Metodologias de Redes Neurais Artificiais Aplicadas em Diferentes Bancos de Dados

Gabriel Saraiva Espeschit
Escola de Engenharia da UFMG
Belo Horizonte, Brasil
gabrielespeschit@ufmg.br

***Resumo*—Nesse artigo investiga-se o desempenho de métodos de redes neurais artificiais para classificação de dados em bancos de dados diferentes. Os métodos de redes neurais usados foram: Perceptron Simples, Máquinas de Aprendizado Extremo, Redes Neurais com Base em Funções Radiais e o Perceptron de Múltiplas Camadas. Selecionou-se 3 bancos de dados diferentes para verificar o desempenho de cada método em cada banco.**

***Palavras Chave*—redes neurais artificiais, perceptron, máquinas de aprendizado extremo, base em funções radiais**

I. Introdução

Redes Neurais Artificiais (RNA) são uma das mais estudas formas para a resolução de problemas de reconhecimento de padrões. Problemas de reconhecimento de padrões são problemas em que, para uma série de dados de entrada, deve-se associar esses dados a uma classe conhecida. Esse tipo de problemas também são conhecidos como problemas de classificação.

Nesse estudo, pretende-se investigar o desempenho de métodos de RNAs capazes de resolver o problema de classificação em três bancos de dados diferentes. Os métodos escolhidos foram: Perceptron Simples, Máquinas de Aprendizado Extremo (ELM), Redes Neurais com Base em Funções Radiais (RBF) e o Perceptron de Múltiplas Camadas (PMC). Os bancos a serem investigados forma obtidos da base de dados da Universidade da Califórnia. As bases de dados foram escolhidos a mão, sem nenhum padrão de escolha específico, desde que se tratem de problemas de classificação.

A seguir, será feito uma breve revisão a respeito dos métodos implementados e das base de dados utilizadas.

II. Revisão da Literatura

Nessa seção iremos fazer uma breve explicação a respeito do funcionamento de cada metodologia adotada nesse estudo. Em seguida, iremos apresentar os bancos de dados selecionados e suas características.

A. Perceptron Simples

Classificadores lineares são responsáveis por fazer a divisão do espaço do problema por meio de retas, planos ou hiperplanos. O Perceptron Simples é caracterizado por ser um classificador que aplica uma não-linearidade em cima da função de ativação. Essa função é dada por $f(x \cdot w) = \hat{y}$, em que $\mathbf{x}$ é o vetor contendo os atributos de entrada e $\mathbf{w}$ é o vetor contendo os parâmetros e $\hat{y}$ é a saída (classe) do modelo para aquela determinada entrada.

Sendo assim, temos que a função de ativação $f(\cdot)$ mapeia o produto escalar $x \cdot w = \sum x_i w_i$ à variável $\hat{y}$. Em termos matriciais, temos que $f(\mathbf{XW}) = \hat{\mathbf{Y}}$. A função de ativação pode assumir várias formas, como a função degrau, sigmoidal e tangente hiperbólica. Nesse estudo utilizaremos somente a função sigmoidal ($f(u) = \frac{1}{1+e^{-\beta u}}$) por ser uma aproximação na forma continua da função degrau.

O Perceptron Simples é treinado por correções de erros, isto é, o erro entre a saída prevista e o valor real ajusta os pesos com o objetivo de a correção. Uma das formas que temos pra fazer isso é utilizando a técnica do gradiente descendente. Sendo assim, o ajuste dos pesos será feito por meio de uma minimização do erro quadrático médio, dado por (1), sobre todo o conjunto de treinamento.

$$J = \frac{1}{2}\sum_j (y_j - f(u_j))^2 \tag{1}$$

em que $f(u_j) = \hat{y_j}$. A partir de (1), podemos fazer uma derivação para encontrar o gradiente desse problema, isto é, a direção que, derivando em relação à w_i e u_j o valor de J aumenta. Se caminharmos na direção oposta do gradiente, estaremos minimizando a função do erro quadrático. A derivação da função sigmoidal é $(f(u_j)(1 - f(u_j))$. A Equação (2), nos dá a forma para atualização dos pesos usando a função sigmoidal.

