UNIVERSIDADE FEDERAL DO CEARÁ - *CAMPUS* SOBRAL

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E COMPUTAÇÃO

INTELIGÊNCIA COMPUTACIONAL APLICADA (2021.2)

PROF. DR. MARCIO ANDRE BAIMA AMORA

KAMILA AMÉLIA SOUSA GOMES

RENAN HENRIQUE CARDOSO

APLICAÇÃO DAS REDES MLP E RBF NA BASE DE DADOS: *GAS SENSORS FOR HOME ACTIVITY MONITORING*

SOBRAL

2021

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AI	Artificial Intelligence
DANN	*Deep Autoencoder Neural Network*
ML	Machine Learning
MLP	*Multilayer Perceptron*
NN	*Neural Network*
RBF	*Radial Basis Function*
RNA	*Redes Neurais Artificiais*

SUMÁRIO

1 INTRODUÇÃO

A Inteligência Artificial (AI) é um ramo da ciência da computação que tem como propósito procurar compreender e desenvolver entidades inteligentes. Assim, dentro da AI, existe uma área denominada de aprendizado de máquina, mais conhecida do inglês como *machine learning* (ML), cujo o objetivo é na utilização de técnicas que adquiram conhecimento automático através de sistemas computacionais (SANTOS; ARAUJO, 2018).

As técnicas de ML têm sido aplicadas em diversos âmbitos reais, como exemplo: predição, detecção, diagnóstico e reconhecimento de fala. Essa nova metodologia têm conquistado gradativamente mais sucesso devido ao desenvolvimento de algoritmos cada vez mais eficientes e com melhor performance, além da transcendente capacidade dos recursos computacionais disponíveis atualmente. Outro fator importante, é sua capacidade de generalização, ou seja, os modelos elaborados são ainda capazes de lidar com situações não apresentadas durante seu desenvolvimento (VINICIUS, 2017).

Dessa forma, o presente trabalho pretende comparar o desempenho de duas abordagens inseridas nas redes neurais, que se configura como uma subárea do aprendizado de máquina. Portanto, as arquiteturas de rede MLP e RBF foram empregadas para realizar uma classificacão no conjunto de dados: *Gas Sensors for Home Activity Monitoring*. O *dataset* possui gravações de oito sensores de gás MOX e um sensor de temperatura e umidade. Os dados foram obtidos em três ambientes com condições diversas: presença de vinho, presença de banana e nenhum estimulante.

As redes MLP e RBF são arquiteturas do tipo *feedforward* não-lineares, com várias camadas e consideradas aproximadores universais. Numa comparação geral, a rede RBF tem como principal vantagem a rapidez no processo de treinamento enquanto a rede MLP tem um menor uso de memória (MELO *et al.*, 2014). Assim, para a realização dos experimentos deste trabalho, foi utilizado a *toolbox* de redes neurais artificiais do *software Matlab*, fornecendo todo o ferramental necessário para desenvolvimento e validação dos modelos gerados.

1.1 Objetivo Geral

Este trabalho tem como objetivo aplicar as arquiteturas de redes MLP e RBF para classificação do conjunto de dados *Gas Sensors for Home Activity Monitoring* para fins de comparação de desempenho das mesmas através de suas acurácias.

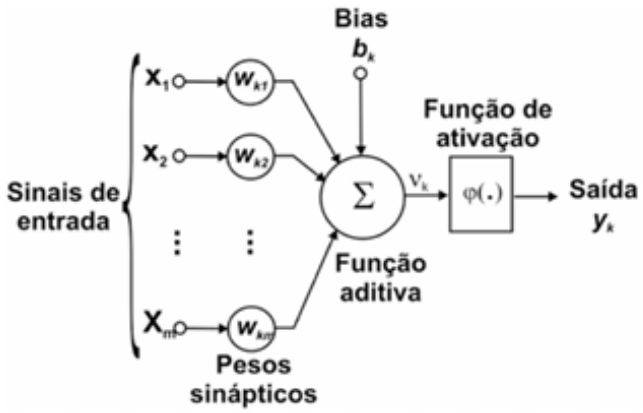
2 FUNDAMENTAÇÃO TEÓRICA

Esta seção aborda alguns dos conceitos fundamentais para o desenvolvimento do trabalho.

2.1 Redes Neurais Artificiais (RNA)

As Redes Neurais Artificiais (RNA) são exemplos de algoritmos de aprendizagem de máquina inspirados na rede de neurônios presente no cérebro biológico (SANTOS; ARAUJO, 2018). Apenas um único neurônio biológico é formado pelo corpo da célula, dendritos e o axônio. Os dendritos são os responsáveis por adquirir as informações através dos impulsos nervosos e conduzi-los até o corpo celular, onde essa informação é processada e por fim, acaba produzindo um novo impulso. O axônio é a parte do neurônio responsável pela condução dos impulsos. Com isso, chama-se de sinapse o ponto de contato entre a terminação do axônio de um neurônio e o dendrito de outro, formando-se assim um sistema capaz de executar a maioria das funções do cérebro (BONIFÁCIO, 2010). Através de uma visão matemática, o modelo de um neurônio artificial pode ser observado na Figura 1.

Figura 1 – Modelo matemático de um neurônio artificial.



