimentos, tanto para problemas de aprendizado de máquina *offline* quanto *online*.

Trabalhos futuros poderão ser conduzidos, no contexto de aprendizado de máquina *offline*, para avaliar o comportamento das técnicas propostas utilizando métricas diferentes da acurácia, para classificação, bem como a utilização de *kernels* diferentes do gaussiano, testado neste trabalho. Outra possibilidade envolve avaliar uma abordagem de ponderação das saídas intermediárias para encontrar a saída do modelo de modo a otimizar uma determinada métrica, também no contexto *offline*. As técnicas também podem ser avaliadas em outros contextos de divisão de base de dados, utilizando diferentes formas de validação cruzada. Já para problemas *online*, trabalhos futuros envolvem avaliar a utilização de *kernels* na técnica OSFDSN, que também pode ser adaptada para lidar com problemas de classificação *online*. Outras abordagens de atualização dos pesos dinâmicos que são utilizados para achar as contribuições dos módulos na saída geral do método OSFDSN também podem ser avaliadas, bem como outras formas de atualizar a arquitetura da OSFDSN. Também é possível avaliar o método OSFDSN em bases de dados específicas para *concept drift*, bem como testar as técnicas propostas utilizando diferentes metodologias de experimento. Propõe-se, também, a utilização de outras abordagens de redução de dimensionalidade para avaliar *concept drift* das bases de dados, com a finalidade de relacionar os tipos desse efeito com o desempenho das técnicas propostas neste trabalho.

Referências

AKUSOK, A. et al. High-performance extreme learning machines: a complete toolbox for big data applications. *IEEE Access*, IEEE, v. 3, p. 1011–1025, 2015. Citado na página 76.

ALCALÁ-FDEZ, J. et al. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic & Soft Computing*, Citeseer, v. 17, 2011. Citado 2 vezes nas páginas 80 e 125.

BOYD, S. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends® in Machine Learning*, v. 3, n. 1, p. 1–122, 2010. ISSN 1935-8237. Disponível em: ⟨http://arxiv.org/abs/1408.2927http://www.nowpublishers.com/article/Details/MAL-016⟩. Citado 3 vezes nas páginas 43, 44 e 48.

CHANG, C.-C.; LIN, C.-J. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, Acm, v. 2, n. 3, p. 27, 2011. Citado na página 80.

CHEN, C. et al. The direct extension of admm for multi-block convex minimization problems is not necessarily convergent. *Mathematical Programming*, Springer, v. 155, n. 1-2, p. 57–79, 2016. Citado na página 47.

CIARELLI, P. M.; OLIVEIRA, E.; SALLES, E. O. An incremental neural network with a reduced architecture. *Neural networks*, Elsevier, v. 35, p. 70–81, 2012. Citado na página 33.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, Springer, v. 2, n. 4, p. 303–314, 1989. Citado na página 30.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, v. 7, p. 1–30, 2006. ISSN 15337928. Disponível em: ⟨http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.3142{&}rep=rep1{&}ty⟩. Citado 8 vezes nas páginas 82, 83, 84, 94, 102, 120, 137 e 143.

DENG, L. A tutorial survey of architectures, algorithms, and applications for deep learning. *Sip*, v. 3, n. 2, p. 1–29, 2014. Disponível em: ⟨http://creativecommons.org/licenses/by/3.0/⟩. Citado 2 vezes nas páginas 31 e 32.

DENG, L. Deep Learning: Methods and Applications. *Foundations and Trends® in Signal Processing*, v. 7, n. 3-4, p. 197–387, aug 2014. ISSN 1932-8346. Disponível em: ⟨http://www.bmj.com/cgi/doi/10.1136/bmj.319.7209.0ahttp://nowpublishers.com/articles/foundations-and-trends-in-signal-processing/SIG-039⟩. Citado 3 vezes nas páginas 50, 51 e 69.

DENG, L. *Deep learning: From speech recognition to language and multimodal processing*. 2016. Citado na página 32.

DENG, L.; YU, D. Deep convex net: A scalable architecture for speech pattern classification. *Proceedings of the Annual Conference of the International Speech*

Communication Association, INTERSPEECH, n. August, p. 2285–2288, 2011. ISSN 19909772. Citado 6 vezes nas páginas 32, 54, 55, 69, 70 e 71.

DENG, W.; ZHENG, Q.; CHEN, L. Regularized Extreme Learning Machine. In: *2009 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2009. p. 389–395. ISBN 978-1-4244-2765-9. Disponível em: ⟨http://ieeexplore.ieee.org/document/4938676/⟩. Citado 3 vezes nas páginas 41, 42 e 86.

DENG, W. Y.; ONG, Y. S.; ZHENG, Q. H. A Fast Reduced Kernel Extreme Learning Machine. *Neural Networks*, Elsevier Ltd, v. 76, p. 29–38, 2016. ISSN 18792782. Disponível em: ⟨http://dx.doi.org/10.1016/j.neunet.2015.10.006⟩. Citado 2 vezes nas páginas 50 e 73.

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Disponível em: ⟨http://archive.ics.uci.edu/ml⟩. Citado 3 vezes nas páginas 80, 89 e 125.

DUARTE, J. a.; GAMA, J. a.; BIFET, A. Adaptive model rules from high-speed data streams. *ACM Trans. Knowl. Discov. Data*, Association for Computing Machinery, New York, NY, USA, v. 10, n. 3, jan. 2016. ISSN 1556-4681. Disponível em: ⟨https://doi.org/10.1145/2829955⟩. Citado na página 126.

ELGAMAL, T. et al. spca: Scalable principal component analysis for big data on distributed platforms. In: *Proceedings of the International Conference on Management of Data*. New York, USA: [s.n.], 2015. p. 79–91. ISBN 978-1-4503-2758-9. Citado na página 76.

ELWELL, R.; POLIKAR, R. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, Institute of Electrical and Electronics Engineers (IEEE), v. 22, n. 10, p. 1517–1531, out. 2011. Disponível em: ⟨https://doi.org/10.1109/tnn.2011.2160459⟩. Citado 3 vezes nas páginas 57, 61 e 62.

FERNÁNDEZ-DELGADO, M. et al. Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research*, JMLR. org, v. 15, n. 1, p. 3133–3181, 2014. Citado na página 82.

FRIEDMAN, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, Taylor & Francis, v. 32, n. 200, p. 675–701, 1937. Citado na página 82.

FUKUNAGA, K. *Introduction to Statistical Pattern Recognition (2nd Ed.)*. USA: Academic Press Professional, Inc., 1990. ISBN 0122698517. Citado 3 vezes nas páginas 15, 87 e 88.

HALL, M. et al. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, Association for Computing Machinery, New York, NY, USA, v. 11, n. 1, p. 10–18, nov. 2009. ISSN 1931-0145. Disponível em: ⟨https://doi.org/10.1145/1656274.1656278⟩. Citado na página 125.

HAYKIN, S. S. *Neural networks and learning machines*. Third. Upper Saddle River, NJ: Pearson Education, 2009. Citado na página 29.

HOLLANDER, M.; WOLFE, D. *Nonparametric Statistical Methods*. Wiley, 1999. (Wiley Series in Probability and Statistics). ISBN 9780471190455. Disponível em: ⟨https://books.google.com.br/books?id=RJAQAQAAIAAJ⟩. Citado na página 82.

HUANG, G. et al. Trends in extreme learning machines: A review. *Neural Networks*, v. 61, p. 32–48, jan 2015. ISSN 08936080. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0893608014002214https://linkinghub.elsevier.com/retrieve/pii/S0893608014002214⟩. Citado na página 30.

HUANG, G.-B. et al. Extreme Learning Machine for Regression and Multiclass Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 42, n. 2, p. 513–529, apr 2012. ISSN 1083-4419. Disponível em: ⟨http://ieeexplore.ieee.org/document/6035797/⟩. Citado 5 vezes nas páginas 30, 42, 48, 65 e 86.

HUANG, G.-B.; ZHU, Q.-Y.; SIEW, C.-K. Extreme learning machine: a new learning scheme of feedforward neural networks. In: *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*. IEEE, 2004. v. 2, p. 985–990. ISBN 0-7803-8359-1. ISSN 1098-7576. Disponível em: ⟨http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1380068http://ieeexplore.ieee.org/document/1380068/⟩. Citado 3 vezes nas páginas 30, 39 e 40.

HUANG, G.-B.; ZHU, Q.-Y.; SIEW, C.-K. Extreme learning machine: Theory and applications. *Neurocomputing*, v. 70, n. 1-3, p. 489–501, dec 2006. ISSN 09252312. Disponível em: ⟨http://linkinghub.elsevier.com/retrieve/pii/S0925231206000385⟩. Citado 3 vezes nas páginas 30, 39 e 42.

HUYNH, H. T.; WON, Y. Regularized online sequential learning algorithm for single-hidden layer feedforward neural networks. *Pattern Recognition Letters*, Elsevier B.V., v. 32, n. 14, p. 1930–1935, oct 2011. ISSN 01678655. Disponível em: ⟨http://dx.doi.org/10.1016/j.patrec.2011.07.016http://linkinghub.elsevier.com/retrieve/pii/S0167865511002297⟩. Citado na página 129.

IKONOMOVSKA, E.; GAMA, J.; DZEROSKI, S. Online tree-based ensembles and option trees for regression on evolving data streams. *Neurocomputing*, v. 150, 11 2014. Citado na página 126.

INABA, F. K. *Generalização do extreme learning machine regularizado para problemas de múltiplas saídas*. 113 p. Tese (Doutorado em Engenharia Elétrica) — Universidade Federal do Espírito Santo, 2018. Citado 5 vezes nas páginas 31, 41, 46, 81 e 83.

INABA, F. K. et al. DGR-ELM - Distributed Generalized Regularized ELM for classification. *Neurocomputing*, Elsevier B.V., v. 275, p. 1522–1530, 2018. ISSN 18728286. Disponível em: ⟨https://doi.org/10.1016/j.neucom.2017.09.090⟩. Citado 7 vezes nas páginas 31, 40, 42, 43, 44, 45 e 86.

JONG, S. de. Simpls: An alternative approach to partial least squares regression. *Chemometrics and Intelligent Laboratory Systems*, v. 18, n. 3, p. 251–263, 1993. ISSN 0169-7439. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/016974399385002X⟩. Citado na página 128.

KASUN, L. L. C. et al. Representational learning with extreme learning machine for big data. *IEEE Intelligent Systems*, v. 28, n. 6, p. 30–59, nov 2013. Disponível em: ⟨http://www.ntu.edu.sg/home/egbhuang/pdf/ELM{_}Auto{_}Encoder.pdfhttp://ieeexplore.ieee.org/document/67⟩. Citado 4 vezes nas páginas 51, 52, 65 e 66.

KRAWCZYK, B. et al. Ensemble learning for data stream analysis: A survey. *Information Fusion*, v. 37, p. 132–156, 2017. ISSN 1566-2535. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S1566253516302329⟩. Citado na página 57.

LIANG, N.-Y. et al. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks*, v. 17, n. 6, p. 1411–1423, nov 2006. ISSN 1045-9227. Disponível em: ⟨http://ieeexplore.ieee.org/document/4012031/⟩. Citado 4 vezes nas páginas 33, 57, 58 e 59.

LOSING, V.; HAMMER, B.; WERSING, H. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, v. 275, p. 1261–1274, 2018. ISSN 0925-2312. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S0925231217315928⟩. Citado na página 82.

LU, J. et al. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, PP, p. 1–1, 10 2018. Citado na página 56.

MA, Y. et al. Robust sparse hyperspectral unmixing with $\ell_{2,1}$ norm. *IEEE Transactions on Geoscience and Remote Sensing*, v. 55, n. 3, p. 1227–1239, March 2017. ISSN 1558-0644. Citado na página 46.

MARTÍNEZ-MARTÍNEZ, J. M. et al. Regularized extreme learning machine for regression problems. *Neurocomputing*, v. 74, n. 17, p. 3716–3721, oct 2011. ISSN 09252312. Disponível em: ⟨http://linkinghub.elsevier.com/retrieve/pii/S092523121100378X⟩. Citado na página 42.

MIETTINEN, K. *Nonlinear Multiobjective Optimization*. Springer US, 1999. (International Series in Operations Research & Management Science). ISBN 9780792382782. Disponível em: ⟨https://books.google.com.br/books?id=ha_zLdNtXSMC⟩. Citado na página 37.

MINKU, L. L. Online ensemble learning in the presence of concept drift. v. 22, n. 5, p. 730–742, 2011. Disponível em: ⟨https://etheses.bham.ac.uk/id/eprint/1334/⟩. Citado 3 vezes nas páginas 33, 56 e 121.

NEMENYI, P. *Distribution-free Multiple Comparisons*. Princeton University, 1963. Disponível em: ⟨https://books.google.com.br/books?id=nhDMtgAACAAJ⟩. Citado na página 83.

PETERSEN, K. B.; PEDERSEN, M. S. The Matrix Cookbook. *Citeseer*, v. 16, n. 4, p. 1–66, 2007. ISSN 09621083. Citado 5 vezes nas páginas 36, 58, 59, 60 e 65.

POLIKAR, R. et al. Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, v. 31, n. 4, p. 497–508, 2001. ISSN 10946977. Disponível em: ⟨http://ieeexplore.ieee.org/document/983933/⟩. Citado 4 vezes nas páginas 33, 56, 60 e 121.

RAMZAN, M. et al. A survey on using neural network based algorithms for hand written digit recognition. *International Journal of Advanced Computer Science and Applications*, v. 9, 10 2018. Citado na página 30.

RAO, C.; MITRA, S. K. *Generalized Inverse of Matrices and Its Applications*. Wiley, 1971. (Probability and Statistics Series). ISBN 9780471708216. Disponível em: ⟨https://books.google.com.br/books?id=EHpdrzuKeNYC⟩. Citado na página 41.

RUSSAKOVSKY, O. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, p. 211–252, 2015. Citado na página 32.

SANTOS, C. C. dos; SAMATELO, J. L. A.; VASSALLO, R. F. Dynamic gesture recognition by using cnns and star rgb: A temporal information condensation. *Neurocomputing*, v. 400, p. 238 – 254, 2020. ISSN 0925-2312. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S092523122030391X⟩. Citado na página 32.

SANTOS, M. M. dos; SILVA FILHO, A. G. d.; SANTOS, W. P. dos. Deep convolutional extreme learning machines: Filters combination and error model validation. *Neurocomputing*, Elsevier B.V., v. 329, p. 359–369, feb 2019. ISSN 09252312. Disponível em: ⟨https://doi.org/10.1016/j.neucom.2018.10.063https://linkinghub.elsevier.com/retrieve/pii/S0925231218312967⟩. Citado na página 30.

SEVERIANO, C. A. et al. Evolving fuzzy time series for spatio-temporal forecasting in renewable energy systems. *Renewable Energy*, v. 171, p. 764–783, 2021. ISSN 0960-1481. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0960148121002962⟩. Citado 12 vezes nas páginas 15, 109, 110, 111, 112, 113, 114, 115, 116, 118, 119 e 120.

SHAO, Z.; ER, M. J. An online sequential learning algorithm for regularized Extreme Learning Machine. *Neurocomputing*, Elsevier, v. 173, p. 778–788, 2016. ISSN 18728286. Disponível em: ⟨http://dx.doi.org/10.1016/j.neucom.2015.08.029⟩. Citado 3 vezes nas páginas 33, 59 e 129.