$$w_i(t+1) = w_i(t) + \eta e(t) f(u_j)(1 - f(u_j))x_j(t) \tag{2}$$

em que $w(t)$, $e(t)$ e η são, respectivamente, o vetor de pesos na iteração t, o erro na iteração t e a taxa de aprendizado (normalmente na faixa de 10^{-3} à 10^{-4}).

B. Máquinas de Aprendizado Extremo

ELMs (*Extreme Learning Machines*) são redes neurais artificiais que fazem o mapeamento da variável de entrada por meio de uma função de mapeamento $\phi_h(\mathbf{x_i}, \mathbf{Z})$, tal que não haja nenhum tipo de restrição sob esse mapeamento e a matriz $\mathbf{Z}$ seja selecionada de forma aleatória. O número de funções (neurônios) $h(\mathbf{x}, \mathbf{z_i})$ deverá ser suficiente para garantir a separabilidade no espaço da camada intermediaria.

Para treinar uma ELM, o primeiro passo é criar uma matriz aleatória de pesos $\mathbf{Z}$. Em seguida, deve-se encontrar a matriz de mapeamento $\mathbf{H}$ por meio da operação $\mathbf{H} = \phi_h(\mathbf{xZ})$, em

que ϕ_h é usualmente uma função sigmoidal como a tangente hiperbólica. Sendo assim, o problema passa a ser encontrar uma matriz de pesos $\mathbf{W}$, de dimensões pXm, que satisfaça (3).

$$\phi_o(\mathbf{HW}) = \mathbf{Y} \tag{3}$$

Para problemas de classificação, podemos fazer uma linearização de (3) e reduzir a equação à $\mathbf{HW} = \mathbf{Y}$, de tal modo que podemos obter a solução por meio da pseudo-inversa de $\mathbf{H}$, $\mathbf{H}^+$, como pode ser visto em (4).

$$\mathbf{W} = \mathbf{H}^+\mathbf{Y} \tag{4}$$

C. Redes com Base em Funções Radiais

RBFs (*Radial Basis Functions*) são caracterizadas pelo uso de funções radiais nos neurônios da sua única camada intermediária. A Equação (5) descreve formalmente uma rede RBF.

$$f(x,\theta) = \sum_{i=1}^{p} w_i h_i(x, z_i) + w_0 \tag{5}$$

em que $\mathbf{w} = [w_1, w_2, ..., w_p]$ é o vetor de parâmetros do neurônio de saída da rede; z_i é o vetor que contem todos os parâmetros da função de ativação radial do neurônio i da camada intermediária; e $\theta = [w, z_1, z_2, ..., z_p]$ é o vetor que contem a concatenação de todos os vetores de parâmetros da rede.

No caso desse estudo, utilizaremos funções radiais normais. Sendo assim, para o treinamento do modelo, teremos que encontrar a matriz de covariância Σ_i e o vetor de médias μ_i. A partir disso, poderemos calcular a expressão $h(x, z_i)$, está representada em (6).

$$h(x, z_i) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_i|}} e^{(-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i))} \tag{6}$$

Uma vez encontrada as matriz $\mathbf{H}$, representada em (7), podemos obter a matriz de pesos $\mathbf{w}$ por meio de (8).

$$H = \begin{bmatrix} h_1(x_1, z_1) & h_2(x_1, z_2) & \cdots & h_h(x_1, z_p) \\ h_1(x_2, z_1) & h_2(x_2, z_2) & \cdots & h_h(x_2, z_p) \\ \vdots & \vdots & \cdots & \vdots \\ h_1(x_N, z_1) & h_2(x_N, z_2) & \cdots & h_h(x_N, z_p) \end{bmatrix} \tag{7}$$

$$\mathbf{w} = \mathbf{H}^+\mathbf{Y} \tag{8}$$

em que $\mathbf{H}^+$ é a pseudo-inversa de $\mathbf{H}$.