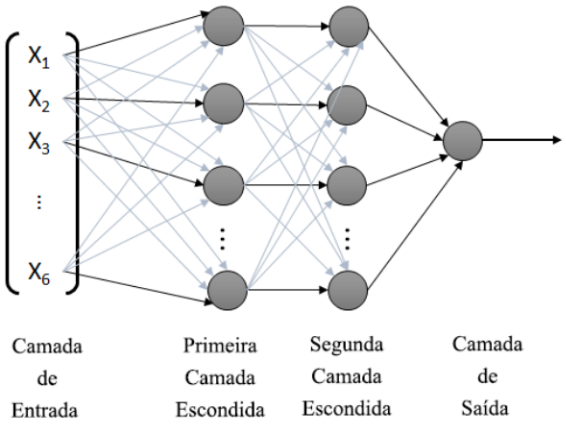
Fonte: (BONIFÁCIO, 2010)

Nota-se na Figura 1, que a informação é recebida pelos sinais de entrada, em seguida é aplicado um peso ou força sináptica, onde seu valor é multiplicado por valores positivos ou negativos. Em seguida, existe um combinador linear, responsável pela adição dos sinais de entrada ponderados pelos respectivos valores dos pesos sinápticos. O valor *bias* tem o efeito de aumentar ou diminuir a entrada da função de ativação. Por fim, a função de ativação define o valor de saída do neurônio, tipicamente com intervalos de [0,1] ou [-1,1] (BONIFÁCIO, 2010).

2.2 MLP

A *Multilayer Perceptron* (MLP) são redes *feedfoward*, ou seja, a propagação dos sinais ocorrem em um único sentido (da entrada para a saída) (TAVARES *et al.*, 2018). Essa rede é caracterizada por um conjuntos de neurônios, no seu estado mais simples apresenta uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída, no qual o sinal é propagado a cada camada. Ela é bastante utilizada em problemas de classificação, regressão, previsão e modelagem de séries temporais (BRITO *et al.*, 2020). Na Figura 2, é mostrada a arquitetura básica de uma MLP.

Figura 2 – Arquitetura de uma MLP



Fonte: (BRITO *et al.*, 2020)

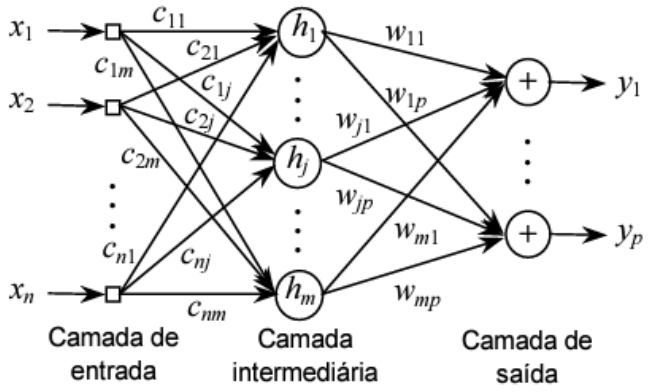
Através da Figura 2, percebe-se que a camada de entrada é responsável por adicionar padrões à rede neural e, nas camadas ocultas, é realizada uma grande parcela do processamento, por fim, as camadas de saída apresentam o resultado final (TAVARES *et al.*, 2018).

No treinamento da MLP é empregado o algoritmo denominado *backpropagation*, que ocorre em duas fases: a fase *forward* e a fase *backward*. Na fase *forward*, a entrada é apresentada à primeira camada da RNA, que calcula seus sinais de saída e passa os valores para a próxima camada, ocorrendo dessa forma até a camada de saída onde é calculado a saída final da RNA, onde posteriormente são comparadas com as saídas esperadas do modelo. Durante a fase *backward*, é realizado o caminho contrário, ou seja, os pesos dos neurônios vão sendo ajustados a partir da camada de saída até a camada de entrada, de forma a diminuir os erros. O algoritmo *backpropagation* tem atingindo sua popularidade por ter alcançado sucesso em diversas aplicações, apesar de ainda conter muitos desafios (BONIFÁCIO, 2010).

2.3 RBF

A Rede de Função de Base Radial (RBF) também são do tipo *feedforward* e completamente conectadas (BONIFÁCIO, 2010). Ela é formada na sua forma mais básica, pela camada de entrada, que está ligada às informações iniciais da rede, após, a uma única camada escondida que é formada por funções de ativação de base radial que desempenham uma transformação não linear do espaço de entrada e, por último, a camada de saída linear que proporciona o resultado do que foi aplicado nas entradas (BRITO *et al.*, 2020). A arquitetura da rede RBF é apresentada na Figura 3.

Figura 3 – Rede de Função de Base Radial(RBF)



Fonte: (ZUBEN, 2013)

A camada de entrada é responsável por agrupar os dados em *clusters*, utilizando funções de base radial, convertendo o conjunto dos padrões de entrada não linearmente separáveis em um conjunto de dados linearmente separáveis. Na camada oculta, por sua vez, é aplicada uma transformação não-linear do espaço de entrada para um espaço oculto de alta dimensionalidade. A função da camada de saída é classificar os padrões da camada anterior, através de operações simples por serem linearmente separados. A transformação não-linear seguida de uma transformação linear é baseada no Teorema de Cover, no qual afirma que um problema complexo de classificação de padrões alinhados não-linearmente em um espaço de alta dimensionalidade tem maior probabilidade de ser linearmente separável do que em um espaço de baixa dimensionalidade (BONIFÁCIO, 2010).