SILVA, B. L. S. da; INABA, F. K.; CIARELLI, P. M. A FAST DEEP STACKED NETWORK USING EXTREME LEARNING MACHINE. In: *Proceedings XXII Congresso Brasileiro de Automática*. SBA Sociedade Brasileira de Automática, 2018. ISSN 2525-8311. Disponível em: ⟨https://doi.org/10.20906/cps/cba2018-0809⟩. Citado 2 vezes nas páginas 53 e 55.

SILVA, B. L. S. da et al. Fast deep stacked networks based on extreme learning machine applied to regression problems. *Neural Networks*, v. 131, p. 14–28, 2020. ISSN 0893-6080. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0893608020302598⟩. Citado na página 123.

SILVA, B. L. S. da et al. Outlier Robust Extreme Machine Learning for multi-target regression. *Expert Systems with Applications*, v. 140, p. 112877, feb 2020. ISSN 09574174. Disponível em: ⟨https://linkinghub.elsevier.com/retrieve/pii/S0957417419305871⟩. Citado 7 vezes nas páginas 31, 42, 45, 46, 48, 81 e 86.

SOARES, S. G.; ARAÚJO, R. A dynamic and on-line ensemble regression for changing environments. *Expert Systems with Applications*, v. 42, n. 6, p. 2935–2948, 2015. ISSN 0957-4174. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0957417414007520⟩. Citado 8 vezes nas páginas 56, 57, 62, 64, 123, 126, 127 e 130.

SOARES, S. G.; ARAÚJO, R. An on-line weighted ensemble of regressor models to handle concept drifts. *Engineering Applications of Artificial Intelligence*, v. 37, p. 392–406, 2015. ISSN 0952-1976. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0952197614002437⟩. Citado 5 vezes nas páginas 57, 62, 63, 126 e 127.

SOARES, S. G.; ARAÚJO, R. An adaptive ensemble of on-line extreme learning machines with variable forgetting factor for dynamic system prediction. *Neurocomputing*, v. 171, p. 693–707, 2016. ISSN 0925-2312. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0925231215010164⟩. Citado 4 vezes nas páginas 57, 64, 126 e 127.

SOARES, S. G.; ARAÚJO, R. An adaptive ensemble of on-line Extreme Learning Machines with variable forgetting factor for dynamic system prediction. *Neurocomputing*, v. 171, p. 693–707, 2016. ISSN 18728286. Citado na página 30.

TIKHONOV, A. N.; ARSENIN, V. Y. *Solutions of Ill-posed problems*. [S.l.]: W.H. Winston, 1977. Citado na página 37.

TURK, M.; PENTLAND, A. Eigenfaces for recognition. *Journal of cognitive neuroscience*, MIT Press, v. 3, n. 1, p. 71–86, 1991. Citado 2 vezes nas páginas 54 e 70.

WANG, W.; LIU, X. The selection of input weights of extreme learning machine: A sample structure preserving point of view. *Neurocomputing*, Elsevier B.V., v. 261, p. 28–36, oct 2017. ISSN 09252312. Disponível em: ⟨http://linkinghub.elsevier.com/retrieve/pii/S0925231217302114⟩. Citado 2 vezes nas páginas 65 e 72.

WANG, Y. et al. Pr-elm: parallel regularized extreme learning machine based on cluster. *Neurocomputing*, Elsevier, v. 173, p. 1073–1081, 2016. Citado na página 31.

WATTS, M. A Decade of Kasabov's Evolving Connectionist Systems: A Review. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 39, n. 3, p. 253–269, may 2009. ISSN 1094-6977. Disponível em: ⟨http://ieeexplore.ieee.org/document/4804783/⟩. Citado na página 33.

WERBOS, P. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science. Thesis (Ph. D.). Appl. Math. Harvard University*. Tese (Doutorado), 01 1974. Citado na página 29.

WOLPERT, D. H. Stacked generalization. *Neural networks*, Elsevier, v. 5, n. 2, p. 241–259, 1992. Citado 5 vezes nas páginas 32, 52, 69, 76 e 87.

YOUNG, T. et al. Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, v. 13, p. 55–75, 08 2018. Citado 2 vezes nas páginas 31 e 32.

Yu, D.; Wang, S.; Deng, L. Sequential labeling using deep-structured conditional random fields. *IEEE Journal of Selected Topics in Signal Processing*, v. 4, n. 6, p. 965–973, 2010. Citado na página 54.

ZENG, G. et al. Modified bidirectional extreme learning machine with Gram–Schmidt orthogonalization method. *Neurocomputing*, v. 316, p. 405–414, nov 2018. ISSN 09252312. Disponível em: ⟨https://linkinghub.elsevier.com/retrieve/pii/S0925231218309639⟩. Citado na página 65.

ZHANG, K.; LUO, M. Outlier-robust extreme learning machine for regression problems. *Neurocomputing*, Elsevier, v. 151, n. P3, p. 1519–1527, mar 2015. ISSN 18728286. Disponível em: ⟨http://dx.doi.org/10.1016/j.neucom.2014.09.022http://linkinghub.elsevier.com/retrieve/pii/S0925231214012053⟩. Citado 5 vezes nas páginas 30, 42, 45, 46 e 86.

ZHEN, X. et al. Multi-target regression via robust low-rank learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 40, n. 2, p. 497–504, Feb 2018. ISSN 1939-3539. Citado na página 46.

ZHOU, H. et al. Stacked Extreme Learning Machines. *IEEE transactions on cybernetics*, PP, n. 99, p. 1, sep 2014. ISSN 2168-2275. Disponível em: ⟨http://www.ncbi.nlm.nih.gov/pubmed/25361517http://ieeexplore.ieee.org/document/6937189/⟩. Citado 8 vezes nas páginas 31, 32, 52, 53, 54, 69, 75 e 77.

ZHOU, Z.; CHEN, J.; ZHU, Z. Regularization incremental extreme learning machine with random reduced kernel for regression. *Neurocomputing*, Elsevier B.V., v. 321, p. 72–81, dec 2018. ISSN 09252312. Disponível em: ⟨https://doi.org/10.1016/j.neucom.2018.08.082https://linkinghub.elsevier.com/retrieve/pii/S0925231218311093⟩. Citado na página 30.

ZHU, S.; ZANIBBI, R. A text detection system for natural scenes with convolutional feature learning and cascaded classification. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2016. p. 625–632. Citado na página 32.

ZOU, H.; HASTIE, T. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, v. 67, p. 301–320, 2005. Citado na página 42.

ZYARAH, A. M.; KUDITHIPUDI, D. Extreme learning machine as a generalizable classification engine. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2017. p. 3371–3376. ISSN 2161-4407. Citado na página 30.

Apêndices

APÊNDICE A – Tabelas de Resultados - Métodos Offline - Problemas de Classificação

Tabela 22 – Acurácias (%) obtidas sobre o conjunto de teste para os experimentos da Seção 3.4.4.1, no formato "média (desvio padrão)".

Base/Método	R-ELM			GR-ELM			DSN			FDSN			FDSN-GR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
cnae9	72,1	**94,4**	74,0	72,0	**94,4**	74,0	55,7	90,4	61,4	57,0	88,8	57,7	57,8	88,7	60,0	68,9
	(1,1)	**(0,4)**	(0,9)	(1,1)	**(0,4)**	(1,0)	(1,9)	(1,0)	(2,0)	(1,7)	(1,3)	(2,4)	(1,8)	(1,3)	(2,2)	(1,2)
coil2000	**94,1**	**94,1**	**94,1**	**94,1**	**94,1**	**94,1**	**94,1**	**94,1**	**94,1**	**94,1**	**94,1**	**94,1**	94,0	**94,1**	**94,1**	**94,1**
	(0,0)	**(0,0)**	**(0,0)**	**(0,0)**	**(0,0)**	**(0,0)**	**(0,0)**	**(0,0)**	**(0,0)**	**(0,0)**	**(0,0)**	**(0,0)**	(0,1)	**(0,1)**	**(0,0)**	**(0,0)**
contracep,	54,9	56,3	56,2	54,7	56,3	56,0	55,3	55,9	56,3	55,4	56,5	56,9	55,7	56,3	57,0	**57,2**
	(0,6)	(0,6)	(0,4)	(0,6)	(0,6)	(0,7)	(0,9)	(0,7)	(0,7)	(0,9)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	**(0,7)**
marketing	33,3	**33,5**	33,3	33,3	33,4	33,3	32,9	33,1	33,3	32,5	33,3	33,3	32,5	33,1	33,3	33,1
	(0,3)	**(0,3)**	(0,2)	(0,3)	(0,3)	(0,2)	(0,4)	(0,5)	(0,4)	(0,5)	(0,4)	(0,5)	(0,7)	(0,5)	(0,4)	(0,4)
pageblocks	**96,8**	96,5	95,9	**96,8**	96,5	95,9	96,4	96,3	96,4	96,3	96,3	96,3	96,4	96,3	96,3	96,0
	(0,2)	(0,1)	(0,1)	**(0,2)**	(0,1)	(0,1)	(0,1)	(0,2)	(0,1)	(0,2)	(0,1)	(0,1)	(0,2)	(0,1)	(0,1)	(0,2)
penbased	**99,6**	**99,6**	**99,6**	**99,6**	**99,6**	**99,6**	99,0	99,2	99,3	98,2	99,0	99,1	98,3	99,1	99,1	98,7
	(0,1)	**(0,1)**	**(0,0)**	**(0,1)**	**(0,1)**	**(0,0)**	(0,1)	(0,1)	(0,1)	(0,2)	(0,1)	(0,1)	(0,2)	(0,1)	(0,1)	(0,2)
phoneme	**87,8**	86,6	83,9	**87,8**	86,4	79,7	85,7	85,9	85,8	85,6	85,5	84,4	85,7	85,6	85,8	85,0
	(0,4)	(0,4)	(0,2)	**(0,4)**	(0,5)	(12,2)	(0,5)	(0,5)	(0,4)	(0,4)	(0,5)	(0,6)	(0,5)	(0,5)	(0,4)	(0,5)
satimage	90,0	**90,7**	88,7	90,3	**90,7**	88,7	88,8	89,4	88,5	87,4	89,4	88,8	87,5	89,4	88,8	87,5
	(0,3)	**(0,3)**	(0,3)	(0,4)	**(0,3)**	(0,3)	(0,4)	(0,3)	(0,3)	(0,5)	(0,4)	(0,4)	(0,5)	(0,4)	(0,4)	(0,4)
segment	96,0	93,8	95,4	95,2	95,8	95,4	95,0	95,8	95,2	94,1	96,5	**96,6**	94,1	96,4	**96,6**	95,4
	(0,3)	(0,3)	(0,2)	(0,4)	(0,4)	(0,3)	(0,4)	(0,4)	(0,4)	(0,6)	(0,4)	**(0,3)**	(0,6)	(0,4)	**(0,3)**	(0,4)
spambase	93,7	**93,9**	93,6	93,5	**93,9**	93,3	93,3	93,6	93,7	92,7	93,7	93,6	93,2	93,6	93,7	93,5
	(0,3)	**(0,3)**	(0,2)	(0,3)	**(0,3)**	(0,3)	(0,4)	(0,3)	(0,3)	(0,5)	(0,2)	(0,3)	(0,3)	(0,4)	(0,3)	(0,2)
thyroid	96,2	97,3	95,8	96,0	97,2	95,7	96,6	96,9	96,7	96,2	97,4	97,5	96,3	97,5	**97,6**	96,4
	(0,1)	(0,2)	(0,2)	(0,3)	(0,5)	(0,3)	(0,1)	(0,2)	(0,2)	(0,2)	(0,2)	(0,2)	(0,2)	(0,2)	**(0,2)**	(0,3)
titanic	77,1	77,1	77,0	77,7	76,1	72,7	**78,0**	77,1	77,6	77,2	77,1	77,9	77,8	77,7	77,7	76,2
	(0,0)	(0,0)	(0,1)	(0,9)	(5,6)	(8,1)	**(0,0)**	(0,1)	(0,4)	(0,3)	(0,2)	(0,1)	(0,5)	(0,5)	(0,6)	(0,4)
vowel	47,1	41,6	40,2	45,9	40,9	42,5	47,8	52,8	51,6	45,1	55,2	**55,5**	45,1	52,4	53,3	53,2
	(2,5)	(1,7)	(1,4)	(2,7)	(1,9)	(1,3)	(3,4)	(3,0)	(2,6)	(3,6)	(3,3)	**(2,9)**	(3,6)	(3,5)	(3,0)	(2,7)
wineq-red	**62,2**	60,1	59,7	**62,2**	58,7	59,8	59,9	59,1	60,4	59,9	59,7	59,5	59,9	58,8	58,5	60,1
	(1,6)	(0,6)	(0,7)	**(1,6)**	(1,3)	(0,7)	(0,8)	(0,7)	(0,6)	(1,2)	(0,9)	(0,9)	(1,2)	(1,6)	(1,2)	(0,6)
wineq-white	57,4	54,4	54,4	**57,5**	54,4	54,2	55,1	54,5	54,2	54,2	53,4	54,5	54,2	54,8	54,5	54,8
	(0,8)	(0,5)	(0,3)	**(0,8)**	(0,5)	(0,5)	(0,6)	(0,5)	(0,4)	(0,8)	(1,1)	(0,7)	(0,8)	(0,8)	(0,7)	(0,4)

Tabela 23 – Tempos de treinamento dos experimentos da Seção 3.4.4.1, no formato "média (desvio padrão)".