D. Perceptron de Múltiplas Camadas

Redes MLP são muito similares às redes ELMs. Elas se diferem, no entanto, quanto ao seu treinamento, particularmente o *backpropagation*. Optou-se por não entrar em detalhes a respeito do funcionamento de uma MLP, visto que essa explicação é muito extensa e que não será implementada a metodologia de uma MLP do princípio, optando pelo uso de uma biblioteca pronta para tal. No entanto é importante entender o passo-a-passo de uma MLP.

Uma MLP consiste de uma estrutura de neurônios de duas ou mais camadas. MLPs são aproximadoras universais de funções, sendo assim conseguem resolver um espectro amplo de problemas. O treinamento de uma MLP se da em duas etapas: a etapa de *feed-foward* e a etapa de *backpropagation*. Na primeira etapa, é feito o calculo da saída. Em seguida é calculado o erro da saída em relação ao valor esperado por meio de uma função de erro, usualmente o erro quadrático. Em seguida, é feito a atualização dos pesos da rede, por meio do *backpropagation*. Para tal, utiliza-se diversos algoritmos, o mais comum sendo o gradiente descendente. Esse processo é repetido iterativamente até o erro quadrático médio ficar abaixo de uma faixa de tolerância ou um número arbitrário de iterações forem atingidas.

E. Bancos de Dados

Foram escolhidos três bancos de dados do repositório da Universidade da Califórnia em Irvine para fazer a comparação entre os métodos. As base de dados escolhidas foram: *Adult, Tic-Tac-Toe Endgame* e *Mammographic Masses*. Nessa seção fará-se uma breve explicação de cada base.

A *Adult* é uma base de dados para resolver um problema de classificação em que se deve dizer, com base em fatores socio-econômicos, se uma pessoa ganha mais de 50 mil dólares por ano ou não. Sendo assim, entre os 14 atributos dessa base de dados, estão: idade, educação, estado-civil e país de residência.

A *Tic-Tac-Toe Endgame* também é uma base de dados de classificação. Nessa base de dados tem-se todos as configurações do final de um jogo da velha em que 'X' começou primeiro. Sendo assim, a tarefa é classificar se há ou não uma possibilidade de vitória para 'X'. São 9 atributos no total, em que, cada um representa um quadrado do tabuleiro do jogo da velha.

Por fim, a *Mammographic Masses* é uma base de dados que contem 5 atributos a respeito de pacientes e suas respectivas mamografias e tenta-se prever se a massa vista na tomografia é benigna ou maligna. Os atributos dados são: avaliação BI-RADS (uma avaliação feita pelo médico a respeito da massa), a idade da paciente, o formato da massa, a margem da massa e a densidade da massa.

III. Metodologia

Para cada método explorado nesse estudo, foi desenvolvido um código base que depois foi adaptado para cada banco de dados. Nessa seção iremos falar o que foi feito para cada método, bem como explicar as modificações feitas em cada base de dados.

A. Pré Processamento das Bases de Dados

Para as bases de dados, foi necessário aplicar um algoritmo de fatoração, isto é, transformar instancias do tipo "string", em instâncias numéricas. Especialmente nos bancos *Adult* e no *Tic-Tac-Toe Endgame* utiliza-se de strings para fazer a designação dos atributos. Nesses casos, foi utilizado a

função *factorize* da biblioteca *Pandas* para as transformar em números, habilitando-as para poderem ser passadas para as funções desenvolvidas.