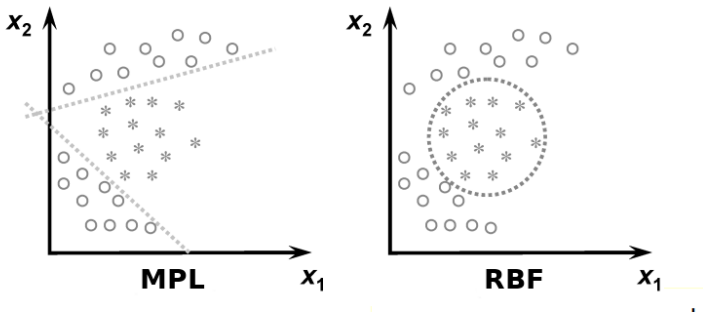
Pode-se dizer, que uma função é de base radial quando seus valores funcionais são iguais às normas de seus argumentos, ou seja, quando seu valor funcional depende apenas da distância de seu argumento à origem (MOTA *et al.*, 2011). Em uma RBF, as unidades escondidas

fornecem um conjunto de funções que representam uma base arbitrária para os vetores de entrada, sendo expandidas sobre o espaço oculto. Posteriormente, cada uma dessas funções é centrada em uma coordenada do espaço multidimensional e, essa coordenada, define o centro de uma região de maior aglomeração de pontos (*clusters*), do espaço de dados de entrada (KALMAN, S.D.).

2.3.1 MLP x RBF

A primeira característica que distingue essas duas abordagens diz respeito ao número de camadas, onde a rede RBF possui apenas uma camada oculta, enquanto a MLP possui muitas outras. A RBF geralmente possui mais neurônios na oculta, diferentemente da MLP que pode necessitar de um número maior de parâmetros ajustáveis, dessa forma, se tornando menos sensível a inserção de dados novos. A MLP gera regiões globais de decisão, tendo uma maior capacidade de generalização que a RBF, lidando melhor com *outliers*. A Figura 4 faz uma comparação gráfica entre as redes, mostrando que RBF separa classes por hiperelipsoides e a MLP por hiperplanos (BARRETO *et al.*, 2018).

Figura 4 – Comparação entre as redes RBF e MLP



Fonte: Adaptado de (BARRETO *et al.*, 2018)

2.4 Normalização *Z-Score* e *Min-Max*

A normalização é uma técnica comum aplicada durante a preparação dos dados quando se trabalha com aprendizado de máquina, isso porque, ela é responsável por modificar os valores de um conjunto de dados para uma escala comum, sem distorcer as diferenças nos intervalos de valores. Essa técnica só deve ser empregada quando os recursos têm intervalos diferentes (JAITLEY, 2018).

O Z-score, estrutura o dado em uma distribuição cuja média é determinada como 0 e o desvio padrão estabelecido como 1. Ele é definido como a diferença absoluta entre o valor do

dado e sua média, normalizado com o desvio padrão. A finalidade do Z-score é retirar os efeitos da localização e escala do dado, permitindo a comparação direta entre diferentes bases de dados. Esse método é eficiente em bases em que os dados seguem uma distribuição gaussiana, além de sua fácil implementação (TOCCI, 2018).

Por outro lado, a normalização Min-Max é um procedimento de converter valores discrepantes utilizando uma escala que vai de 0,0 a 1,0; onde 0 é para o menor valor e 1,0 para o maior valor. Com isso, a comparação de valores que foram medidos usando diferentes escalas é facilitada. (CARLOS, 2020).

2.5 Métricas

Em problemas de classificação, é necessário aplicar métricas para avaliar a eficiência do seu modelo (RODRIGUES, 2019). Algumas delas será mostrada a seguir.

2.5.1 Matriz de Confusão

A matriz de confusão refere-se a uma tabela que mostra os acertos e erros do seu modelo, comparando com o resultado esperado. A Figura 5 representa a Matriz de Confusão graficamente.

Figura 5 – Matriz de Confusão

		Detectada	
		Sim	**Não**
Real	**Sim**	Verdadeiro Positivo (VP)	Falso Negativo (FN)
	Não	Falso Positivo (FP)	Verdadeiro Negativo (VN)

Fonte: (RODRIGUES, 2019)

- Verdadeiros Positivos(VP): classificação correta da classe Positivo;
- Verdadeiros Negativos(VN): classificação correta da classe Negativo;
- Falsos Negativos (FN): erro em que o modelo previu a classe Negativo quando o valor real era classe Positivo;
- Falsos Positivos (FP): erro em que o modelo previu a classe Positivo quando o valor real era classe Negativo.

2.5.2 Acurácia

A Acurácia é uma métrica comum para avaliação de performace do modelo. Ela define dentre todas as classificações, quantas o modelo classificou corretamente (RODRIGUES, 2019). Sua formulação matemática é dada pela Equação 2.1.

$$A = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.1}$$

Onde,

- VP: refere-se a Verdadeiro Positivo;
- VN: refere-se a Verdadeiro Negativo;
- FP: refere-se a Falso Positivo;
- FN: refere-se a Falso Negativo.

3 EXPERIMENTOS

Nesta seção são apresentadas a descrição do experimento através da análise da base de dados utilizada e a aplicação do algoritmo com as fases de treinamento e teste.