Base/Método	R-ELM			GR-ELM			DSN			FDSN			FDSN-GR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
cnae9	**0,01**	0,03	**0,01**	0,02	0,05	0,02	0,03	0,05	0,05	**0,01**	**0,01**	**0,01**	0,03	0,03	0,02	**0,01**
	(0,00)	(0,01)	**(0,00)**	(0,00)	(0,01)	(0,00)	(0,00)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	(0,00)	(0,00)	(0,00)	**(0,00)**
coil2000	0,12	0,24	0,12	0,05	0,27	0,14	0,06	0,09	0,07	**0,04**	**0,04**	**0,04**	0,06	0,09	0,08	0,06
	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,00)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	(0,00)	(0,00)	(0,00)	(0,00)
contraceptive	**0,01**	0,03	**0,01**	0,02	0,05	0,05	**0,01**	0,02	**0,01**	**0,01**	**0,01**	**0,01**	0,04	0,02	**0,01**	**0,01**
	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,00)	**(0,00)**	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,00)	(0,00)	**(0,00)**	**(0,00)**

Tabela 23 – *Continuação*

Base/Método	R-ELM			GR-ELM			DSN			FDSN			FDSN-GR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
marketing	0,09 (0,01)	0,16 (0,01)	0,08 (0,01)	0,06 (0,01)	0,15 (0,01)	0,09 (0,01)	0,05 (0,00)	0,08 (0,01)	0,05 (0,00)	**0,02 (0,00)**	**0,02 (0,00)**	0,03 (0,00)	0,07 (0,01)	0,04 (0,00)	0,04 (0,00)	0,04 (0,00)
pageblocks	0,06 (0,01)	0,11 (0,01)	0,06 (0,01)	0,05 (0,00)	0,11 (0,01)	0,11 (0,00)	0,04 (0,00)	0,06 (0,00)	0,04 (0,00)	0,03 (0,00)	**0,02 (0,00)**	**0,02 (0,00)**	0,03 (0,00)	0,04 (0,00)	0,04 (0,00)	0,03 (0,00)
penbased	0,14 (0,01)	0,26 (0,01)	0,13 (0,01)	0,10 (0,01)	0,40 (0,02)	0,14 (0,01)	0,09 (0,00)	0,14 (0,01)	0,09 (0,01)	0,06 (0,01)	0,06 (0,01)	**0,04 (0,00)**	0,08 (0,01)	0,05 (0,00)	0,05 (0,00)	0,08 (0,00)
phoneme	0,06 (0,01)	0,11 (0,01)	0,05 (0,01)	0,04 (0,00)	0,11 (0,01)	0,70 (0,00)	0,04 (0,00)	0,06 (0,00)	0,03 (0,00)	**0,02 (0,00)**	0,03 (0,00)	**0,02 (0,00)**	0,04 (0,00)	0,05 (0,00)	0,04 (0,00)	0,04 (0,00)
satimage	0,08 (0,00)	0,16 (0,01)	0,08 (0,01)	0,08 (0,00)	0,14 (0,01)	0,07 (0,00)	0,05 (0,00)	0,08 (0,01)	0,06 (0,00)	0,03 (0,00)	**0,02 (0,00)**	**0,02 (0,00)**	0,04 (0,00)	0,03 (0,00)	0,05 (0,00)	0,04 (0,00)
segment	0,03 (0,00)	0,05 (0,01)	0,02 (0,00)	0,20 (0,00)	1,17 (0,10)	0,12 (0,00)	**0,01 (0,00)**	0,02 (0,00)	0,02 (0,00)	**0,01 (0,00)**	**0,01 (0,00)**	**0,01 (0,00)**	0,02 (0,00)	0,04 (0,00)	0,02 (0,00)	**0,01 (0,00)**
spambase	0,05 (0,00)	0,11 (0,01)	0,05 (0,01)	0,06 (0,00)	0,09 (0,01)	0,05 (0,00)	0,04 (0,00)	0,05 (0,00)	0,04 (0,00)	**0,02 (0,00)**	**0,02 (0,00)**	**0,02 (0,00)**	0,04 (0,00)	0,04 (0,00)	0,04 (0,00)	0,03 (0,00)
thyroid	0,09 (0,01)	0,19 (0,01)	0,09 (0,01)	0,11 (0,01)	0,17 (0,01)	0,08 (0,01)	0,05 (0,00)	0,08 (0,01)	0,05 (0,01)	0,04 (0,00)	**0,03 (0,00)**	**0,03 (0,00)**	0,06 (0,00)	0,05 (0,01)	0,05 (0,00)	0,04 (0,00)
titanic	0,03 (0,00)	0,05 (0,01)	0,02 (0,00)	0,03 (0,00)	1,03 (0,01)	0,11 (0,02)	**0,01 (0,00)**	0,02 (0,00)	**0,01 (0,00)**	**0,01 (0,00)**	**0,01 (0,00)**	**0,01 (0,00)**	0,04 (0,00)	0,07 (0,00)	0,04 (0,00)	**0,01 (0,00)**
vowel	**0,00 (0,00)**	0,02 (0,00)	**0,00 (0,00)**	0,02 (0,00)	0,74 (0,05)	0,04 (0,00)	0,01 (0,00)	0,02 (0,00)	0,01 (0,00)	0,01 (0,00)	0,01 (0,00)	0,01 (0,00)	0,01 (0,00)	0,03 (0,00)	0,03 (0,00)	0,01 (0,00)
winequalityred	0,02 (0,00)	0,04 (0,00)	0,02 (0,00)	0,04 (0,00)	0,10 (0,00)	0,13 (0,00)	**0,01 (0,00)**	0,02 (0,00)	0,02 (0,00)	**0,01 (0,00)**	**0,01 (0,00)**	**0,01 (0,00)**	**0,01 (0,00)**	0,04 (0,00)	0,03 (0,00)	**0,01 (0,00)**
winequalitywhite	0,05 (0,01)	0,10 (0,01)	0,05 (0,01)	0,05 (0,00)	0,10 (0,01)	0,07 (0,00)	**0,02 (0,00)**	0,04 (0,00)	0,04 (0,00)	**0,02 (0,00)**	**0,02 (0,00)**	**0,02 (0,00)**	0,03 (0,00)	0,04 (0,00)	0,03 (0,00)	0,03 (0,00)

Tabela 24 – Quantidade de kilobytes gastos em cada modelo dos experimentos da Seção 3.4.4.1.

Base/Método	R-ELM			GR-ELM			DSN			FDSN			FDSN-GR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
cnae9	6766	6766	6766	6766	6766	6766	7083	7083	7083	**811**	**811**	**811**	864	834	839	812
coil2000	688	688	688	688	373	**¡1**	759	759	759	105	105	105	273	627	297	106
contracep.	109	109	109	109	109	82	251	251	251	**75**	**75**	**75**	212	91	91	76
marketing	188	188	188	188	188	188	540	540	540	**168**	**168**	**168**	248	184	184	**168**
pageblocks	141	141	141	141	141	141	388	388	388	**121**	**121**	**121**	144	159	139	**121**
penbased	211	211	211	211	211	211	564	564	564	**170**	**170**	**170**	191	190	188	**170**
phoneme	63	63	63	63	59	48	134	134	134	**43**	**43**	**43**	69	70	70	**43**
satimage	336	336	336	326	335	336	548	548	548	**126**	**126**	**126**	143	142	148	127
segment	211	211	211	152	**94**	211	458	458	458	128	128	128	144	157	144	128
spambase	469	469	469	267	469	344	540	540	540	**83**	**83**	**83**	148	198	132	84
thyroid	203	203	203	106	166	175	345	345	345	**85**	**85**	**85**	108	108	115	**85**
titanic	55	55	55	21	13	**1**	161	161	161	56	56	56	156	156	154	56
vowel	172	172	172	171	**104**	172	560	560	560	180	180	180	196	200	200	181
wineq-red	164	164	164	164	158	163	482	482	482	**151**	**151**	**151**	167	269	245	152
wine-white	172	172	172	172	172	**162**	524	524	524	166	166	166	183	222	182	166

Tabela 25 – Hiperparâmetros de regularização (C) obtidos após o 5-fold para os métodos R-ELM, DSN e FDSN, nos experimentos da Seção 3.4.4.1.

Base/Método	R-ELM			DSN			FDSN		
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd
cnae9	2^{-9}	2^{1}	2^{-3}	2^{-9}	2^{-2}	2^{-4}	2^{-7}	2^{-3}	2^{-5}
coil2000	2^{-15}	2^{-4}	2^{-9}	2^{-12}	2^{-5}	2^{20}	2^{-12}	2^{-8}	2^{-8}
contracep.	2^{-9}	2^{-1}	2^{10}	2^{-8}	2^{0}	2^{-3}	2^{-8}	2^{-5}	2^{-6}
marketing	2^{-7}	2^{-4}	2^{7}	2^{-5}	2^{-5}	2^{-5}	2^{-10}	2^{-7}	2^{-1}
pageblocks	2^{0}	2^{2}	2^{19}	2^{2}	2^{12}	2^{3}	2^{3}	2^{-2}	2^{-2}
penbased	2^{3}	2^{14}	2^{19}	2^{3}	2^{8}	2^{7}	2^{3}	2^{7}	2^{4}
phoneme	2^{4}	2^{18}	2^{20}	2^{2}	2^{15}	2^{10}	2^{1}	2^{15}	2^{13}
satimage	2^{-6}	2^{4}	2^{11}	2^{4}	2^{6}	2^{7}	2^{-1}	2^{3}	2^{2}
segment	2^{-2}	2^{9}	2^{16}	2^{-3}	2^{8}	2^{1}	2^{3}	2^{-1}	2^{2}
spambase	2^{-8}	2^{1}	2^{-2}	2^{-5}	2^{-6}	2^{2}	2^{-5}	2^{-4}	2^{-3}
thyroid	2^{0}	2^{7}	2^{11}	2^{20}	2^{20}	2^{20}	2^{20}	2^{-2}	2^{20}
titanic	2^{-4}	2^{20}	2^{20}	2^{-4}	2^{8}	2^{3}	2^{-6}	2^{-3}	2^{-2}
vowel	2^{9}	2^{17}	2^{20}	2^{9}	2^{12}	2^{9}	2^{6}	2^{0}	2^{2}
wineq-red	2^{-5}	2^{-2}	2^{17}	2^{-7}	2^{1}	2^{2}	2^{-7}	2^{-4}	2^{-3}
wineq-white	2^{-8}	2^{-3}	2^{15}	2^{-8}	2^{1}	2^{2}	2^{-1}	2^{-6}	2^{-4}

Tabela 26 – Hiperparâmetros obtidos após o 5-fold para os métodos GR-ELM e FDSN-GR, nos experimentos da Seção 3.4.4.1.

Base/Método	GR-ELM						FDSN-GR					
	rfm		ae		svd		rfm		ae		svd	
	C	α	C	α	C	α	C	α	C	α	C	α
cnae9	2^{-9}	0,00	2^{1}	0,00	2^{-3}	0,00	2^{-3}	0,75	2^{-1}	0,75	2^{0}	0,50
coil2000	2^{-15}	0,00	2^{-2}	0,50	2^{-6}	1,00	2^{-1}	0,25	2^{-3}	0,50	2^{11}	0,75
contracep.	2^{-9}	0,00	2^{-1}	0,00	2^{7}	0,50	2^{-2}	1,00	2^{-1}	0,00	2^{-6}	0,00
marketing	2^{-7}	0,00	2^{-4}	0,00	2^{7}	0,00	2^{-3}	0,75	2^{-7}	0,00	2^{-1}	0,00
pageblocks	2^{0}	0,00	2^{2}	0,00	2^{18}	0,50	2^{-1}	0,25	2^{-1}	0,25	2^{5}	0,50
penbased	2^{5}	0,50	2^{19}	0,75	2^{18}	0,25	2^{3}	1,00	2^{6}	0,25	2^{7}	0,25
phoneme	2^{4}	0,00	2^{15}	0,50	2^{17}	1,00	2^{2}	0,75	2^{13}	1,00	2^{4}	0,75
satimage	2^{1}	0,50	2^{4}	0,25	2^{11}	0,00	2^{8}	1,00	2^{3}	0,00	2^{0}	0,75
segment	2^{0}	0,75	2^{10}	1,00	2^{14}	0,75	2^{3}	0,00	2^{2}	1,00	2^{2}	0,00
spambase	2^{-2}	0,25	2^{1}	0,00	2^{7}	0,75	2^{-2}	0,50	2^{3}	0,75	2^{6}	1,00
thyroid	2^{0}	0,50	2^{7}	0,75	2^{8}	0,75	2^{3}	1,00	2^{4}	0,25	2^{4}	0,50
titanic	2^{20}	0,25	2^{18}	1,00	2^{3}	1,00	2^{-1}	0,75	2^{20}	1,00	2^{14}	0,75
vowel	2^{10}	0,50	2^{12}	1,00	2^{20}	0,25	2^{6}	0,00	2^{3}	0,75	2^{4}	1,00
wineq-red	2^{-5}	0,00	2^{8}	0,50	2^{11}	0,75	2^{-7}	0,00	2^{-3}	0,75	2^{0}	1,00
wineq-white	2^{-8}	0,00	2^{-3}	0,00	2^{4}	0,25	2^{1}	0,25	2^{-3}	0,25	2^{-4}	0,00

Tabela 27 – Hiperparâmetros C e γ obtidos após o 5-fold para o método FKDSN, nos experimentos da Seção 3.4.4.1.

Base/Método	FKDSN	
	C	γ
cnae9	2^{15}	2^{12}
coil2000	2^{17}	2^{20}
contracep.	2^{15}	2^{9}
marketing	2^{0}	2^{5}
pageblocks	2^{20}	2^{6}
penbased	2^{14}	2^{7}
phoneme	2^{10}	2^{2}
satimage	2^{16}	2^{7}
segment	2^{20}	2^{8}
spambase	2^{20}	2^{8}
thyroid	2^{20}	2^{7}
titanic	2^{5}	2^{7}
vowel	2^{20}	2^{6}
wineq-red	2^{15}	2^{9}
wineq-white	2^{7}	2^{6}

Tabela 28 – Funções de ativação obtidas após o 5-fold nos experimentos da Seção 3.4.4.1. As funções sigmoide, *hard-limit*, gaussiana e multiquádrica são representadas pelas letras 's', 'h', 'r' e 'm', respectivamente.

Base/Método	R-ELM			GR-ELM			DSN			FDSN			FDSN-GR		
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd
cnae9	s	m	s	s	m	s	s	s	s	s	s	s	s	s	s
coil2000	m	m	m	m	r	m	m	m	m	m	m	m	m	m	m
contracep.	m	r	s	m	r	r	m	s	m	m	m	r	s	s	r
marketing	s	r	s	s	r	s	s	r	r	m	m	s	s	r	s
pageblocks	r	r	r	r	r	r	s	s	r	s	m	m	m	m	s
penbased	s	r	m	s	s	m	s	s	s	s	s	m	m	m	m
phoneme	r	r	r	r	r	r	r	s	r	m	s	s	m	r	r
satimage	m	r	r	s	r	r	s	s	s	s	m	m	s	m	r
segment	m	m	m	r	m	m	m	m	r	s	m	s	s	m	s
spambase	h	s	r	s	s	s	s	r	s	s	m	s	s	m	s
thyroid	s	s	r	s	s	r	s	s	r	s	m	s	s	s	s
titanic	m	r	r	s	r	h	m	m	m	m	m	m	m	m	m
vowel	s	r	r	s	r	m	s	s	m	s	m	m	s	m	m
wineq-red	h	r	m	h	m	r	m	m	s	m	m	m	m	r	s
wineq-white	h	r	s	h	r	r	m	m	s	s	h	r	s	m	r

APÊNDICE B – Tabelas de Resultados - Métodos Offline - Problemas de Regressão

Tabela 29 – Valores de aRRMSE obtidos sobre o conjunto de teste para os experimentos da Seção 3.4.4.2, no formato "média (desvio padrão)". A notação xe^y representa o número $x \times 10^y$.