Em outros casos, tinha-se o uso de um caractere '?' para designar que aquela informação não foi obtida. Nessas circunstancias, usou-se da função *replace*, também da biblioteca *Pandas* para fazer a mudança do caractere para um int que não estava sendo usado no banco (usualmente 0). Um exemplo de como que ficou um dos bancos de dados apos a operação de fatoração pode ser visto na Fig. 1. Nesse caso, temos que as colunas 0-13 representam os diferentes atributos da base de dados e a coluna 14 representa a classe (0 para quem ganha menos de 50 mil dólares por ano, 1 c.c).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	39	7	77516	9	13	4	1	1	4	1	2174	0	40	39	0
1	50	6	83311	9	13	2	4	0	4	1	0	0	13	39	0
2	38	4	215646	11	9	0	6	1	4	1	0	0	40	39	0
3	53	4	234721	1	7	2	6	0	2	1	0	0	40	39	0
4	28	4	338409	9	13	2	10	5	2	0	0	0	40	5	0

Fig. 1. Início do Banco de dados Adult fatorizado.

A base de dados *Adult* já vem com os dados de treino e teste divididos. Para as outras duas bases, os dados foram misturados para garantir a aleatoriedade e selecionou-se 70% dos dados para treino e 30% para teste.

B. Perceptron Simples

Para a criação de modelos de perceptron simples, usou-se funções que aplicam as noções vistas na seção II-A. Utilizando a linguagem de programação *Python*, criou-se duas funções: *trainperceptron*, que encontra os pesos w buscando minimizar o erro e, e *yperceptron*, que dado os pesos w, mapeiam os Y equivalente para uma entrada de teste xin.

Sendo assim, a função *trainperceptron* aplica o método para treinamento de perceptron. Ela recebe como parâmetros de entrada:

- yd: classe correspondente a linha de xin
- xin: entrada x de dados de matriz
- eta: valor de atualização do passo
- tol: tolerância do erro
- maxepocas: numero máximo de épocas permitido

E retorna:

- wt: pesos encontrados
- evec: erro médio por época

A *yperceptron*, por sua vez, retorna a saída de um sistema cujo parâmetros foram obtidos usando a função *trainperceptron*. Sendo assim, os parâmetros de entrada para essa função são:

- xin: vetor x de entrada
- w: pesos encontrados no *trainperceptron*

E retorna:

- y_pred: vetor y correspondente ao modelo com parâmetros w

Para o perceptron simples, todos os modelos foram treinados por 500 épocas e com uma tolerância de 0.001. Foram criados 10 modelos para cada banco de dados e a acurácia média desses modelos foi obtida. Para cada base, encontrou-se, experimentalmente, os valores da taxa de aprendizado.

C. Máquinas de Aprendizado Extremo

A criação da ELM utilizada nesse projeto foi feita seguindo as equações vistas em II-B. Sendo assim, foi criada duas funções na linguagem de programação *Python* para fazer o treinamento da ELM e a validação dos resultados.

Na função de treinamento se passa os dados de treino, os valores **Y** correspondentes e o número de neurônios que se deseja ter no modelo. A função de treinamento nos retorna os parâmetros **W, H** e **Z** encontrados para aquela quantidade de neurônios. Os parâmetros **W** e **Z** são passados para função *ELM_y* com os dados de de treino e retorna o **Y** previsto para cada amostra.

Para cada banco de dados, foram criadas ELMs com 100, 200, 300, 500, 1000, 1500 e 2000 neurônios para avaliar qual seria o impacto disso na acurácia obtida. Para cada número de neurônios, foram criados 10 modelos cada um sendo validado conta o conjunto de teste. A acurácia média para cada número de neurônio foi salva.

D. Redes com Base em Funções Radiais

Para a criação de redes com base em funções radiais, assim como nas outras instâncias, criou-se duas funções, uma para realização do treino dos parâmetros e outra para realização da validação (encontrar o Y estimado pelo modelo).