3.1 Descrição da Base De Dados

O conjunto de dados utilizado para execução deste trabalho é denominado: *Gas sensors for home activity monitoring*, disponível em: <https://archive.ics.uci.edu/ml/datasets/Gas+sensors+for+home+activity+monitoring>. Esse conjunto de dados foi criado em 2016, contendo 919438 amostras e 11 atributos reais. Como atributos, existem 8 sensores MOX, um sensor de temperatura e um sensor de umidade, os outros atributos se referem à data e hora no qual foi realizada a gravação. Este *dataset* contém um conjunto de séries temporais em três condições de ambiente diferentes: presença de vinho, presença de banana e nenhum estimulante (denominado pelos autores de atividade de fundo, ou *background*). Ao todo, são 36 gravações onde foi apresentado vinho aos sensores, 33 foram apresentados banana e 31 das gravações com nenhum estimulante (*background*). Uma aplicação de classificação refere-se a discriminação entre as classes vinho, banana e *background*. Os 8 sensores MOX são: TGS2611(R1), TGS2612(R2), TGS2610 (R3), TGS2600 (R4), TGS2602 (R5), TGS2602 (R6), TGS2620 (R7) e TGS2620(R8). A temperatura e umidade foram medidas usando o sensor Sensirion SHT75.

3.2 Especificações de Hardware e Software

3.2.1 Hardware

Os experimentos foram realizados em:

1. Um computador equipado com um processador Intel Core i7-9750H, com 16GB de memória RAM, uma placa de vídeo NVIDIA GeForce GTX 1650 (4GB dedicado) e contando com um disco rígido SSD de 512 GB de capacidade.
2. Um computador equipado com um processador Intel Core i5-10210U, com 8GB de memória RAM, uma placa de vídeo NVIDIA GeForce MX110 (2 GB dedicado) e contando com armazenamento de disco rígido de 1 TB.

3.2.2 Software

O *software* utilizado para simulação dos experimentos foi o MATLAB na versão R2020a no Sistema Operacional *Windows* 10. A escrita do trabalho foi realizada no *Overleaf*, uma plataforma gratuita que possui um editor de LaTex online.

3.3 Preparação dos Dados

O início do processo de desenvolvimento do algoritmo é necessário primeiramente obter a base de dados. O conjunto de dados utilizado é composto por dois arquivos: HT_sensor_dataset.dat, onde as séries temporais reais são armazenadas, e HT_Sensor_metadata.dat, onde os metadados para cada indução foram armazenados. Cada indução é identificada exclusivamente por um id em ambos os arquivos. Assim, os metadados para uma indução particular podem ser facilmente encontrados combinando o id das colunas entre os arquivos. Dessa forma, a segunda etapa teve foco na realização de uma combinação das tabelas. Como mostrado na Figura 6.

Figura 6 – Leitura e combinação dos dados.

```
%% Importing data
ht_sensor = importdata('HT_Sensor_dataset.dat');
T = array2table(ht_sensor.data, 'VariableNames', ht_sensor.colheaders);
ht_sensor_metadata = readtable('HT_Sensor_metadata.csv');

%% Join tables by id
T = join(T, ht_sensor_metadata);
```

Fonte: Elaborada pelos Autores (2021)

Por se tratar de uma série temporal, a base contém informações não necessárias para a classificação, dessa forma, optou-se pela remoção de algumas colunas indesejadas, como: id, *time*, *date*, etc. Os atributos como temperatura e umidade foram removidos por serem condições do próprio ambiente. Como mostrado na Figura 7.

Figura 7 – Remoção de atributos

```
%% Preparing data
% Removing columns
T.id = [];
T.('Temp.') = [];
T.Humidity = [];
T.('time    ') = [];
T.date = [];
T.t0 = [];
T.dt = [];
```

Fonte: Elaborada pelos Autores (2021)

Em seguida, visto que os dados estão em uma escala diferente, é aplicado alguma normalização. Ela é importante para reduzir a redundância de dados, aumentar a integridade e o seu desempenho. Desta maneira, foi decidido a utilização de duas mais conhecidas: Z-Score e Min-Max, afim de testar qual apresenta melhor resultado, como mostrado na Figura 8. Além disso, é proposto utilizar nenhum tipo de normalização.

Figura 8 – Normalização dos dados

```
% Normalization data
% 0: No, 1: Z-score, 2: Minmax (integer)
norm = 1;
switch norm
    case 1
        for column = 1:8
            vector_column = T{:, column};
            T{:, column} = (vector_column - mean(vector_column))/std(vector_column);
        end
    case 2
        for column = 1:8
            vector_column = T{:, column};
            T{:, column} = (vector_column - min(vector_column))/(max(vector_column) - min(vector_column));
        end
end
```

Fonte: Elaborada pelos Autores (2021)

3.4 Experimento 1

Em busca de obter uma quantidade fixa de amostras, usamos o código apresentado na Figura 9. A função *datasample* é responsável por obter uma subamostra aleatória de um conjunto maior. Logo, são obtidas as mesmas quantidades de amostras para cada classe, o procedimento foi realizado pois a base não é totalmente balanceada entre os rótulos: vinho, banana e *background*.

Figura 9 – Quantidade fixa de amostras

```
% Sub-sampling
samples = 6000
T = datasample(T, samples);

% Get samples by class
banana = T(string(T{:, 9})=='banana', :);
wine = T(string(T{:, 9})=='wine', :);
bg = T(string(T{:, 9})=='background', :);
```

Fonte: Elaborada pelos Autores (2021)

Em seguida foi realizada a discretização das saídas, isso é importante para limitar o número de possíveis estados.

O código é mostrado na Figura 10.