Base/Método	R-ELM			GOR-ELM			DSN			FDSN			FDSN-GOR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
ANACALT	0,21 (0,01)	0,27 (0,01)	0,35 (0,03)	0,22 (0,01)	0,49 (0,06)	0,39 (0,03)	0,17 (0,01)	**0,15 (0,00)**	0,17 (0,02)	0,22 (0,02)	**0,15 (0,01)**	**0,15 (0,01)**	0,26 (0,03)	**0,15 (0,01)**	0,16 (0,02)	**0,15 (0,00)**
abalone	0,69 (0,00)	0,68 (0,00)	0,67 (0,00)	0,68 (0,00)	0,67 (0,00)	0,67 (0,00)	0,71 (0,01)	0,67 (0,00)	0,67 (0,00)	**0,66 (0,00)**	**0,66 (0,00)**	**0,66 (0,00)**	0,67 (0,00)	0,69 (0,01)	0,69 (0,01)	**0,66 (0,00)**
ailerons	0,44 (0,00)	**0,39 (0,00)**	**0,39 (0,00)**	0,44 (0,00)	0,41 (0,01)	**0,39 (0,00)**	0,50 (0,01)	0,42 (0,02)	0,40 (0,00)	0,47 (0,01)	0,53 (0,18)	0,40 (0,00)	0,47 (0,01)	0,54 (0,20)	0,41 (0,00)	**0,39 (0,00)**
airfoil	**0,54 (0,00)**	**0,54 (0,01)**	**0,54 (0,00)**	0,57 (0,00)	0,66 (0,00)	0,58 (0,00)	0,57 (0,01)	0,56 (0,02)	0,57 (0,01)	0,59 (0,02)	0,61 (0,02)	0,63 (0,02)	0,61 (0,02)	0,62 (0,02)	0,61 (0,02)	0,57 (0,01)
bank32nh	0,76 (0,01)	0,68 (0,00)	0,68 (0,00)	0,76 (0,01)	0,72 (0,01)	0,71 (0,00)	0,73 (0,00)	**0,67 (0,00)**	0,69 (0,00)	0,74 (0,01)	0,68 (0,00)	0,68 (0,00)	0,76 (0,01)	0,69 (0,01)	0,68 (0,00)	0,69 (0,00)
bank8fm	0,20 (0,00)	0,20 (0,00)	0,20 (0,00)	0,21 (0,00)	0,21 (0,00)	0,22 (0,00)	0,23 (0,00)	**0,19 (0,00)**	**0,19 (0,00)**	0,22 (0,01)	0,20 (0,00)	**0,19 (0,00)**	0,22 (0,00)	0,20 (0,00)	0,20 (0,00)	0,20 (0,00)
california	0,49 (0,01)	0,82 (0,37)	0,97 (0,44)	0,50 (0,01)	0,53 (0,02)	0,59 (0,07)	0,49 (0,00)	0,51 (0,11)	**0,48 (0,00)**	0,51 (0,00)	1,25 (1,48)	0,50 (0,01)	0,52 (0,00)	0,69 (0,46)	0,49 (0,00)	0,50 (0,00)
carbonnano.	**0,02 (0,00)**	**0,02 (0,00)**	**0,02 (0,00)**	**0,02 (0,00)**	**0,02 (0,00)**	**0,02 (0,00)**	0,05 (0,00)	0,03 (0,00)	**0,02 (0,00)**	0,05 (0,00)	0,03 (0,00)	0,03 (0,00)	0,05 (0,00)	0,03 (0,00)	0,03 (0,00)	**0,02 (0,00)**
combinedcy.	0,46 (0,00)	0,46 (0,00)	0,47 (0,01)	0,47 (0,00)	0,47 (0,00)	0,48 (0,01)	**0,45 (0,00)**	**0,45 (0,01)**	0,89 (0,25)	**0,45 (0,00)**	**0,45 (0,00)**	**0,45 (0,00)**	**0,45 (0,00)**	**0,45 (0,00)**	**0,45 (0,00)**	**0,45 (0,00)**
compactiv	0,19 (0,00)	0,19 (0,01)	0,17 (0,01)	0,22 (0,01)	0,22 (0,01)	0,21 (0,01)	0,29 (0,02)	0,17 (0,01)	0,16 (0,00)	0,31 (0,02)	0,18 (0,02)	**0,15 (0,01)**	0,35 (0,03)	0,18 (0,01)	**0,15 (0,00)**	0,23 (0,01)
concrete	0,33 (0,01)	0,33 (0,01)	0,34 (0,01)	0,36 (0,01)	0,43 (0,01)	0,41 (0,01)	0,36 (0,01)	0,33 (0,01)	**0,32 (0,01)**	0,40 (0,02)	0,37 (0,01)	0,38 (0,02)	0,40 (0,02)	0,39 (0,03)	0,41 (0,08)	0,36 (0,01)
cpuact	0,19 (0,00)	0,17 (0,01)	0,16 (0,01)	0,22 (0,01)	0,22 (0,01)	0,19 (0,01)	0,28 (0,01)	0,17 (0,01)	0,16 (0,00)	0,32 (0,02)	0,16 (0,01)	**0,15 (0,01)**	0,36 (0,05)	0,17 (0,02)	0,16 (0,01)	0,22 (0,01)
cpusmall	0,19 (0,00)	0,18 (0,01)	0,19 (0,01)	0,20 (0,01)	0,19 (0,01)	0,20 (0,01)	0,23 (0,01)	**0,17 (0,00)**	**0,17 (0,00)**	0,23 (0,01)	**0,17 (0,01)**	0,18 (0,01)	0,25 (0,02)	0,20 (0,04)	**0,17 (0,00)**	0,19 (0,01)
delta_ail	**0,56 (0,00)**	0,57 (0,00)	0,57 (0,00)	0,59 (0,01)	0,57 (0,00)	**0,56 (0,00)**	**0,56 (0,00)**	**0,56 (0,00)**	0,60 (0,03)	**0,56 (0,00)**	**0,56 (0,00)**	0,59 (0,02)	0,57 (0,00)	0,57 (0,01)	**0,56 (0,01)**	**0,56 (0,00)**
delta_elv	0,60 (0,00)	**0,59 (0,00)**	0,60 (0,00)	0,61 (0,00)	0,60 (0,00)	0,60 (0,00)	**0,59 (0,00)**	**0,59 (0,00)**	0,60 (0,00)	**0,59 (0,00)**	**0,59 (0,00)**	**0,59 (0,00)**	0,60 (0,00)	**0,59 (0,00)**	**0,59 (0,00)**	**0,59 (0,00)**
ele2	**0,04 (0,00)**	0,05 (0,00)	0,05 (0,00)	0,05 (0,00)	0,07 (0,00)	0,07 (0,00)	0,05 (0,00)	**0,04 (0,00)**	0,05 (0,00)	**0,04 (0,00)**	**0,04 (0,00)**	**0,04 (0,00)**	0,05 (0,00)	0,05 (0,00)	0,05 (0,00)	0,06 (0,01)
elevators	0,35 (0,00)	0,34 (0,00)	0,34 (0,00)	0,36 (0,00)	0,36 (0,03)	0,34 (0,01)	0,42 (0,01)	0,32 (0,00)	**0,30 (0,00)**	0,41 (0,01)	0,34 (0,01)	**0,30 (0,00)**	0,41 (0,01)	0,34 (0,01)	**0,30 (0,00)**	0,32 (0,00)
friedman	0,25 (0,00)	0,24 (0,01)	**0,22 (0,00)**	0,26 (0,00)	0,29 (0,02)	0,32 (0,01)	0,27 (0,01)	0,24 (0,01)	**0,22 (0,00)**	0,27 (0,01)	0,25 (0,01)	0,26 (0,01)	0,27 (0,01)	0,27 (0,02)	0,27 (0,02)	**0,22 (0,00)**
house	0,71 (0,01)	0,94 (0,12)	4,65 (3,15)	0,72 (0,00)	0,83 (0,05)	0,85 (0,10)	0,70 (0,01)	0,69 (0,01)	**0,68 (0,01)**	0,72 (0,01)	0,74 (0,16)	0,70 (0,02)	0,80 (0,06)	0,72 (0,20)	**0,68 (0,01)**	0,69 (0,00)
house8l	**0,58 (0,00)**	0,59 (0,00)	0,61 (0,02)	0,61 (0,01)	0,60 (0,00)	0,62 (0,00)	0,61 (0,00)	0,59 (0,00)	0,60 (0,01)	0,61 (0,00)	0,62 (0,04)	0,61 (0,01)	0,64 (0,03)	0,64 (0,05)	0,61 (0,01)	0,61 (0,00)
kin8nm	**0,38 (0,01)**	**0,38 (0,01)**	0,39 (0,00)	**0,38 (0,01)**	0,55 (0,02)	0,54 (0,01)	0,46 (0,01)	0,40 (0,01)	0,41 (0,01)	0,57 (0,03)	0,51 (0,04)	0,47 (0,01)	0,51 (0,02)	0,48 (0,04)	0,45 (0,02)	0,48 (0,01)

Tabela 29 – *Continuação*

Base/Método	R-ELM			GOR-ELM			DSN			FDSN			FDSN-GOR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
laser	0,10	**0,08**	0,11	0,15	0,36	0,30	0,14	0,10	0,09	0,13	0,12	0,13	0,13	0,18	0,13	0,13
	(0,01)	**(0,01)**	(0,01)	(0,00)	(0,04)	(0,03)	(0,01)	(0,01)	(0,01)	(0,01)	(0,02)	(0,02)	(0,02)	(0,08)	(0,03)	(0,01)
mortgage	**0,02**	**0,02**	**0,02**	**0,02**	0,03	0,03	0,03	0,03	**0,02**	0,03	0,03	**0,02**	0,03	0,03	0,03	0,03
	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	(0,00)	(0,00)	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,00)	(0,00)
mv	0,05	**0,02**	**0,02**	0,05	0,03	0,03	0,15	0,04	0,03	0,18	0,05	0,04	0,16	0,05	0,04	0,05
	(0,00)	**(0,00)**	**(0,00)**	(0,00)	(0,00)	(0,00)	(0,01)	(0,00)	(0,00)	(0,01)	(0,01)	(0,00)	(0,01)	(0,01)	(0,00)	(0,00)
navalprop.	**0,00**	0,01	0,01	**0,00**	0,33	0,03	0,04	0,01	0,01	0,04	0,05	0,03	0,04	0,07	0,09	0,05
	(0,00)	(0,00)	(0,00)	**(0,00)**	(0,01)	(0,00)	(0,01)	(0,00)	(0,00)	(0,00)	(0,01)	(0,00)	(0,01)	(0,01)	(0,03)	(0,01)
plastic	**0,43**	**0,43**	**0,43**	**0,43**	0,45	0,45	**0,43**	**0,43**	**0,43**	**0,43**	**0,43**	**0,43**	0,44	0,44	0,50	**0,43**
	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,00)	(0,00)	(0,06)	**(0,00)**
pole	0,36	0,39	0,44	0,37	0,40	0,45	0,32	0,29	0,33	0,41	**0,28**	0,35	0,41	**0,28**	0,32	0,33
	(0,01)	(0,01)	(0,02)	(0,01)	(0,01)	(0,02)	(0,01)	(0,01)	(0,01)	(0,02)	**(0,01)**	(0,01)	(0,02)	**(0,01)**	(0,01)	(0,01)
puma32h	0,96	0,96	0,96	0,99	1,00	1,00	0,89	**0,88**	0,90	0,89	0,89	**0,88**	0,89	0,91	0,91	**0,88**
	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,00)	**(0,00)**	(0,01)	(0,00)	(0,01)	**(0,01)**	(0,00)	(0,01)	(0,01)	**(0,00)**
puma8nh	0,61	0,59	0,59	0,63	0,63	0,63	0,63	0,59	0,59	0,68	0,61	**0,58**	0,66	0,60	0,60	0,64
	(0,00)	(0,00)	(0,00)	(0,00)	(0,02)	(0,01)	(0,00)	(0,00)	(0,00)	(0,01)	(0,01)	**(0,00)**	(0,01)	(0,00)	(0,00)	(0,01)
quake	2,49	1,16	1,07	2,71	1,06	1,12	18,47	$1{,}8e^4$	$9{,}1e^5$	21,04	$2{,}1e^3$	$6{,}6e^3$	2,77	8,64	5,12	**1,03**
	(0,17)	(0,16)	(0,05)	(0,18)	(0,01)	(0,04)	(30,9)	$(7{,}2e^4)$	$(5{,}1e^6)$	(71,36)	$(5{,}0e^3)$	$(1{,}3e^4)$	(2,89)	(32,71)	(8,14)	**(0,01)**
rf1	0,15	0,15	0,20	**0,10**	0,11	0,17	0,28	0,68	0,24	0,28	0,45	0,26	0,26	1,09	0,20	0,23
	(0,01)	(0,01)	(0,01)	**(0,01)**	(0,01)	(0,01)	(0,01)	(0,14)	(0,01)	(0,01)	(0,11)	(0,01)	(0,01)	(0,60)	(0,01)	(0,01)
rf2	0,51	0,49	0,49	0,28	**0,14**	**0,14**	0,58	0,42	0,46	0,57	0,73	0,49	0,58	0,68	0,43	0,34
	(0,01)	(0,01)	(0,01)	(0,02)	**(0,01)**	**(0,01)**	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)
scm1d	0,37	**0,34**	0,35	0,38	**0,34**	0,35	0,48	0,44	0,40	0,49	0,48	0,44	0,49	0,46	0,44	0,38
	(0,00)	**(0,00)**	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,01)	(0,00)	(0,01)	(0,01)	(0,00)	(0,01)	(0,00)	(0,00)	(0,00)
scm20d	0,49	**0,44**	0,50	0,50	**0,44**	0,50	0,62	0,59	0,57	0,65	0,62	0,61	0,66	0,62	0,61	0,60
	(0,00)	**(0,00)**	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,00)	(0,00)	(0,00)	(0,01)	(0,00)	(0,01)	(0,01)	(0,00)	(0,00)
stock	**0,11**	0,12	0,13	0,12	0,22	0,17	0,13	0,12	0,12	0,13	0,13	0,14	0,14	0,14	0,14	0,14
	(0,00)	(0,00)	(0,00)	(0,00)	(0,00)	(0,00)	(0,00)	(0,00)	(0,00)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	(0,00)
superconduc.	0,22	0,15	**0,14**	0,23	0,17	0,15	0,38	0,28	0,23	0,42	0,35	0,28	0,42	0,35	0,28	0,23
	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,00)	(0,01)	(0,00)	(0,00)	(0,01)	(0,01)	(0,00)	(0,01)	(0,01)	(0,00)	(0,00)
tic	1,09	1,41	1,66	1,03	1,17	1,03	0,99	**0,98**	0,99	0,99	1,02	0,99	1,04	1,03	1,00	**0,98**
	(0,02)	(0,10)	(0,24)	(0,00)	(0,05)	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,04)	(0,01)	(0,00)	(0,05)	(0,01)	**(0,00)**
treasury	0,06	**0,05**	0,06	0,07	0,07	0,06	0,07	0,08	0,06	0,07	1,21	0,06	0,07	0,10	0,06	0,06
	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,00)	(0,00)	(0,00)	(0,02)	(0,00)	(0,00)	(3,80)	(0,00)	(0,00)	(0,09)	(0,00)	(0,00)
wankara	0,09	0,09	**0,08**	0,09	0,10	**0,08**	0,12	0,10	0,10	0,11	0,19	0,11	0,11	0,12	0,10	0,10
	(0,00)	(0,01)	**(0,00)**	(0,00)	(0,02)	**(0,00)**	(0,01)	(0,01)	(0,01)	(0,01)	(0,13)	(0,03)	(0,01)	(0,03)	(0,01)	(0,00)
wizmir	0,09	**0,08**	**0,08**	0,09	**0,08**	**0,08**	0,12	0,09	**0,08**	0,11	0,11	**0,08**	0,10	0,10	**0,08**	**0,08**
	(0,00)	**(0,00)**	**(0,00)**	(0,00)	**(0,00)**	**(0,00)**	(0,01)	(0,00)	**(0,00)**	(0,01)	(0,01)	**(0,00)**	(0,01)	(0,00)	**(0,00)**	**(0,00)**
wq	8,61	22,56	30,04	6,29	1,34	1,77	1,12	1,52	1,90	1,12	1,51	1,54	1,11	1,55	1,55	**0,96**
	(0,54)	(2,26)	(3,20)	(0,30)	(0,05)	(0,09)	(0,02)	(0,35)	(0,28)	(0,03)	(0,19)	(0,15)	(0,02)	(0,24)	(0,21)	**(0,00)**

Tabela 30 – Tempos de treinamento dos experimentos da Seção 3.4.4.2, no formato "média (desvio padrão)".