Na função de treinamento se passa os dados de treino, os valores **Y** correspondentes, o número de centros que se deseja utilizar e o método de clusterização a ser utilizado pelo algoritmo, no caso desse estudo optamos pelo uso do *kmeans*. A função de treinamento nos retorna: *m* (centro dos agrupamentos encontrados), *covlist* (lista de covariâcias para cada agrupamento), os parâmetros **W, H** vistos na seção II-C e o modelo *kmeans* gerado pela função do pacote *sklearn*. Esses dados são passados para função *YRBF* com os dados de de treino X. A função então retorna o **Y** previsto para cada amostra.

Para cada banco de dados criou-se modelos com 6 números de funções radiais diferentes. Cada modelo foi treinado 10 vezes e a acurácia média de cada modelo foi computada.

E. Perceptron de Múltiplas Camadas

Para criação do modelo MLP, optou pela utilização de um modelo externo por ser mais otimizado e permitir a variação dos parâmetros. Sendo assim, usou-se da classe *MLPClassifier* da biblioteca *SKLearn*. Os modelos MLP criados tiveram duas camadas escondidas com 20 e 10 neurônios, respectivamente. Os modelos foram treinados por 500 épocas. Todos os outros parâmetros foram mantidos como padrão.

Cada modelo foi treinado 10 vezes e a acurácia média foi computada.

Todos os códigos utilizados nesse trabalho, podem ser encontrados no repositório: https://github.com/GabrielEspeschit/Trabalho_Final_RNA.

IV. Resultados

A. Perceptron Simples

a) Base de dados Adult: : Para a base de dados *adult*, os resultados encontrados estão dispostos na Tabela I. Como podemos ver, alguns resultados foram próximos de 80%, uma métrica razoável para esse banco de dados. No entanto, outros sofreram alguma forma de possível *overfitting* e tiveram resultados próximos de 20%.

TABELA I
RESULTADOS DO PERCEPTRON SIMPLES PARA O ADULT

Iteração	Acurácia Obtida	Acurácia Média
1	23,6%	
2	79,6%	
3	23,6%	
4	23,6%	
5	78,9%	
6	77,9%	56,8%
7	23,6%	
8	78,5%	
9	79,7%	
10	79,5%	

b) Tic-Tac-Toe Endgame: Para a base de dados *Tic-Tac-Toe Endgame*, os resultados encontrados estão dispostos na Tabela II. Como podemos ver, os resultados foram próximos de 55%, uma métrica aquém do esperado para esse banco de dados.

TABELA II
RESULTADOS DO PERCEPTRON SIMPLES PARA O TIC-TAC-TOE ENDGAME

Iteração	Acurácia Obtida	Acurácia Média
1	62,2%	
2	51,4%	
3	52,1%	
4	42,7%	
5	55,2%	
6	56,6%	56,7%
7	66,0%	
8	58,7%	
9	62,2%	
10	50,4%	

c) Mammographic Masses: Para a base de dados *Mammographic Masses*, os resultados encontrados estão dispostos na Tabela III. Como podemos ver, os resultados foram melhores, com uma média de 71%. Ainda assim a métrica pode melhorar para esse banco de dados.

B. Máquinas de Aprendizado Extremo

a) Base de dados Adult: : Para a base de dados *adult*, os resultados encontrados estão dispostos na Figura 2. Como podemos ver, o melhor resultado obteve uma métrica de acurácia média próxima à 80% com 1500 neurônios. A disparidade entre o número de neurônios não foi muito grande.

TABELA III
RESULTADOS DO PERCEPTRON SIMPLES PARA O MAMMOGRAPHIC MASSES

Iteração	Acurácia Obtida	Acurácia Média
1	68,2%	
2	74,0%	
3	65,1%	
4	50,5%	
5	75,1%	
6	69,2%	71,4%
7	64,3%	
8	74,7%	
9	64,7%	
10	50,5%	