Figura 10 – Discretização das Saídas

```
% Discretizing classes
banana_bin = array2table(ones(size(banana, 1), 3).*[1 0 0], 'VariableNames', {'y1', 'y2', 'y3'});
wine_bin = array2table(ones(size(wine, 1), 3).*[0 1 0], 'VariableNames', {'y1', 'y2', 'y3'});
bg_bin = array2table(ones(size(bg, 1), 3).*[0 0 1], 'VariableNames', {'y1', 'y2', 'y3'});
banana = [banana(:, 1:8) banana_bin];
wine = [wine(:, 1:8) wine_bin];
bg = [bg(:, 1:8) bg_bin];
```

Fonte: Elaborada pelos Autores (2021)

Por fim, para que a base possua a mesma quantidade de amostras, é calculado a quantidade de cada classe e obtido o valor mínimo dentre os 3 tamanhos, isso foi realizado para que a matriz T_b possua uma quantidade uniforme de amostras aleatórias em cada classe no conjunto total. A função *table2array()* converte a tabela T do tipo *table* em uma matriz do tipo *double* para posterior separação e entrada no modelo. A função *cvpartition* particiona as observações aleatoriamente em um conjunto de treinamento e de teste. O código é mostrado na Figura 11.

Figura 11 – Classes com mesma quantidade de amostras

```
% Data same quantity classes (balanced)
size_classes = [size(banana, 1) size(wine, 1) size(banana_bin, 1)];
n_min = min(size_classes);
T_b = [datasample(banana, n_min); datasample(wine, n_min); datasample(bg, n_min)];

T_b = table2array(T_b);

test_p = 0.15
cv = cvpartition(size(T_b, 1), 'holdout', test_p);
```

Fonte: Elaborada pelos Autores (2021)

3.4.1 Treinamento e Teste

O particionamento dos dados entre treino e teste foi feito através da técnica *Hold-out*. O conjunto de treino (*train*), são os dados que o modelo faz uso para obter a separação dentre as classes e, o conjunto de teste (*test*), são os dados que fazem parte para obter o quão bom esse modelo é executado em dados não conhecidos. Nesse caso, foi utilizado duas formas de divisão a fim de verificar qual apresenta melhor resultado: 85% treino e 15% teste e, 80% treino e 20% teste. Como mostrado na Figura 12.

Figura 12 – Divisão Treino e Teste

```
%% Separate to training and test data

test_data = T_b(cv.test, :);
train_data = T_b(cv.training, :);

train_y = train_data(:, 9:11);
train_x = train_data(:, 1:8);
test_y = test_data(:, 9:11);
test_x = test_data(:, 1:8);
```

Fonte: Elaborada pelos Autores (2021)

3.4.2 Experimentos MLP

Com os dados preparados e a divisão de treinamento já feita, pode-se elaborar o experimento para a primeira técnica, no caso a MLP. O seu código é mostrado na Figura 13.

Figura 13 – Código MLP - Experimento 1

```
%% MLP
layer1 = 50;
layer2 = 50;
net = feedforwardnet([layer1, layer2]);

% 1 'trainlm'   Levenberg-Marquardt         % 2 'trainbr'   Bayesian Regularization
% 3 'trainbfg' BFGS Quasi-Newton           % 4 'trainrp'   Resilient Backpropagation
% 5 'trainscg' Scaled Conjugate Gradient    % 6 'traincgb' Conjugate Gradient with Powell/Beale Restarts
% 7 'traincgf' Fletcher-Powell Conjugate Gradient         % 8 'traincgp' Polak-Ribiére Conjugate Gradient
% 9 'trainoss' One Step Secant % 10 'traingdx' Variable Learning Rate Gradient Descent
% 11 'traingdm' Gradient Descent with Momentum % 12 'traingd' Gradient Descent

net.trainFcn = 'trainlm';
net.trainParam.epochs = 1000;

mlp_net = train(net, train_x', train_y');
predict_y = mlp_net(test_x');
vec_ind_test = vec2ind(test_y');
vec_ind_pred = vec2ind(predict_y);
hit1 = sum(vec_ind_test == 1 & vec_ind_pred == 1);
hit2 = sum(vec_ind_test == 2 & vec_ind_pred == 2);
hit3 = sum(vec_ind_test == 3 & vec_ind_pred == 3);
hits = hit1 + hit2 + hit3;
acc = hits/size(vec_ind_pred, 2);

dlmwrite('test2.csv', [2, 1, samples, test_p, layer1, layer2, 1, 1000, hit1, hit2, hit3, hits, acc], 'delimiter', ';', '-append');
```

Fonte: Elaborada pelos Autores (2021)

Para o desenvolvimento da MLP, optou-se por uma arquitetura de duas camadas ocultas contendo 50 neurônios cada uma. Contendo 8 sensores de gás de entrada e camada de saída tem 3 neurônios para as 3 classes. Assim, a rede é iniciada na variável *net* com a função *feedforwardnet*, responsável por retornar uma rede neural *feedforward*. A variável *net.TrainFcn* especifica que o algoritmo *Levenberg-Marquardt* será utilizado para o treinamento. Já a variável *net.trainParam.epochs* informa que são usadas 1000 épocas. Enfim o treino é realizado nas linhas sucessivas. A estrutura da rede e os parâmetros foram definidos através de sucessivos testes, sendo escolhidos aqueles que apresentaram melhores acurácias. A última linha, que utiliza a função *dlmwrite*, é encarregada de gravar as informações de teste em um arquivo separadamente.

3.4.3 Experimentos RBF

Para o experimento da técnica RBF, é utilizado o seguinte treinamento, como mostrado na Figura 14.