Base/Método	R-ELM			GOR-ELM			DSN			FDSN			FDSN-GOR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
ANACALT	0,04	0,09	0,04	0,21	0,95	0,76	0,02	0,03	0,03	0,02	**0,01**	**0,01**	0,37	0,03	0,03	0,02
	(0,00)	(0,00)	(0,00)	(0,03)	(0,04)	(0,15)	(0,00)	(0,00)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	(0,00)	(0,00)	(0,00)	(0,00)
abalone	0,05	0,09	0,05	0,95	0,99	0,96	**0,02**	0,05	0,03	**0,02**	**0,02**	**0,02**	0,37	0,04	0,04	**0,02**
	(0,01)	(0,01)	(0,00)	(0,03)	(0,03)	(0,03)	**(0,00)**	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**
ailerons	0,17	0,33	0,16	4,37	4,70	4,56	0,11	0,12	0,10	0,07	**0,06**	0,07	1,19	0,09	0,11	0,09
	(0,01)	(0,01)	(0,01)	(0,24)	(0,04)	(0,05)	(0,01)	(0,01)	(0,01)	(0,01)	**(0,00)**	(0,01)	(0,04)	(0,00)	(0,01)	(0,00)
airfoil	0,02	0,04	0,02	0,24	0,24	0,22	**0,01**	0,02	**0,01**	**0,01**	**0,01**	**0,01**	0,20	0,02	0,02	**0,01**
	(0,00)	(0,00)	(0,00)	(0,01)	(0,00)	(0,00)	**(0,00)**	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,00)	(0,00)	(0,00)	**(0,00)**

Tabela 30 – *Continuação*

Base/Método	R-ELM			GOR-ELM			DSN			FDSN			FDSN-GOR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
bank32nh	0,10	0,20	0,10	1,85	1,92	1,84	0,06	0,06	0,06	0,04	**0,03**	0,04	0,65	0,07	0,07	0,05
	(0,01)	(0,01)	(0,01)	(0,03)	(0,02)	(0,03)	(0,00)	(0,00)	(0,01)	(0,00)	**(0,00)**	(0,00)	(0,01)	(0,01)	(0,01)	(0,00)
bank8fm	0,10	0,19	0,10	1,84	1,95	1,83	0,06	0,09	0,05	0,04	**0,03**	0,04	0,65	0,06	0,07	0,05
	(0,01)	(0,01)	(0,01)	(0,04)	(0,03)	(0,02)	(0,00)	(0,01)	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,02)	(0,00)	(0,01)	(0,00)
california	0,25	0,47	0,24	6,37	6,59	6,33	0,15	0,19	0,12	0,10	**0,09**	0,10	2,88	0,15	0,15	0,13
	(0,01)	(0,01)	(0,01)	(0,04)	(0,05)	(0,04)	(0,01)	(0,01)	(0,01)	(0,01)	**(0,01)**	(0,01)	(0,09)	(0,01)	(0,01)	(0,00)
carbonnano.	0,14	0,26	0,13	4,03	4,18	4,06	0,08	0,08	0,08	**0,05**	0,06	**0,05**	1,22	0,12	0,11	0,07
	(0,01)	(0,01)	(0,01)	(0,06)	(0,06)	(0,06)	(0,00)	(0,00)	(0,01)	**(0,00)**	(0,00)	**(0,00)**	(0,02)	(0,01)	(0,01)	(0,00)
combinedcy.	0,12	0,23	0,12	2,27	3,74	2,77	0,07	0,11	0,07	0,05	0,05	**0,04**	0,98	0,10	0,10	0,06
	(0,01)	(0,01)	(0,01)	(0,22)	(0,06)	(0,85)	(0,01)	(0,01)	(0,01)	(0,00)	(0,00)	**(0,00)**	(0,03)	(0,01)	(0,01)	(0,00)
compactiv	0,10	0,19	0,10	1,62	1,95	1,83	**0,03**	0,06	0,06	0,04	**0,03**	**0,03**	0,65	0,06	0,06	0,05
	(0,01)	(0,01)	(0,01)	(0,13)	(0,03)	(0,02)	**(0,00)**	(0,00)	(0,01)	(0,00)	**(0,00)**	**(0,00)**	(0,02)	(0,00)	(0,00)	(0,00)
concrete	**0,01**	0,02	**0,01**	0,10	0,14	0,12	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	0,17	0,02	0,02	**0,01**
	(0,00)	(0,00)	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**
cpuact	0,10	0,19	0,10	1,84	1,95	1,84	**0,03**	0,06	0,06	0,04	**0,03**	0,04	0,65	0,06	0,07	0,05
	(0,01)	(0,01)	(0,01)	(0,02)	(0,03)	(0,02)	**(0,00)**	(0,00)	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,02)	(0,00)	(0,01)	(0,00)
cpusmall	0,10	0,19	0,10	1,85	1,93	1,83	0,06	0,06	0,05	**0,03**	**0,03**	0,04	0,64	0,06	0,07	0,05
	(0,01)	(0,01)	(0,01)	(0,02)	(0,03)	(0,02)	(0,00)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	(0,00)	(0,01)	(0,00)	(0,01)	(0,00)
delta_ail	0,09	0,18	0,08	1,59	1,68	1,59	0,05	0,05	0,04	**0,03**	**0,03**	**0,03**	0,61	0,07	0,06	0,04
	(0,01)	(0,01)	(0,01)	(0,02)	(0,02)	(0,02)	(0,00)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	(0,02)	(0,01)	(0,01)	(0,00)
delta_elv	0,12	0,23	0,12	3,12	3,25	3,11	0,06	0,10	**0,04**	**0,04**	0,05	**0,04**	0,73	0,08	0,08	0,05
	(0,01)	(0,01)	(0,01)	(0,03)	(0,04)	(0,04)	(0,00)	(0,01)	**(0,00)**	**(0,00)**	(0,00)	**(0,00)**	(0,02)	(0,01)	(0,01)	(0,00)
ele2	**0,01**	0,02	**0,01**	0,12	0,14	0,13	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	0,17	0,02	0,02	**0,01**
	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**
elevators	0,21	0,39	0,19	5,38	5,56	5,39	0,12	0,19	0,11	**0,08**	0,09	**0,08**	1,72	0,12	0,12	0,10
	(0,01)	(0,01)	(0,01)	(0,04)	(0,04)	(0,05)	(0,01)	(0,01)	(0,01)	**(0,01)**	(0,01)	**(0,01)**	(0,08)	(0,01)	(0,01)	(0,00)
friedman	**0,01**	0,03	**0,01**	0,15	0,18	0,16	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	0,18	0,02	0,02	**0,01**
	(0,00)	(0,00)	**(0,00)**	(0,01)	(0,01)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,00)	(0,00)	(0,00)	**(0,00)**
house	0,28	0,55	0,26	7,24	7,52	7,23	0,13	0,22	0,16	0,10	0,11	**0,09**	3,41	0,18	0,16	0,15
	(0,01)	(0,01)	(0,01)	(0,05)	(0,05)	(0,04)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	**(0,01)**	(0,09)	(0,01)	(0,01)	(0,01)
house8l	0,28	0,55	0,26	7,33	7,52	7,32	0,13	0,22	0,13	**0,10**	**0,10**	0,11	3,38	0,18	0,18	0,15
	(0,01)	(0,02)	(0,01)	(0,04)	(0,05)	(0,04)	(0,01)	(0,01)	(0,00)	**(0,01)**	**(0,01)**	(0,01)	(0,08)	(0,01)	(0,01)	(0,01)
kin8nm	0,10	0,19	0,10	2,70	2,78	2,68	0,06	0,08	0,05	0,04	**0,03**	0,04	0,67	0,08	0,07	0,05
	(0,01)	(0,01)	(0,01)	(0,04)	(0,03)	(0,04)	(0,00)	(0,01)	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,01)	(0,01)	(0,01)	(0,00)
laser	**0,01**	0,02	**0,01**	0,12	0,13	0,08	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	0,16	0,02	0,02	**0,01**
	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,01)	(0,01)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**
mortgage	**0,01**	0,02	**0,01**	0,12	0,14	0,13	**0,01**	0,02	**0,01**	**0,01**	**0,01**	**0,01**	0,17	0,02	0,02	**0,01**
	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,00)	**(0,00)**	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**
mv	0,46	0,92	0,45	13,12	13,54	13,07	0,32	0,51	0,31	**0,21**	**0,21**	**0,21**	8,16	0,36	0,36	0,32
	(0,01)	(0,01)	(0,01)	(0,08)	(0,06)	(0,08)	(0,01)	(0,02)	(0,01)	**(0,01)**	**(0,01)**	**(0,01)**	(0,05)	(0,01)	(0,01)	(0,01)
navalprop.	0,15	0,29	0,14	4,38	4,47	4,34	0,09	0,09	**0,06**	**0,06**	**0,06**	**0,06**	1,48	0,12	0,11	0,08
	(0,01)	(0,01)	(0,01)	(0,06)	(0,06)	(0,06)	(0,01)	(0,00)	**(0,00)**	**(0,00)**	**(0,01)**	**(0,00)**	(0,02)	(0,01)	(0,01)	(0,00)
plastic	0,02	0,04	0,02	0,26	0,27	0,26	**0,01**	0,02	0,02	**0,01**	**0,01**	**0,01**	0,21	0,02	0,03	**0,01**
	(0,00)	(0,00)	(0,00)	(0,01)	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**
pole	0,18	0,36	0,18	4,92	5,16	4,98	0,11	0,16	0,09	0,07	0,07	**0,06**	1,42	0,10	0,11	0,10
	(0,01)	(0,01)	(0,01)	(0,04)	(0,05)	(0,04)	(0,01)	(0,01)	(0,01)	(0,01)	(0,01)	**(0,01)**	(0,04)	(0,01)	(0,01)	(0,01)
puma32h	0,10	0,20	0,10	2,73	2,80	2,69	0,06	0,10	0,06	0,04	**0,03**	0,04	0,68	0,06	0,06	0,05
	(0,01)	(0,01)	(0,01)	(0,04)	(0,04)	(0,04)	(0,01)	(0,01)	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,02)	(0,00)	(0,00)	(0,00)
puma8nh	0,10	0,19	0,10	2,75	2,78	2,71	0,05	0,09	0,05	**0,04**	**0,04**	**0,04**	0,65	0,07	0,06	0,05
	(0,01)	(0,01)	(0,01)	(0,04)	(0,04)	(0,03)	(0,00)	(0,01)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	(0,02)	(0,01)	(0,01)	(0,00)
quake	0,02	0,05	0,02	0,45	0,46	0,45	**0,01**	0,02	**0,01**	**0,01**	**0,01**	**0,01**	0,24	0,02	0,02	**0,01**
	(0,00)	(0,00)	(0,00)	(0,04)	(0,05)	(0,03)	**(0,00)**	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,00)	(0,00)	(0,00)	**(0,00)**
rf1	0,12	0,23	0,11	1,75	2,25	1,51	0,08	0,08	0,08	**0,04**	**0,04**	**0,04**	0,39	0,11	0,12	0,06
	(0,01)	(0,01)	(0,01)	(0,09)	(0,12)	(0,12)	(0,00)	(0,00)	(0,01)	**(0,00)**	**(0,00)**	**(0,00)**	(0,02)	(0,00)	(0,01)	(0,00)
rf2	0,11	0,24	0,11	2,07	3,32	2,53	0,19	0,24	0,20	**0,04**	**0,04**	**0,04**	0,47	0,12	0,12	0,06
	(0,01)	(0,01)	(0,01)	(0,12)	(0,06)	(0,15)	(0,02)	(0,02)	(0,01)	**(0,00)**	**(0,00)**	**(0,00)**	(0,02)	(0,00)	(0,01)	(0,00)

Tabela 30 – *Continuação*

Base/Método	R-ELM			GOR-ELM			DSN			FDSN			FDSN-GOR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
scm1d	0,14	0,26	0,13	1,33	2,39	1,23	0,13	0,17	0,14	**0,05**	0,06	**0,05**	0,58	0,17	0,16	0,07
	(0,01)	(0,01)	(0,01)	(0,08)	(0,19)	(0,10)	(0,01)	(0,01)	(0,01)	**(0,01)**	(0,01)	**(0,00)**	(0,04)	(0,01)	(0,01)	(0,00)
scm20d	0,12	0,23	0,11	1,53	3,08	4,22	0,08	0,10	0,09	**0,04**	0,05	0,05	0,66	0,15	0,15	0,06
	(0,01)	(0,01)	(0,00)	(0,06)	(0,27)	(0,07)	(0,00)	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,02)	(0,01)	(0,01)	(0,00)
stock	**0,01**	0,02	**0,01**	0,06	0,12	0,11	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	0,17	0,02	0,02	**0,01**
	(0,00)	(0,00)	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,00)	(0,00)	(0,00)	**(0,00)**
superconduc.	0,26	0,50	0,26	4,77	7,57	7,33	0,18	0,25	0,19	0,11	**0,10**	0,11	1,51	0,19	0,20	0,15
	(0,01)	(0,02)	(0,01)	(0,47)	(0,11)	(0,10)	(0,01)	(0,01)	(0,01)	(0,01)	**(0,01)**	(0,01)	(0,08)	(0,01)	(0,02)	(0,01)
tic	0,12	0,26	0,13	3,29	3,41	3,29	0,08	0,09	0,09	0,05	**0,04**	**0,04**	0,25	0,07	0,08	0,06
	(0,01)	(0,01)	(0,01)	(0,05)	(0,05)	(0,05)	(0,00)	(0,00)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	(0,01)	(0,00)	(0,01)	(0,00)
treasury	**0,01**	0,02	**0,01**	0,12	0,14	0,12	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	0,17	0,02	0,02	**0,01**
	(0,00)	(0,00)	**(0,00)**	(0,00)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**
wankara	0,02	0,04	0,02	0,23	0,27	0,25	0,02	0,02	0,02	**0,01**	**0,01**	**0,01**	0,20	0,02	0,02	**0,01**
	(0,00)	(0,00)	(0,00)	(0,02)	(0,01)	(0,00)	(0,00)	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**
wizmir	**0,01**	0,03	**0,01**	0,20	0,23	0,22	**0,01**	0,02	**0,01**	**0,01**	**0,01**	**0,01**	0,20	0,02	0,02	**0,01**
	(0,00)	(0,00)	**(0,00)**	(0,02)	(0,01)	(0,00)	**(0,00)**	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	**(0,00)**	(0,01)	(0,00)	(0,00)	**(0,00)**
wq	**0,01**	0,02	**0,01**	0,63	0,65	0,63	**0,01**	0,02	0,02	**0,01**	**0,01**	**0,01**	0,28	0,06	0,06	**0,01**
	(0,00)	(0,00)	**(0,00)**	(0,01)	(0,01)	(0,01)	**(0,00)**	(0,00)	(0,00)	**(0,00)**	**(0,00)**	**(0,00)**	(0,02)	(0,00)	(0,01)	**(0,00)**

Tabela 31 – Quantidade de kilobytes gastos em cada modelo dos experimentos da Seção 3.4.4.2.