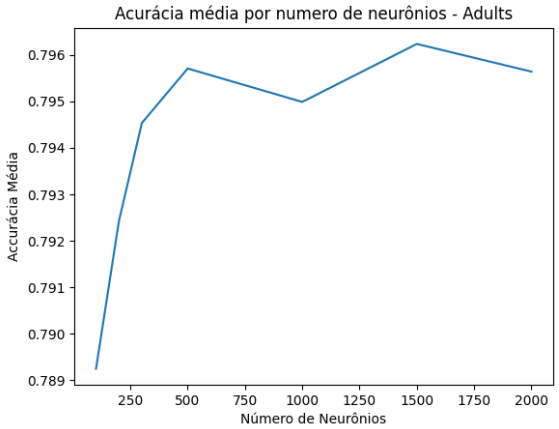

Fig. 2. Resultados da ELM aplicada ao banco Adult

b) Tic-Tac-Toe Endgame: : Para a base de dados *Tic-Tac-Toe Endgame*, os resultados encontrados estão dispostos na Figura 3. Como podemos ver, o melhor resultado obteve uma métrica de acurácia média acima de 90% com 500 neurônios. Nessa base, já podemos observar que utilizar mais neurônios na ELM causa uma grande diferença, visto que, usando entre 1 e 50 neurônios obtivemos acurácias abaixo de 80%.

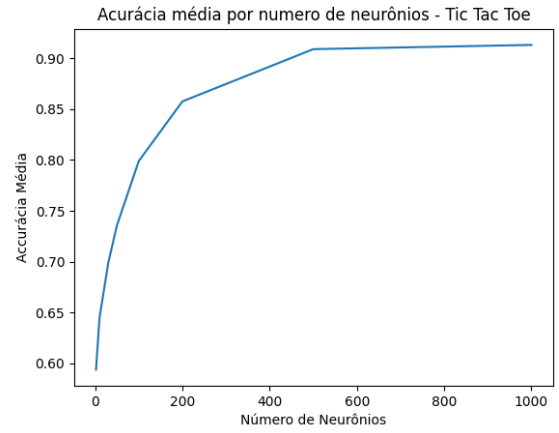

Fig. 3. Resultados da ELM aplicada ao banco Tic-Tac-Toe Endgame

c) Mammographic Masses: : Para a base de dados *Mammographic Masses*, os resultados encontrados estão dispostos na Figura 4. Como podemos ver, diferentemente dos bancos anteriores, o melhor resultado foi obtido usando menos neurônios. Observou-se uma acurácia média acima de 77% com 100 neurônios. Diferentemente do observado com as outras bases, aumentar o número de neurônios diminuiu a acurácia média obtida.

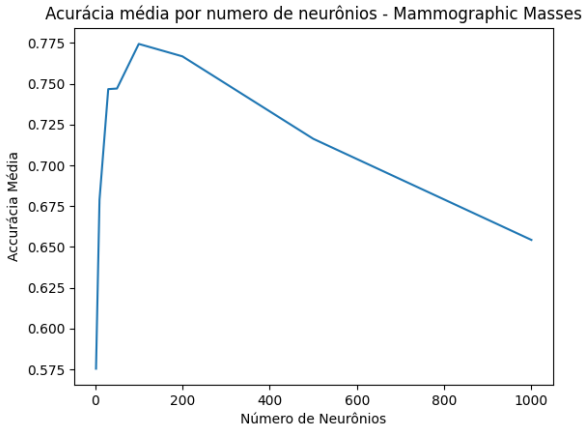

Fig. 4. Resultados da ELM aplicada ao banco Mammographic Masses

C. Redes com Base em Funções Radiais

a) Base de dados Adult: : Para a base de dados *adult*, os resultados encontrados estão dispostos na Figura 5. Podemos ver que aumentando ou diminuindo o número de funções radiais utilizadas não teve impacto na acurácia média obtida, ficando essa contante em 76,5%.