Figura 14 – Código RBF - Experimento 1

```
%% RBF

goal = 0; % Error Limit
DF = 25; % Intervals between neurons
MN = 700; % neurons
spread = 40; % raio
rbf_nn = newrb(train_x', train_y', goal, spread, MN, DF);
predict_y = rbf_nn(test_x');
vec_ind_test = vec2ind(test_y');
vec_ind_pred = vec2ind(predict_y);
hit1 = sum(vec_ind_test == 1 & vec_ind_pred == 1);
hit2 = sum(vec_ind_test == 2 & vec_ind_pred == 2);
hit3 = sum(vec_ind_test == 3 & vec_ind_pred == 3);
hits = hit1 + hit2 + hit3;
acc = hits/size(vec_ind_pred, 2);

dlmwrite('test.csv', [1, 0, samples, test_p, MN, spread, hit1, hit2, hit3, hits, acc], 'delimiter', ';', '-append');
```

Fonte: Elaborada pelos Autores (2021)

Para o desenvolvimento da RBF, optou-se por uma arquitetura contendo 8 sensores de gás de entrada, uma camada oculta contendo 700 neurônios (representada pela variável MN) e para a camada de saída existem 3 neurônios para as 3 classes. A variável *goal* é a meta de desempenho do erro que é definida como 0. O parâmetro DF, responsável por definir o intervalo de neurônios para a plotagem do erro, é estabelecido com o valor 25. Já o parâmetro *spread* está relacionado com o tamanho do raio das redes RBF, e ele é definido como 40. A função *newrb* gera uma rede RBF com um neurônio para cada vetor de entrada, com uma largura das funções de base determinada por *spread*. Assim como na rede MLP, a estrutura da rede e os parâmetros também foram definidos através de testes sucessivos. A última linha, que utiliza a função *dlmwrite*, é encarregada de gravar as informações de teste em um arquivo separadamente.

3.5 Experimento 2 - abrangendo maior número de amostras

Em busca de obter uma quantidade de amostras por classe, usamos a primeira parte código apresentado na Figura 9. Após separar os dados de banana, vinho e *background*, é realizada a discretização das saídas. Isso é importante para limitar o número de possíveis estados, sua representação é mostrada na segunda parte código da Figura 15.

A Figura 16 mostra a parte do código responsável por obter as amostras de cada classe, randomizá-las e separá-las por treino e teste. Durante esse processo, é usado a função

Figura 15 – Discretização das Saídas

```
% Get samples by class
banana = T(string(T{:, 9})=='banana', :);
wine = T(string(T{:, 9})=='wine', :);
bg = T(string(T{:, 9})=='background', :);

% Discretizing classes
banana = [banana(:, 1:8) array2table(ones(size(banana, 1), 3).*[1 0 0], 'VariableNames', {'y1', 'y2', 'y3'})];
wine = [wine(:, 1:8) array2table(ones(size(wine, 1), 3).*[0 1 0], 'VariableNames', {'y1', 'y2', 'y3'});];
bg = [bg(:, 1:8) array2table(ones(size(bg, 1), 3).*[0 0 1], 'VariableNames', {'y1', 'y2', 'y3'});];
T = [banana; wine; bg];
T = table2array(T);
```

Fonte: Elaborada pelos Autores (2021)

datasample, que é responsável por obter uma subamostra aleatória de um conjunto maior.

Figura 16 – Classes com mesma quantidade de amostras

```
%% Get samples, randomize and separe train-test by class
a1 = 1;
a2 = 50000;
b1 = 50001;
b2 = 100000;

banana = T(T(:, 9)==1, :);
banana = datasample(banana, size(banana, 1));
train_banana = banana(a1:a2, :);
test_banana = banana(b1:b2, :);

wine = T(T(:, 10)==1, :);
wine = datasample(wine, size(wine, 1));
train_wine = wine(a1:a2, :);
test_wine = wine(b1:b2, :);

bg = T(T(:, 11)==1, :);
bg = datasample(bg, size(bg, 1));
train_bg = bg(a1:a2, :);
test_bg = bg(b1:b2, :);

samples_class_train = 5000;
samples_class_test = 500;
```

Fonte: Elaborada pelos Autores (2021)

3.5.1 Treinamento e Teste

Feito isso, o próximo passo é a execução de 10 ciclos com diferentes dados de treino e teste para cada classe, isso é importante para verificar se a eficácia do algoritmo se mantém quando aplicado diferentes dados. Para os dados de treino *loop* processado com a variável i, já para os dados de teste é utilizado a variável j. O ciclo j é rodado dentro do ciclo i. Exemplificando melhor, temos que são buscadas 15000 amostras para treino e 1500 para teste, isso é feito para que a cada rodada sejam usados diferentes dados para teste. A Figura 17 mostra o *loop* da variável i.

Figura 17 – *Loop* da variável i

```
for i = 1:10
    %% Training net

    train_lim_inf_class = (samples_class_train * (i-1)) + 1;
    train_lim_sup_class = samples_class_train * i;

    train_banana_i = train_banana(train_lim_inf_class:train_lim_sup_class, :);
    train_wine_i = train_wine(train_lim_inf_class:train_lim_sup_class, :);
    train_bg_i = train_bg(train_lim_inf_class:train_lim_sup_class, :);

    train_data = [train_banana_i; train_wine_i; train_bg_i;];
    train_data = datasample(train_data, size(train_data, 1));

    [trained_net, path_model, t_end] = mlp_net(train_data, i);
    % [trained_net, path_model] = rbf_net(train_data);

    [acc, C, hit1, hit2, hit3, hits] = get_results(train_data, trained_net);
    dlmwrite('./logs/results.csv', [i, 0, train_lim_inf_class, train_lim_sup_class, t_end, hit1, hit2, hit3, hits, acc], 'delimiter', ';', '-append');
```

Fonte: Elaborada pelos Autores (2021)

A Figura 18 mostra o *loop* da variável j.