Base/Método	R-ELM			GOR-ELM			DSN			FDSN			FDSN-GOR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
ANACALT	70	70	70	70	70	70	107	107	107	**29**	**29**	**29**	46	45	45	30
abalone	78	78	78	78	78	78	114	114	114	**30**	**30**	**30**	67	54	58	31
ailerons	328	328	328	328	328	328	364	364	364	**55**	**55**	**55**	147	71	87	56
airfoil	55	55	55	55	55	55	91	91	91	**28**	**28**	**28**	62	55	57	**28**
bank32nh	266	266	266	266	266	266	302	302	302	**49**	**49**	**49**	91	133	71	**49**
bank8fm	78	78	78	78	78	78	114	114	114	**30**	**30**	**30**	60	46	46	31
california	78	78	78	78	78	78	114	114	114	**30**	**30**	**30**	49	61	53	31
carbonnano.	78	78	78	78	78	78	220	220	220	**72**	**72**	**72**	89	89	89	73
combinedcy.	**55**	**55**	**55**	**55**	**55**	**55**	196	196	196	70	70	70	86	86	86	70
compactiv	180	180	180	180	180	180	216	216	216	**40**	**40**	**40**	81	56	56	41
concrete	78	78	78	78	78	78	114	114	114	**30**	**30**	**30**	63	60	59	31
cpuact	180	180	180	180	180	180	216	216	216	**40**	**40**	**40**	95	56	64	41
cpusmall	109	109	109	110	110	110	146	146	146	**33**	**33**	**33**	61	49	57	34
delta_ail	55	55	55	55	55	55	91	91	91	**28**	**28**	**28**	54	46	44	**28**
delta_elv	63	63	63	63	63	63	99	99	99	**29**	**29**	**29**	47	49	49	**29**
ele2	47	47	47	47	47	47	83	83	83	**27**	**27**	**27**	53	43	43	**27**
elevators	156	156	156	156	156	156	192	192	192	**38**	**38**	**38**	77	54	54	**38**
friedman	55	55	55	55	55	55	91	91	91	**28**	**28**	**28**	61	60	59	**28**
house	141	141	141	141	141	141	177	177	177	**36**	**36**	**36**	93	60	93	37
house8l	78	78	78	78	78	78	114	114	114	**30**	**30**	**30**	61	64	50	31
kin8nm	78	78	78	78	78	78	114	114	114	**30**	**30**	**30**	62	58	56	31
laser	47	47	47	47	47	47	83	83	83	**27**	**27**	**27**	55	43	43	**27**
mortgage	133	133	133	133	133	133	169	169	169	**36**	**36**	**36**	75	51	51	**36**
mv	94	94	94	94	94	94	130	130	130	**32**	**32**	**32**	61	47	47	**32**
navalprop.	156	156	156	156	156	156	263	263	263	**66**	**66**	**66**	82	83	94	**66**
plastic	31	31	31	31	31	31	67	67	67	**25**	**25**	**25**	63	59	64	26
pole	219	219	219	219	219	219	255	255	255	**44**	**44**	**44**	112	60	105	45
puma32h	266	266	266	266	266	266	302	302	302	**49**	**49**	**49**	127	65	65	**49**
puma8nh	78	78	78	78	78	78	114	114	114	**30**	**30**	**30**	57	46	46	31
quake	39	39	39	39	39	39	75	75	75	**26**	**26**	**26**	42	48	48	27
rf1	570	570	570	570	570	570	853	853	853	**178**	**178**	**178**	194	194	194	**178**

Tabela 31 – *Continuação*

Base/Método	R-ELM			GOR-ELM			DSN			FDSN			FDSN-GOR			FKDSN
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	
rf2	4570	4570	4570	4570	4570	4570	4853	4853	4853	**578**	**578**	**578**	594	594	595	**578**
scm1d	2320	2320	2320	2320	2320	2320	2884	2884	2884	**465**	**465**	**465**	481	499	481	466
scm20d	609	609	609	610	610	610	1173	1173	1173	**294**	**294**	**294**	310	310	310	295
stock	86	86	86	86	86	86	122	122	122	**31**	**31**	**31**	55	60	68	**31**
superconduc.	656	656	656	656	656	656	798	798	798	**130**	**130**	**130**	164	146	146	131
tic	680	680	680	680	680	680	716	716	716	**90**	**90**	**90**	106	106	285	91
treasury	133	133	133	133	133	133	169	169	169	**36**	**36**	**36**	81	51	51	**36**
wankara	86	86	86	86	86	86	122	122	122	**31**	**31**	**31**	67	47	47	**31**
wizmir	86	86	86	86	86	86	122	122	122	**31**	**31**	**31**	63	47	47	**31**
wq	242	242	242	242	242	242	735	735	735	**229**	**229**	**229**	245	245	245	230

Tabela 32 – Hiperparâmetros de regularização (C) obtidos após o 5-fold para os métodos R-ELM, DSN e FDSN, nos experimentos da Seção 3.4.4.2.

Base/Método	R-ELM			DSN			FDSN		
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd
ANACALT	2^{-2}	2^{20}	2^{-1}	2^{-5}	2^{0}	2^{9}	2^{2}	2^{-2}	2^{-2}
abalone	2^{5}	2^{11}	2^{19}	2^{3}	2^{3}	2^{1}	2^{1}	2^{1}	2^{4}
ailerons	2^{-4}	2^{3}	2^{-1}	2^{20}	2^{-1}	2^{3}	2^{20}	2^{20}	2^{20}
airfoil	2^{6}	2^{19}	2^{20}	2^{1}	2^{4}	2^{4}	2^{-2}	2^{-2}	2^{-1}
bank32nh	2^{20}	2^{10}	2^{5}	2^{20}	2^{20}	2^{20}	2^{20}	2^{0}	2^{20}
bank8fm	2^{-1}	2^{6}	2^{17}	2^{20}	2^{5}	2^{11}	2^{0}	2^{4}	2^{8}
california	2^{5}	2^{13}	2^{15}	2^{20}	2^{13}	2^{5}	2^{20}	2^{12}	2^{14}
carbonnano.	2^{11}	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{11}	2^{2}	2^{5}
combinedcy.	2^{4}	2^{20}	2^{20}	2^{9}	2^{10}	2^{14}	2^{6}	2^{6}	2^{7}
compactiv	2^{-8}	2^{9}	2^{5}	2^{20}	2^{4}	2^{5}	2^{3}	2^{0}	2^{3}
concrete	2^{4}	2^{20}	2^{18}	2^{4}	2^{6}	2^{10}	2^{5}	2^{4}	2^{8}
cpuact	2^{-7}	2^{9}	2^{7}	2^{20}	2^{20}	2^{5}	2^{20}	2^{-2}	2^{4}
cpusmall	2^{1}	2^{1}	2^{7}	2^{3}	2^{6}	2^{7}	2^{-3}	2^{0}	2^{11}
delta_ail	2^{0}	2^{8}	2^{15}	2^{0}	2^{1}	2^{12}	2^{2}	2^{1}	2^{9}
delta_elv	2^{2}	2^{7}	2^{20}	2^{0}	2^{4}	2^{7}	2^{2}	2^{2}	2^{4}
ele2	2^{6}	2^{14}	2^{20}	2^{3}	2^{11}	2^{10}	2^{6}	2^{11}	2^{10}
elevators	2^{-3}	2^{3}	2^{1}	2^{20}	2^{20}	2^{6}	2^{20}	2^{20}	2^{7}
friedman	2^{5}	2^{9}	2^{17}	2^{5}	2^{6}	2^{11}	2^{10}	2^{20}	2^{7}
house	2^{20}	2^{12}	2^{15}	2^{20}	2^{20}	2^{20}	2^{20}	2^{3}	2^{20}
house8l	2^{1}	2^{7}	2^{15}	2^{20}	2^{7}	2^{6}	2^{20}	2^{6}	2^{11}
kin8nm	2^{-1}	2^{20}	2^{20}	2^{20}	2^{11}	2^{5}	2^{20}	2^{20}	2^{10}
laser	2^{12}	2^{20}	2^{20}	2^{-1}	2^{5}	2^{8}	2^{5}	2^{6}	2^{6}
mortgage	2^{10}	2^{20}	2^{20}	2^{10}	2^{10}	2^{13}	2^{9}	2^{11}	2^{13}
mv	2^{-1}	2^{20}	2^{20}	2^{20}	2^{9}	2^{11}	2^{20}	2^{5}	2^{8}
navalprop.	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{15}	2^{20}
plastic	2^{8}	2^{20}	2^{20}	2^{0}	2^{1}	2^{12}	2^{0}	2^{2}	2^{2}

Tabela 32 – *Continuação*

Base/Método	R-ELM			DSN			FDSN		
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd
pole	2^{-6}	2^{-1}	2^{-3}	2^{20}	2^{20}	2^{20}	2^{20}	2^{-3}	2^{-2}
puma32h	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{10}	2^{20}
puma8nh	2^{2}	2^{10}	2^{19}	2^{20}	2^{12}	2^{10}	2^{0}	2^{1}	2^{3}
quake	2^{20}	2^{13}	2^{20}	2^{20}	2^{20}	2^{15}	2^{15}	2^{17}	2^{11}
rf1	2^{0}	2^{0}	2^{-3}	2^{0}	2^{1}	2^{3}	2^{-1}	2^{-8}	2^{4}
rf2	2^{-1}	2^{-1}	2^{4}	2^{20}	2^{20}	2^{20}	2^{20}	2^{-4}	2^{20}
scm1d	2^{-6}	2^{2}	2^{-3}	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{-2}
scm20d	2^{-2}	2^{0}	2^{5}	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{2}
stock	2^{-1}	2^{15}	2^{20}	2^{5}	2^{12}	2^{6}	2^{9}	2^{12}	2^{6}
superconduc.	2^{-3}	2^{13}	2^{13}	2^{20}	2^{20}	2^{20}	2^{20}	2^{1}	2^{20}
tic	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{20}	2^{2}	2^{20}
treasury	2^{7}	2^{11}	2^{14}	2^{6}	2^{13}	2^{11}	2^{6}	2^{14}	2^{10}
wankara	2^{1}	2^{6}	2^{8}	2^{2}	2^{5}	2^{12}	2^{2}	2^{4}	2^{13}
wizmir	2^{1}	2^{4}	2^{4}	2^{3}	2^{4}	2^{7}	2^{3}	2^{2}	2^{4}
wq	2^{20}	2^{20}	2^{20}	2^{-2}	2^{11}	2^{9}	2^{0}	2^{14}	2^{10}

Tabela 33 – Hiperparâmetros C e γ obtidos após o 5-fold para o método FKDSN, nos experimentos da Seção 3.4.4.1.

Base	FKDSN		Base	FKDSN	
	C	γ		C	γ
ANACALT	2^{20}	2^{9}	laser	2^{20}	2^{8}
abalone	2^{16}	2^{6}	mortgage	2^{20}	2^{6}
ailerons	2^{20}	2^{11}	mv	2^{20}	2^{9}
airfoil	2^{20}	2^{7}	navalpropulsion	2^{20}	2^{6}
bank32nh	2^{14}	2^{10}	plastic	2^{15}	2^{7}
bank8fm	2^{20}	2^{8}	pole	2^{13}	2^{6}
california	2^{20}	2^{6}	puma32h	2^{2}	2^{6}
carbonnanotubes	2^{20}	2^{17}	puma8nh	2^{19}	2^{8}
combinedcycle	2^{20}	2^{5}	quake	2^{20}	2^{0}
compactiv	2^{20}	2^{10}	rf1	2^{20}	2^{8}
concrete	2^{20}	2^{5}	rf2	2^{20}	2^{16}
cpuact	2^{20}	2^{9}	scm1d	2^{20}	2^{16}
cpusmall	2^{20}	2^{9}	scm20d	2^{18}	2^{9}
delta_ail	2^{20}	2^{8}	stock	2^{20}	2^{6}
delta_elv	2^{20}	2^{6}	superconductivity	2^{20}	2^{12}
ele2	2^{20}	2^{6}	tic	2^{20}	2^{12}
elevators	2^{20}	2^{9}	treasury	2^{20}	2^{8}
friedman	2^{20}	2^{6}	wankara	2^{20}	2^{10}
house	2^{16}	2^{7}	wizmir	2^{20}	2^{9}
house8l	2^{20}	2^{8}	wq	2^{20}	2^{9}
kin8nm	2^{20}	2^{8}	-	-	-

Tabela 34 – Hiperparâmetros obtidos após o 5-fold para os métodos GOR-ELM e FDSN-GOR, nos experimentos da Seção 3.4.4.2.