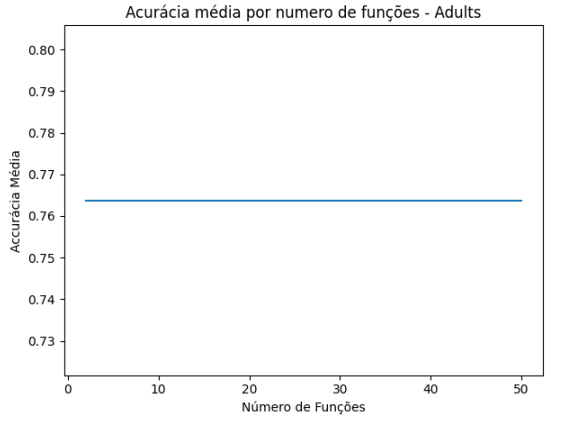

Fig. 5. Resultados da RBF aplicada ao banco Adult

b) Tic-Tac-Toe Endgame: : Para a base de dados *Tic-Tac-Toe Endgame*, os resultados encontrados estão dispostos na Figura 6. O melhor resultado foi obtido utilizando 10 funções radiais, em que se observou uma acurácia média de aproximadamente 80%. Diferente das ELMs, podemos ver que aumentar o número de neurônios na RBF não melhora o desempenho da rede.

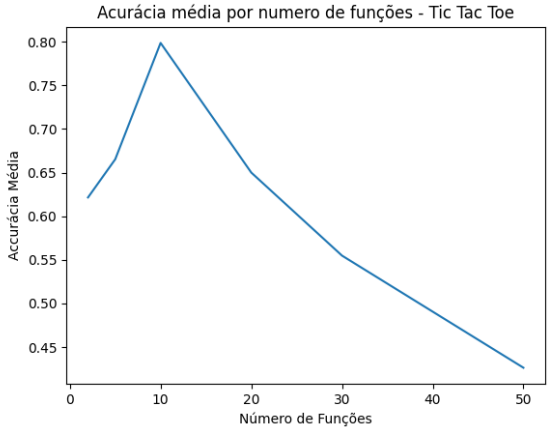

Fig. 6. Resultados da RBF aplicada ao banco Tic-Tac-Toe Endgame

c) Mammographic Masses: : Para a base de dados *Mammographic Masses*, os resultados encontrados estão dispostos na Figura 7. Nesse caso, o uso de RBFs se provou inferior ao uso de ELMs, pois a melhor acurácia obtida foi de 62%.

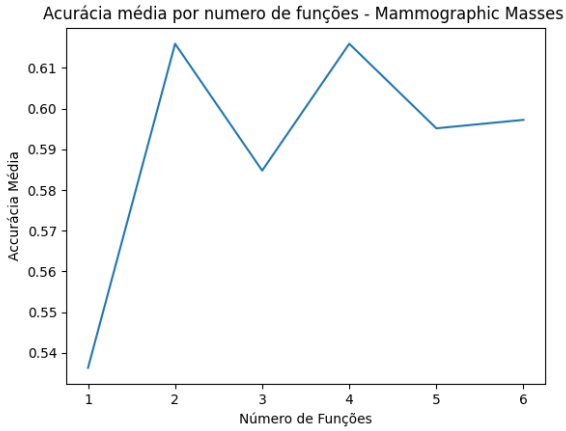

Fig. 7. Resultados da RBF aplicada ao banco Mammographic Masses

D. Perceptron de Múltiplas Camadas

a) Base de dados Adult: : Para a base de dados *adult*, os resultados encontrados estão dispostos na Tabela IV. Como podemos ver, maioria dos resultados foram próximos de 80%, uma métrica razoável para esse banco de dados. No entanto, assim como no caso do perceptron simples, um caso sofreu alguma forma de possível *overfitting* e teve resultados próximos de 20%. A média obtida foi de 71,5%

b) Tic-Tac-Toe Endgame: Para a base de dados *Tic-Tac-Toe Endgame*, os resultados encontrados estão dispostos na Tabela V. Como podemos ver, os resultados foram bem consistentes entre si, de tal modo que a média obtida para acurácia foi de 80,6%.