Figura 18 – *loop* da variável j

```
%% Test trained net
for j = 1:10
    test_lim_inf_class = (samples_class_test * (j-1)) + 1;
    test_lim_inf_class = ((samples_class_test*10) * (i-1)) + test_lim_inf_class;
    test_lim_sup_class = samples_class_test * j;
    test_lim_sup_class = ((samples_class_test*10) * (i-1)) + test_lim_sup_class;

    test_banana_j = test_banana(test_lim_inf_class:test_lim_sup_class, :);
    test_wine_j = test_wine(test_lim_inf_class:test_lim_sup_class, :);
    test_bg_j = test_bg(test_lim_inf_class:test_lim_sup_class, :);

    test_data = [test_banana_j; test_wine_j; test_bg_j];
    test_data = datasample(test_data, size(test_data, 1));

    load(path_model, 'trained_net');

    [acc, C, hit1, hit2, hit3, hits] = get_results(test_data, trained_net);
    dlmwrite('./logs/results.csv', [i, j, test_lim_inf_class, test_lim_sup_class, 0, hit1, hit2, hit3, hits, acc], 'delimiter', ';', '-append');
end
dlmwrite('./logs/results.csv', ' ', '-append');
```

Fonte: Elaborada pelos Autores (2021)

3.5.2 Funções Auxiliares

A Figura 19 exibe uma função auxiliar que tem como objetivo calcular e plotar os resultados e apresenta como parâmetro: acertos na classe 1 (Hit1), Acertos na classe 2 (Hit2), Acertos na classe 3 (Hit3), soma de acertos das classes(Hits), acurácia(Acc) e a matriz de confusão(C).

Figura 19 – Função Auxiliar

```
%% Aux function to calcule and plot results
function [acc, C, hit1, hit2, hit3, hits] = get_results(test_data, trained_net)
    test_y = test_data(:, 9:11)';
    test_x = test_data(:, 1:8)';
    predict_y = trained_net(test_x);
    vec_ind_test = vec2ind(test_y);
    vec_ind_pred = vec2ind(predict_y);
    vec_size = size(vec_ind_pred, 2);
    hit1 = sum(vec_ind_test == 1 & vec_ind_pred == 1);
    hit2 = sum(vec_ind_test == 2 & vec_ind_pred == 2);
    hit3 = sum(vec_ind_test == 3 & vec_ind_pred == 3);
    hits = hit1 + hit2 + hit3;
    acc = hits/vec_size * 100;
    C = confusionmat(vec_ind_test, vec_ind_pred);
end
```

Fonte: Elaborada pelos Autores (2021)

3.5.3 *Experimentos MLP*

Com os dados preparados e a divisão de treinamento já feita, pode-se elaborar o experimento para a primeira técnica, no caso a MLP. O seu código é mostrado na Figura 20.

Figura 20 – Código MLP - Experimento 2

```
%% MLP Net
function [trained_net, path_model, t_end] = mlp_net(train_data, i)
    path_model = sprintf('./models/mlp_%d_restore.mat', i);
    layer1 = 50;
    layer2 = 50;
    f_training = 'trainlm';
    epochs = 100;
    feedforward = feedforwardnet([layer1, layer2]);
    feedforward.trainFcn = f_training;
    feedforward.trainParam.epochs = epochs;
    train_y = train_data(:, 9:11);
    train_x = train_data(:, 1:8);
    t_start = tic;
    trained_net = train(feedforward, train_x', train_y');
    t_end = toc(t_start);
    save(path_model, 'trained_net');
end
```

Fonte: Elaborada pelos Autores (2021)

Para o desenvolvimento da MLP, optou-se por uma arquitetura de duas camadas ocultas contendo 50 neurônios cada uma. Contendo 8 sensores de gás de entrada e camada de saída tem 3 neurônios para as 3 classes. Assim, a rede é iniciada na variável *net* com a função *feedforwardnet*, responsável por retornar uma rede neural *feedforward*. A variável *net.TrainFcn* especifica que o algoritmo *Levenberg-Marquardt* será utilizado para o treinamento. Já a variável *net.trainParam.epochs* informa que são usadas 1000 épocas. Enfim o treino é realizado nas linhas sucessivas. A estrutura da rede e os parâmetros foram definidos através de sucessivos testes, sendo escolhidos aqueles que apresentaram melhores acurácias.

3.5.4 *Experimentos RBF*

Para o experimento da técnica RBF, é utilizado o seguinte treinamento, como mostrado na Figura 21.