Base/Método	GOR-ELM						FDSN-GOR					
	rfm		ae		svd		rfm		ae		svd	
	C	α	C	α	C	α	C	α	C	α	C	α
ANACALT	2^{3}	0,00	2^{-2}	0,00	2^{2}	0,00	2^{-9}	0,50	2^{-11}	0,00	2^{0}	0,00
abalone	2^{-2}	0,00	2^{-3}	0,00	2^{-9}	0,00	2^{-1}	0,50	2^{8}	0,75	2^{6}	0,75
ailerons	2^{5}	0,00	2^{-13}	0,00	2^{2}	0,00	2^{3}	0,75	2^{6}	0,00	2^{20}	0,50
airfoil	2^{2}	0,00	2^{-7}	0,00	2^{-20}	0,00	2^{0}	1,00	2^{-1}	0,25	2^{-2}	0,25
bank32nh	2^{-20}	0,00	2^{-8}	0,00	2^{-10}	0,00	2^{3}	0,25	2^{8}	0,50	2^{20}	0,25
bank8fm	2^{4}	0,00	2^{0}	0,00	2^{-4}	0,00	2^{2}	0,50	2^{20}	0,00	2^{3}	0,00
california	2^{-5}	0,00	2^{-5}	0,00	2^{-20}	0,00	2^{0}	0,25	2^{16}	1,00	2^{6}	0,75
carbonnano.	2^{-5}	0,00	2^{-5}	0,00	2^{-4}	0,00	2^{1}	0,50	2^{-2}	0,25	2^{-1}	0,25
combinedcy.	2^{-2}	0,00	2^{-6}	0,00	2^{-6}	0,00	2^{-5}	0,75	2^{4}	0,00	2^{4}	0,00
compactiv	2^{9}	0,00	2^{1}	0,00	2^{-3}	0,00	2^{0}	1,00	2^{-13}	0,00	2^{1}	0,00
concrete	2^{-2}	0,00	2^{-4}	0,00	2^{-20}	0,00	2^{-2}	1,00	2^{3}	0,50	2^{4}	0,50
cpuact	2^{7}	0,00	2^{1}	0,00	2^{-5}	0,00	2^{1}	0,75	2^{5}	0,00	2^{20}	0,25
cpusmall	2^{1}	0,00	2^{1}	0,00	2^{-5}	0,00	2^{1}	0,50	2^{5}	0,00	2^{4}	0,25
delta_ail	2^{-5}	0,00	2^{-8}	0,00	2^{-20}	0,00	2^{-2}	1,00	2^{11}	0,25	2^{20}	0,00
delta_elv	2^{-2}	0,00	2^{-7}	0,00	2^{-7}	0,00	2^{-5}	0,75	2^{11}	0,50	2^{6}	0,25
ele2	2^{1}	0,00	2^{-5}	0,00	2^{-8}	0,00	2^{-4}	0,75	2^{14}	0,00	2^{2}	0,00
elevators	2^{3}	0,00	2^{-6}	0,00	2^{-1}	0,00	2^{2}	1,00	2^{7}	0,00	2^{20}	0,00
friedman	2^{-2}	0,00	2^{-4}	0,00	2^{-20}	0,00	2^{-2}	1,00	2^{2}	0,75	2^{4}	1,00
house	2^{-20}	0,00	2^{-7}	0,00	2^{-20}	0,00	2^{3}	0,25	2^{11}	0,25	2^{-20}	0,50
house8l	2^{-5}	0,00	2^{-3}	0,00	2^{-20}	0,00	2^{2}	0,50	2^{7}	1,00	2^{20}	0,50
kin8nm	2^{-5}	0,00	2^{-6}	0,00	2^{-20}	0,00	2^{2}	0,50	2^{4}	1,00	2^{5}	1,00
laser	2^{-3}	0,00	2^{1}	0,00	2^{3}	0,00	2^{0}	0,25	2^{4}	0,00	2^{3}	0,00
mortgage	2^{-4}	0,00	2^{-6}	0,00	2^{-6}	0,00	2^{-4}	0,75	2^{20}	0,00	2^{20}	0,00
mv	2^{5}	0,00	2^{-6}	0,00	2^{-7}	0,00	2^{3}	0,50	2^{1}	0,00	2^{0}	0,00
navalprop.	2^{-5}	0,00	2^{-6}	0,00	2^{-20}	0,00	2^{-4}	0,75	2^{11}	1,00	2^{20}	1,00
plastic	2^{-20}	0,00	2^{-15}	0,00	2^{-10}	0,00	2^{-1}	0,50	2^{0}	0,50	2^{2}	1,00
pole	2^{5}	0,00	2^{2}	0,00	2^{4}	0,00	2^{2}	1,00	2^{-11}	0,00	2^{-16}	0,25
puma32h	2^{-20}	0,00	2^{-6}	0,00	2^{-20}	0,00	2^{2}	0,75	2^{4}	0,00	2^{4}	0,00
puma8nh	2^{-1}	0,00	2^{-4}	0,00	2^{-20}	0,00	2^{2}	0,25	2^{-3}	0,00	2^{3}	0,00
quake	2^{-20}	0,00	2^{-2}	0,00	2^{-20}	0,00	2^{-5}	0,00	2^{7}	0,75	2^{6}	0,50
rf1	2^{1}	0,00	2^{-1}	0,00	2^{2}	0,00	2^{-20}	1,00	2^{-16}	0,00	2^{9}	0,50
rf2	2^{0}	0,00	2^{-14}	0,00	2^{-2}	0,00	2^{-20}	0,00	2^{-19}	0,00	2^{10}	1,00
scm1d	2^{3}	0,00	2^{-3}	0,00	2^{2}	0,00	2^{-20}	0,00	2^{-14}	1,00	2^{7}	1,00
scm20d	2^{-1}	0,00	2^{-9}	0,00	2^{-7}	0,00	2^{-20}	0,00	2^{8}	1,00	2^{20}	1,00
stock	2^{5}	0,00	2^{-5}	0,00	2^{-8}	0,00	2^{-4}	0,50	2^{13}	0,25	2^{3}	0,50
superconduc.	2^{4}	0,00	2^{-7}	0,00	2^{-4}	0,00	2^{4}	0,25	2^{5}	0,25	2^{5}	0,00
tic	2^{-20}	0,00	2^{-10}	0,00	2^{-20}	0,00	2^{20}	0,00	2^{7}	0,00	2^{20}	0,25
treasury	2^{-4}	0,00	2^{-5}	0,00	2^{-6}	0,00	2^{-1}	0,25	2^{9}	0,00	2^{2}	0,00
wankara	2^{3}	0,00	2^{-9}	0,00	2^{-4}	0,00	2^{-1}	1,00	2^{-3}	0,00	2^{2}	0,00
wizmir	2^{3}	0,00	2^{-1}	0,00	2^{-6}	0,00	2^{0}	0,50	2^{1}	0,00	2^{2}	0,00
wq	2^{-9}	0,00	2^{-5}	0,00	2^{-20}	0,00	2^{-2}	0,00	2^{7}	1,00	2^{6}	0,00

Tabela 35 – Funções de ativação obtidas após o 5-fold nos experimentos da Seção 3.4.4.2. As funções sigmoide, *hard-limit*, gaussiana e multiquádrica são representadas pelas letras 's', 'h', 'r' e 'm', respectivamente.

Base/Método	R-ELM			GOR-ELM			DSN			FDSN			FDSN-GOR		
	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd	rfm	ae	svd
ANACALT	h	m	h	h	h	h	m	m	s	s	m	m	s	m	m
abalone	s	s	s	s	r	r	m	s	r	s	s	s	s	s	s
ailerons	s	m	r	s	r	r	s	m	s	s	m	s	s	m	s
airfoil	s	r	r	m	r	r	s	m	m	m	m	m	m	m	m
bank32nh	m	m	m	m	r	m	s	m	s	s	m	s	s	m	s
bank8fm	s	r	m	s	r	r	s	s	s	s	m	s	s	m	s
california	s	m	m	s	r	r	s	m	r	s	m	s	s	m	s
carbonnanotubes	s	r	r	s	r	r	s	m	s	s	s	s	s	s	s
combinedcycle	h	r	h	h	r	h	s	s	s	s	s	s	s	s	s
compactiv	m	m	m	m	r	m	m	m	s	s	m	s	m	m	s
concrete	s	m	r	s	r	r	s	m	s	s	s	s	s	s	m
cpuact	m	m	m	m	r	m	m	m	s	s	m	s	m	m	s
cpusmall	s	r	r	s	r	r	s	m	s	m	m	s	m	m	s
delta_ail	s	s	r	s	r	r	s	m	s	s	s	s	s	s	s
delta_elv	s	m	m	s	r	r	s	s	m	s	s	s	s	s	s
ele2	m	r	r	m	r	r	m	m	r	m	m	m	m	m	m
elevators	s	r	r	s	r	r	s	s	s	s	s	s	s	s	s
friedman	s	r	r	s	r	r	s	m	s	s	s	s	s	s	s
house	s	s	m	m	r	r	m	m	s	m	m	m	m	m	m
house8l	s	s	m	s	r	r	m	m	m	m	m	s	m	m	s
kin8nm	m	r	r	s	r	r	s	s	r	s	m	s	s	s	s
laser	s	r	r	s	h	h	m	m	m	s	s	s	m	m	m
mortgage	s	m	r	s	r	r	s	s	r	s	s	s	s	m	m
mv	m	r	r	m	r	r	s	s	s	s	m	s	s	m	s
navalpropulsion	m	r	r	m	r	r	s	m	m	s	s	s	s	s	m
plastic	s	r	r	s	m	r	s	s	s	s	s	s	s	s	s
pole	h	r	h	h	r	h	s	r	r	s	m	m	s	r	m
puma32h	s	r	s	s	r	r	s	s	s	s	m	s	s	m	s
puma8nh	s	r	m	s	r	r	s	s	s	s	s	s	s	s	s
quake	h	h	h	h	h	h	m	m	m	m	m	m	m	m	m
rf1	s	r	h	s	r	h	s	m	s	s	m	s	s	m	s
rf2	s	s	s	s	r	s	s	m	s	s	m	s	s	m	s
scm1d	s	m	r	s	m	r	s	m	s	s	s	s	s	s	s
scm20d	s	r	r	s	r	r	s	m	s	s	s	s	s	s	s
stock	r	r	m	r	r	r	s	m	r	s	s	m	s	s	s
superconductivity	s	m	s	s	m	m	s	m	s	s	m	s	s	m	s
tic	m	s	s	r	h	s	s	m	s	s	m	m	m	m	m
treasury	s	r	r	s	r	r	s	m	s	s	m	s	s	m	m
wankara	s	m	m	s	r	m	s	m	s	s	m	s	s	m	s
wizmir	s	m	r	s	m	m	s	m	s	s	m	s	s	s	s
wq	m	s	r	m	h	h	m	m	m	m	m	m	m	m	m

APÊNDICE C – Tabelas de Resultados - Métodos Offline - Estudo de Caso

Tabela 36 – Hiperparâmetros obtidos no estudo de caso feito no Capítulo 4. O hiperparâmetro função de ativação não se aplica ao método FKDSN, onde as funções sigmoide, *hard-limit*, gaussiana e multiquádrica são representadas pelas letras 's', 'h', 'r' e 'm', respectivamente. Já o hiperparâmetro γ se aplica apenas ao método FKDSN, pois ele é um hiperparâmetro do *kernel* deste método. L se refere ao número de neurônios por módulo e S é o número máximo de módulos.

Base	Método	Hiperparâmetros				
		C	L	S	Fun. Ativ.	γ (kernel)
Solar	FDSN_{rfm}	2^2	100	10	s	-
	FDSN_{ae}	2^{15}	50	9	s	-
	FDSN_{svd}	2^4	100	10	r	-
	FKDSN	2^{18}	100	8	-	2^4
Wind	FDSN_{rfm}	2^5	100	1	s	-
	FDSN_{ae}	2^{18}	100	1	s	-
	FDSN_{svd}	2^{11}	100	1	s	-
	FKDSN	2^{10}	100	1	-	2^6

APÊNDICE D – Tabelas de Resultados - Métodos Online - Bases de dados *Online* até 8000 amostras

Tabela 37 – Valores de RMSE obtidas nos experimentos da Seção 5.2.3.1, no formato "média (desvio padrão)".

Base/Método	OSFDSN-v			OSFDSN-v-i			DOER	kNN	Learn^{++} NSE	OEOA	OWE
	-	ae	svd	-	ae	svd					
FriedmanGRA	**0,63**	1,15	0,66	**0,63**	1,81	0,66	0,64	0,69	0,71	0,64	0,65
	(0,02)	(1,65)	(0,14)	**(0,03)**	(2,11)	(0,11)	(0,01)	(0,00)	(0,00)	(0,00)	(0,00)
FriedmanGnRG	0,65	1,17	0,70	0,66	1,82	0,69	**0,64**	0,71	0,72	**0,64**	**0,64**
	(0,02)	(1,65)	(0,13)	(0,03)	(2,10)	(0,11)	**(0,00)**	(0,00)	(0,00)	**(0,00)**	**(0,00)**
FriedmanLA	0,59	0,62	0,56	0,59	0,62	**0,55**	0,62	0,59	0,74	0,62	0,63
	(0,02)	(0,07)	(0,01)	(0,02)	(0,07)	**(0,01)**	(0,01)	(0,00)	(0,00)	(0,00)	(0,00)
Hyperplane	0,64	0,68	0,65	0,63	0,67	0,64	**0,45**	0,84	0,53	**0,45**	**0,45**
	(0,05)	(0,02)	(0,02)	(0,03)	(0,03)	(0,01)	**(0,00)**	(0,00)	(0,00)	**(0,00)**	**(0,00)**
abalone	0,69	0,73	0,69	**0,68**	0,77	0,69	0,77	0,76	0,78	0,77	0,76
	(0,01)	(0,04)	(0,01)	**(0,01)**	(0,12)	(0,04)	(0,00)	(0,00)	(0,01)	(0,00)	(0,00)
debutanizer	1,08	2,76	1,01	1,11	4,17	1,06	**0,78**	1,11	1,87	**0,78**	1,30
	(0,10)	(4,14)	(0,03)	(0,10)	(6,72)	(0,04)	**(0,00)**	(0,00)	(0,00)	**(0,00)**	(0,00)
quake	1,34	1,25	1,03	1,35	1,39	1,12	**1,01**	1,13	**1,01**	**1,01**	1,04
	(0,15)	(0,23)	(0,04)	(0,15)	(0,41)	(0,14)	**(0,00)**	(0,00)	**(0,00)**	**(0,00)**	(0,00)
wind	0,54	0,73	**0,52**	0,54	0,61	**0,52**	0,58	0,59	0,64	0,59	0,60
	(0,01)	(0,47)	**(0,01)**	(0,01)	(0,13)	**(0,02)**	(0,00)	(0,00)	(0,01)	(0,00)	(0,00)
winequality	0,87	0,87	**0,86**	**0,86**	0,92	**0,86**	0,91	0,93	1,13	0,91	1,11
	(0,01)	(0,01)	**(0,01)**	**(0,01)**	(0,10)	**(0,03)**	(0,01)	(0,00)	(0,02)	(0,00)	(0,01)

Tabela 38 – Tempos de treinamento dos testes da Seção 5.2.3.1, no formato "média (desvio padrão)".

Base/Método	OSFDSN-v			OSFDSN-v-i			DOER	kNN	Learn^{++} NSE	OEOA	OWE
	-	ae	svd	-	ae	svd					
FriedmanGRA	2,80	2,86	2,78	0,23	0,27	0,30	1818,63	**0,02**	9,71	1828,02	178,39
	(0,13)	(0,15)	(0,13)	(0,02)	(0,02)	(0,03)	(14,96)	**(0,02)**	(0,03)	(14,55)	(0,31)
FriedmanGnRG	2,77	2,78	2,68	0,23	0,25	0,28	1791,36	**0,02**	9,63	1808,83	171,02
	(0,13)	(0,07)	(0,04)	(0,02)	(0,02)	(0,03)	(15,31)	**(0,01)**	(0,03)	(7,26)	(1,12)
FriedmanLA	2,64	2,67	2,71	0,25	0,27	0,29	1794,68	**0,01**	9,28	1801,91	177,72
	(0,03)	(0,05)	(0,04)	(0,04)	(0,03)	(0,02)	(17,72)	**(0,00)**	(0,01)	(12,14)	(0,16)
Hyperplane	2,73	2,67	2,66	0,28	0,29	0,30	1778,20	**0,02**	9,67	1782,72	168,38
	(0,04)	(0,05)	(0,06)	(0,02)	(0,02)	(0,02)	(7,65)	**(0,00)**	(0,01)	(18,53)	(0,20)
abalone	6,42	6,14	6,33	0,57	0,65	0,70	3999,19	**0,05**	16,21	4007,48	298,70
	(0,18)	(0,16)	(0,17)	(0,17)	(0,22)	(0,26)	(10,11)	**(0,01)**	(0,03)	(13,33)	(2,07)
debutanizer	3,62	3,66	3,63	0,32	0,28	0,35	2121,06	**0,02**	8,07	2134,57	151,62
	(0,11)	(0,12)	(0,08)	(0,06)	(0,02)	(0,06)	(7,08)	**(0,00)**	(0,01)	(7,72)	(0,10)
quake	3,46	3,34	3,40	0,23	0,22	0,23	1995,53	**0,01**	3,73	2001,64	64,45
	(0,09)	(0,05)	(0,13)	(0,02)	(0,00)	(0,01)	(9,10)	**(0,00)**	(0,00)	(14,53)	(0,61)

Tabela 38 – *Continuação*

Base/Método	OSFDSN-v			OSFDSN-v-i			DOER	kNN	Learn^{++} NSE	OEOA	OWE
	-	ae	svd	-	ae	svd					
wind	10,23	10,06	9,89	0,82	0,95	0,99	6261,87	**0,11**	44,34	6274,70	818,75
	(0,27)	(0,65)	(0,25)	(0,26)	(0,34)	(0,31)	(28,19)	**(0,01)**	(0,04)	(27,62)	(0,79)
winequality	7,61	7,62	7,64	0,58	0,60	0,70	4778,26	**0,07**	25,16	4787,07	473,89
	(0,18)	(0,16)	(0,09)	(0,11)	(0,12)	(0,16)	(18,97)	**(0,01)**	(0,02)	(16,14)	(1,08)

Tabela 39 – Quantidade de kilobytes gastos em cada modelo dos experimentos da Seção 5.2.3.1.