TABELA IV
RESULTADOS DO MLP PARA O ADULT

Iteração	Acurácia Obtida	Acurácia Média
1	76,7%	71,5%
2	79,1%	
3	79,4%	
4	23,6%	
5	76,6%	
6	62,1%	
7	79,1%	
8	78,9%	
9	79,6%	
10	79,6%	

TABELA V
RESULTADOS DO MLP PARA O TIC-TAC-TOE ENDGAME

Iteração	Acurácia Obtida	Acurácia Média
1	83,7%	80,6%
2	79,9%	
3	79,2%	
4	81,6%	
5	78,8%	
6	82,3%	
7	80,6%	
8	75,7%	
9	81,9%	
10	81,9%	

c) Mammographic Masses: Para a base de dados *Mammographic Masses*, os resultados encontrados estão dispostos na Tabela VI. Assim como no caso da base de dado *Tic-Tac-Toe Endgame*, os resultados foram bem consistentes entre si. A média obtida para acurácia foi de 79,4%.

TABELA VI
RESULTADOS DO MLP PARA O MAMMOGRAPHIC MASSES

Iteração	Acurácia Obtida	Acurácia Média
1	78,2%	79,4%
2	79,6%	
3	80,6%	
4	79,9%	
5	80,3%	
6	76,5%	
7	79,2%	
8	79,9%	
9	79,9%	
10	79,6%	

V. CONCLUSÃO

Como foi possível observar nos resultados, não existe uma metodologia que consistentemente entrega resultados mais acurados. No caso do banco de dados *Adult*, a melhor acurácia foi obtida utilizando ELM. No entanto, devido ao número de neurônios que foi utilizado, foi um algoritmo computacionalmente custoso, demorando mais de 30 minutos para executar. O uso de RBFs e MLPs também se justifica para resolução desse problema.

Para a base de dados *Tic-Tac-Toe Endgame*, a ELM também obteve o melhor resultado. Nesse caso, o uso da ELM se provou bastante adequado, pois utilizou um número razoável de neurônios e obteve-se um resultado de acurácia acima de 90%. O uso de MLPs e RBFs também são justificadas para esse banco de dados.

Por fim, para a base *Mammographic Masses*, o MLP obteve o melhor resultado entre os modelos investigados. O RBF teve um desempenho abaixo da média, mas isso pode ser devido a baixa disparidade dos dados que temos, impossibilitando uma clusterização de dados eficaz. Os métodos de ELM e Perceptron Simples também se justificam para resolução desse problema, ambos obtendo uma métrica acima de 70%.

A importância de um Engenheiro de Dados conhecer diferentes formas de fazer a classificação dos dados ficou muito evidente nesse estudo. Pode-se observar que, apesar da aplicação de múltiplas técnicas, não houve nenhuma que obteve resultados consistentemente melhores que as outras. Sendo assim, é de extrema importância compreender e aplicar diferentes técnicas e metodologias de classificação de dados por meio de RNAs para que se obtenha um resultado satisfatório e tenha certeza de que aquele resultado é o melhor que se pode obter.

REFERENCIAS

[1] A. P. Braga, “Introdução à Engenharia de Dados Uma Perspectiva de Redes Neurais Artificiais e Reconhecimento de Padrões,” Escola de Engenharia da UFMG, 2020.

[2] Dua, D. and Graff, C. “UCI Machine Learning Repository,” http://archive.ics.uci.edu/ml, Irvine, CA: University of California, School of Information and Computer Science, 2019

[3] Pedregosa, F, Varoquaux, G, Gramfort, A, Michel, V, Thirion, B, Grisel, O, Blondel, M, Prettenhofer, P, Weiss, R, Dubourg, V, Vanderplas, J, Passos, D, Brucher, M, Perrot, M, Duchesnay, E. ”Scikit-learn: Machine Learning in Python”. Journal of Machine Learning Research 2011; 12:2825–2830.

[4] Videoaulas do Professor Antônio Braga