Base/Método	OSFDSN-v			OSFDSN-v-i			DOER	kNN	Learn^{++} NSE	OEOA	OWE
	-	ae	svd	-	ae	svd					
FriedmanGRA	29	34	31	**10**	13	16	71	187	39	69	28
FriedmanGnRG	28	34	31	**10**	13	18	69	187	40	71	28
FriedmanLA	19	22	35	**14**	18	21	72	187	39	73	28
Hyperplane	23	28	28	**21**	23	26	78	187	38	79	29
abalone	94	99	112	30	37	44	55	325	53	55	**25**
debutanizer	98	118	112	24	**15**	26	65	167	44	66	26
quake	133	124	138	**5**	**5**	6	46	84	41	46	30
wind	109	132	120	**28**	35	39	66	820	61	66	**28**
winequality	141	148	155	**23**	26	32	56	495	67	55	26

Tabela 40 – Número médio de módulos dos métodos empilhados dos testes da Seção 5.2.3.1 e número de vizinhos mais próximos do método kNN.

Base/Método	OSFDSN-v			OSFDSN-v-i			kNN
	-	ae	svd	-	ae	svd	
FriedmanGRA	6,90	8,30	7,60	2,00	2,90	3,70	3
FriedmanGnRG	6,70	8,30	7,50	2,10	2,70	4,20	5
FriedmanLA	4,40	5,10	8,40	3,10	4,10	5,00	7
Hyperplane	5,50	6,60	6,80	4,90	5,40	6,20	9
abalone	24,00	25,40	28,60	7,30	9,10	10,90	3
debutanizer	25,10	30,20	28,70	5,70	3,40	6,40	3
quake	34,40	32,00	35,60	1,00	1,00	1,20	11
wind	27,70	33,70	30,50	6,60	8,40	9,40	3
winequality	36,20	38,10	39,80	5,30	6,10	7,70	3

APÊNDICE E – Tabelas de Resultados - Métodos Online - Bases de dados *Online*

Tabela 41 – Valores de RMSE obtidas nos experimentos da Seção 5.2.3.2, no formato "média (desvio padrão)".

Base/Método	OSFDSN-v			OSFDSN-v-i			kNN	Learn^{++}	OWE
	-	ae	svd	-	ae	svd		NSE	
FriedmanGRA	1,32 (1,71)	**0,42** **(0,04)**	0,58 (0,25)	1,35 (1,71)	6,71 (18,17)	0,63 (0,18)	**0,42** **(0,00)**	0,71 (0,00)	0,65 (0,00)
FriedmanGnRG	**0,39** **(0,02)**	0,41 (0,03)	0,40 (0,02)	0,41 (0,05)	0,72 (0,94)	0,41 (0,05)	0,44 (0,00)	0,72 (0,00)	0,64 (0,00)
FriedmanLA	0,37 (0,02)	0,92 (1,46)	**0,35** **(0,01)**	0,41 (0,07)	0,42 (0,09)	0,40 (0,07)	0,39 (0,00)	0,74 (0,00)	0,63 (0,00)
Hyperplane	**0,40** **(0,03)**	0,46 (0,10)	0,42 (0,02)	0,42 (0,07)	0,62 (0,56)	0,43 (0,05)	0,52 (0,00)	0,53 (0,00)	0,45 (0,00)
SRU1	0,79 (0,27)	0,79 (0,10)	**0,66** **(0,02)**	0,79 (0,27)	1,52 (1,15)	0,67 (0,03)	0,69 (0,00)	1,21 (0,00)	1,11 (0,00)
SRU2	0,72 (0,21)	0,74 (0,13)	**0,65** **(0,03)**	0,72 (0,21)	1,20 (1,11)	0,66 (0,03)	0,71 (0,00)	1,20 (0,00)	1,08 (0,00)
abalone	0,43 (0,05)	**0,41** **(0,01)**	0,42 (0,01)	0,44 (0,05)	0,56 (0,17)	0,45 (0,03)	0,45 (0,00)	0,78 (0,01)	0,76 (0,00)
bank8fm	0,25 (0,00)	0,25 (0,01)	**0,24** **(0,00)**	0,25 (0,00)	0,25 (0,01)	**0,24** **(0,00)**	0,33 (0,00)	0,43 (0,00)	0,36 (0,00)
debutanizer	0,81 (0,20)	1,00 (0,42)	**0,76** **(0,14)**	0,84 (0,19)	2,49 (5,30)	0,80 (0,14)	0,81 (0,00)	1,87 (0,00)	1,30 (0,00)
elevators	0,52 (0,25)	0,95 (1,01)	0,50 (0,18)	0,55 (0,25)	1,21 (0,91)	0,53 (0,18)	**0,43** **(0,00)**	0,88 (0,01)	1,71 (0,62)
house16h	0,84 (0,88)	0,92 (0,87)	0,47 (0,02)	0,85 (0,88)	2,17 (4,16)	0,51 (0,04)	**0,44** **(0,00)**	1,02 (0,01)	1,05 (0,00)
kin8nm	0,55 (0,28)	0,48 (0,08)	0,45 (0,02)	0,58 (0,27)	1,15 (1,06)	0,51 (0,05)	**0,35** **(0,00)**	0,85 (0,00)	0,85 (0,00)
mv	0,21 (0,02)	0,20 (0,03)	**0,15** **(0,01)**	0,24 (0,07)	0,20 (0,03)	0,16 (0,05)	0,18 (0,00)	0,60 (0,00)	0,53 (0,00)
pol	0,77 (0,33)	0,73 (0,22)	0,75 (0,21)	0,85 (0,32)	1,04 (0,60)	0,83 (0,20)	**0,51** **(0,00)**	0,99 (0,02)	0,99 (0,00)
puma32h	0,69 (0,34)	1,10 (1,48)	0,63 (0,32)	0,69 (0,34)	1,15 (0,74)	0,64 (0,29)	**0,55** **(0,00)**	1,12 (0,00)	1,11 (0,00)
quake	0,73 (0,14)	0,73 (0,16)	**0,63** **(0,02)**	0,74 (0,14)	0,93 (0,36)	0,65 (0,03)	0,65 (0,00)	1,01 (0,00)	1,04 (0,00)
wind	0,33 (0,01)	0,35 (0,05)	**0,32** **(0,00)**	0,36 (0,03)	0,37 (0,07)	0,35 (0,03)	0,37 (0,00)	0,64 (0,01)	0,60 (0,00)
winequality	0,61 (0,01)	0,61 (0,01)	**0,60** **(0,01)**	0,61 (0,01)	0,62 (0,03)	0,61 (0,02)	0,67 (0,00)	1,13 (0,02)	1,11 (0,01)

Tabela 42 – Tempos de treinamento, em segundos, dos experimentos da Seção 5.2.3.2, no formato "média (desvio padrão)".

Base/Método	OSFDSN-v			OSFDSN-v-i			kNN	Learn^{++}	OWE
	-	ae	svd	-	ae	svd		NSE	
FriedmanGRA	10,46	10,37	10,14	0,45	0,48	0,46	**0,03**	9,71	178,39
	(0,10)	(0,07)	(0,07)	(0,02)	(0,03)	(0,03)	**(0,02)**	(0,03)	(0,31)
FriedmanGnRG	10,34	10,35	10,41	0,48	0,50	0,51	**0,02**	9,63	171,02
	(0,09)	(0,07)	(0,11)	(0,04)	(0,03)	(0,04)	**(0,00)**	(0,03)	(1,12)
FriedmanLA	10,23	10,20	10,21	0,49	0,50	0,54	**0,02**	9,28	177,72
	(0,06)	(0,04)	(0,04)	(0,04)	(0,03)	(0,06)	**(0,00)**	(0,01)	(0,16)
Hyperplane	10,41	10,36	10,47	0,51	0,54	0,57	**0,03**	9,67	168,38
	(0,13)	(0,11)	(0,19)	(0,05)	(0,05)	(0,05)	**(0,00)**	(0,01)	(0,20)
SRU1	64,25	64,61	62,83	1,43	1,37	1,42	**0,21**	25,82	464,63
	(0,71)	(1,38)	(1,00)	(0,30)	(0,33)	(0,38)	**(0,01)**	(0,03)	(0,25)
SRU2	61,92	62,84	62,75	1,47	1,47	1,60	**0,20**	24,84	446,80
	(0,73)	(0,70)	(1,39)	(0,45)	(0,34)	(0,59)	**(0,01)**	(0,03)	(0,35)
abalone	21,90	22,24	22,10	0,62	0,69	0,70	**0,06**	16,21	298,70
	(0,22)	(0,14)	(0,25)	(0,04)	(0,03)	(0,06)	**(0,01)**	(0,03)	(2,07)
bank8fm	44,44	43,26	44,27	2,03	2,10	2,41	**0,15**	31,81	557,97
	(0,69)	(0,44)	(0,36)	(0,60)	(0,62)	(0,58)	**(0,00)**	(0,04)	(4,89)
debutanizer	12,85	12,77	12,82	0,54	0,56	0,58	**0,03**	8,07	151,62
	(0,07)	(0,09)	(0,08)	(0,08)	(0,05)	(0,06)	**(0,00)**	(0,01)	(0,10)
elevators	102,93	108,36	103,60	1,85	3,41	2,46	**0,69**	136,19	2567,88
	(2,28)	(4,32)	(2,57)	(1,08)	(3,52)	(1,79)	**(0,01)**	(0,13)	(4,28)
house16h	155,29	156,34	160,60	2,10	3,39	2,85	**1,34**	168,84	3186,59
	(3,43)	(5,97)	(4,51)	(0,72)	(1,92)	(1,90)	**(0,01)**	(0,10)	(14,76)
kin8nm	45,02	44,72	45,65	0,90	1,00	1,04	**0,15**	31,45	581,09
	(0,35)	(0,56)	(0,33)	(0,07)	(0,06)	(0,09)	**(0,00)**	(0,04)	(0,45)
mv	260,71	260,73	265,27	19,47	21,69	34,29	**3,30**	195,29	3461,62
	(6,01)	(4,96)	(6,90)	(7,17)	(7,55)	(9,70)	**(0,05)**	(0,20)	(38,97)
pol	90,39	86,61	90,04	1,47	5,63	1,76	**1,01**	334,84	6402,71
	(1,36)	(1,32)	(1,63)	(0,15)	(6,33)	(0,48)	**(0,02)**	(6,20)	(38,64)
puma32h	46,60	47,03	47,01	0,84	1,16	0,86	**0,26**	118,85	2324,61
	(0,36)	(0,58)	(0,71)	(0,05)	(0,25)	(0,04)	**(0,01)**	(0,09)	(36,69)
quake	11,43	11,39	11,55	0,45	0,48	0,46	**0,02**	3,73	64,45
	(0,07)	(0,08)	(0,08)	(0,01)	(0,05)	(0,02)	**(0,00)**	(0,00)	(0,61)
wind	35,51	35,83	36,01	1,08	1,27	1,31	**0,13**	44,34	818,75
	(0,60)	(0,21)	(0,27)	(0,25)	(0,29)	(0,36)	**(0,01)**	(0,04)	(0,79)
winequality	27,09	27,00	26,30	0,84	0,87	0,88	**0,07**	25,16	473,89
	(0,21)	(0,13)	(0,14)	(0,15)	(0,17)	(0,20)	**(0,00)**	(0,02)	(1,08)

Tabela 43 – Quantidade de kilobytes usados em cada modelo dos experimentos da Seção 5.2.3.2.

Base/Método	OSFDSN-v			OSFDSN-v-i			kNN	Learn^{++}	OWE
	-	ae	svd	-	ae	svd		NSE	
FriedmanGRA	32	33	35	**7**	12	9	187	39	28
FriedmanGnRG	41	40	53	**14**	16	19	187	40	28
FriedmanLA	34	42	52	**14**	18	24	187	39	28
Hyperplane	33	30	41	**21**	24	28	187	38	29
SRU1	560	585	581	**20**	**20**	23	551	97	24
SRU2	516	521	532	24	**23**	29	551	83	24
abalone	126	130	130	**9**	14	13	325	53	25
bank8fm	179	171	219	76	76	95	639	39	**28**
debutanizer	103	117	103	21	**20**	23	167	44	26
elevators	573	719	603	**21**	66	35	2590	125	37

Tabela 43 – *Continuação*

Base/Método	OSFDSN-v			OSFDSN-v-i			kNN	Learn^{++} NSE	OWE
	-	ae	svd	-	ae	svd			
house16h	837	863	928	**16**	49	33	3203	179	31
kin8nm	201	203	225	**8**	12	12	639	83	26
mv	626	626	679	208	237	402	3821	65	**31**
pol	442	404	481	**15**	160	25	5859	146	36
puma32h	321	350	356	**10**	32	**10**	2173	118	43
quake	134	125	166	**6**	**6**	**6**	84	41	30
wind	149	171	173	30	43	47	820	61	**28**
winequality	193	207	203	**24**	29	30	495	67	26

Tabela 44 – Número médio de módulos dos métodos empilhados dos testes da Seção 5.2.3.2 e número de vizinhos mais próximos do método kNN.

Base/Método	OSFDSN-v			OSFDSN-v-i			kNN
	-	ae	svd	-	ae	svd	
FriedmanGRA	7,50	7,90	8,40	1,00	2,30	1,70	3
FriedmanGnRG	9,90	9,60	13,10	3,00	3,40	4,30	3
FriedmanLA	8,20	10,30	12,80	3,00	3,90	5,60	3
Hyperplane	7,90	7,10	10,00	4,70	5,40	6,40	3
SRU1	145,60	152,20	151,10	4,70	4,70	5,30	5
SRU2	134,10	135,40	138,30	5,70	5,30	7,00	3
abalone	32,30	33,20	33,30	1,60	3,00	2,60	3
bank8fm	46,00	44,00	56,40	19,10	19,20	24,20	3
debutanizer	26,10	30,00	26,20	4,70	4,60	5,40	3
elevators	148,30	186,40	156,20	4,30	16,10	8,10	5
house16h	217,40	224,30	241,10	3,20	11,80	7,50	5
kin8nm	51,70	52,40	57,90	1,40	2,50	2,50	3
mv	162,60	162,70	176,40	53,60	61,20	104,00	3
pol	113,10	103,00	123,30	1,70	39,50	4,20	3
puma32h	82,10	89,70	91,10	1,00	6,70	1,00	3
quake	34,40	32,20	42,70	1,00	1,10	1,20	5
wind	38,00	43,70	44,10	6,80	10,30	11,20	3
winequality	49,60	53,30	52,20	5,50	6,70	7,00	3