Para o desenvolvimento da RBF, optou-se por uma arquitetura contendo 8 sensores de gás de entrada, uma camada oculta contendo 700 neurônios (representada pela variável MN)

Figura 21 – Código RBF - Experimento 2

```
%% RBF Net
function [trained_net, path_model, t_end] = rbf_net(train_data, i)
    path_model = sprintf('./models/rbf_%d_restore.mat', i);
    goal = 0;
    DF = 25;
    MN = 100;
    spread = 50;
    train_y = train_data(:, 9:11)';
    train_x = train_data(:, 1:8)';
    t_start = tic;
    trained_net = newrb(train_x, train_y, goal, spread, MN, DF);
    t_end = toc(t_start);
    save(path_model, 'trained_net');
end
```

Fonte: Elaborada pelos Autores (2021)

e para a camada de saída existem 3 neurônios para as 3 classes. A variável *goal* é a meta de desempenho do erro que é definida como 0. O parâmetro DF, responsável por definir o intervalo de neurônios para a plotagem do erro, é estabelecido com o valor 25. Já o parâmetro *spread* está relacionado com o tamanho do raio das redes RBF, e ele é definido como 50. A função *newrb* gera uma rede RBF com um neurônio para cada vetor de entrada, com uma largura das funções de base determinada por *spread*. Assim como na rede MLP, a estrutura da rede e os parâmetros também foram definidos através de testes sucessivos.

4 RESULTADOS

Nesta seção são apresentadas os resultados obtidos através da metodologia aplicada, analisando-os através de métricas para avaliar a eficiência do experimento.

4.1 Experimento 1

4.1.1 Resultados MLP

A fim de obter o melhor resultado, o algoritmo precisou ser executados diversas vezes modificando os parâmetros de treinamento. Os melhores resultados serão exibidos na Figura 22.

Figura 22 – Resultados MLP

Norm	Samples	%Tests	layer1	layer2	trainFcn	epochs	Hit1	Hit2	Hit3	Hits:	Acc
1	15000	0.15	50	50	1	1000	698	681	766	2145	0.95887
1	10000	0.15	50	50	1	1000	483	443	489	1415	0.93647
1	10000	0.15	50	50	1	1000	452	450	506	1408	0.93369
1	6000	0.15	25	25	1	1000	281	249	281	811	0.91124
1	6000	0.15	75	75	1	1000	257	247	282	786	0.90345
1	6000	0.15	25	25	1	1000	267	263	267	797	0.89350
1	6000	0.15	25	25	1	1000	251	236	308	795	0.87942
1	8000	0.15	75	75	1	1000	349	333	372	1054	0.87542
1	10000	0.15	25	25	1	1000	407	370	499	1276	0.86508
1	6000	0.15	25	25	1	1000	230	197	299	726	0.80936

Fonte: Elaborada pelos Autores (2021)

A descrição das colunas apresentada na Figura 22, será mostrada abaixo:

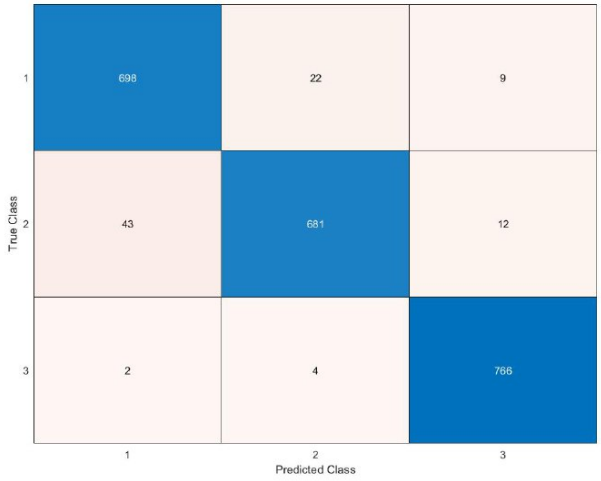
1. Norm: Normalização utilizada. Nenhuma normalização (0), z-score (1) e Min-Max (2);
2. Samples: Quantidade total de amostras;
3. %Tests: Parcela das amostras em % para teste;
4. layer1: Quantidade de neurônios na camada oculta 1;
5. layer2: Quantidade de neurônios na camada oculta 2;
6. trainFcn: Função de treinamento da rede;
7. epochs: Quantidade de épocas;
8. Hit1: Acertos na classe 1 (banana);
9. Hit2: Acertos na classe 2 (vinho);
10. Hit3: Acertos na classe 3 (*background* - ruído do fundo);
11. Hits: Acertos total;

12. Acc: Porcentagem de acertos.

Para análise e comparação dos resultados optou-se por criar um arquivo do tipo *csv* capaz de armazenar as informações de cada novo teste. Através dessa comparação, nota-se que o algoritmo se comporta melhor com 15000 amostras, com duas camadas ocultas de 50 neurônios e 1000 épocas. Foi utilizada 85% dados para treino e 15% dados para teste e a acurácia alcançou 95.887% com a normalização z-score.

Sua matriz de confusão é dada na Figura 25.

Figura 23 – Matriz de Confusão MLP



Fonte: Elaborada pelos Autores (2021)

4.1.2 Resultados RBF

A fim de obter o melhor resultado, o algoritmo precisou ser executados diversas vezes modificando os parâmetros de treinamento. Os melhores resultados serão exibidos na Figura 24.

A descrição das colunas apresentada na Figura 24, será mostrada abaixo:

1. Norm: Normalização utilizada. Nenhuma normalização (0), z-score (1) e Min-Max (2);
2. Samples: Quantidade total de amostras;
3. %Tests: Parcela das amostras em % para teste;
4. Neurons: Refere-se a quantidade de neurônios utilizados na rede;
5. Spread: Distância do raio RBF;
6. Hit1: Acertos na classe 1 (banana);
7. Hit2: Acertos na classe 2 (vinho);

8. Hit3: Acertos na classe 3 (*background* - ruído do fundo);
9. Hits: Acertos total;
10. Acc: Porcentagem de acertos.

Figura 24 – Resultados RBF

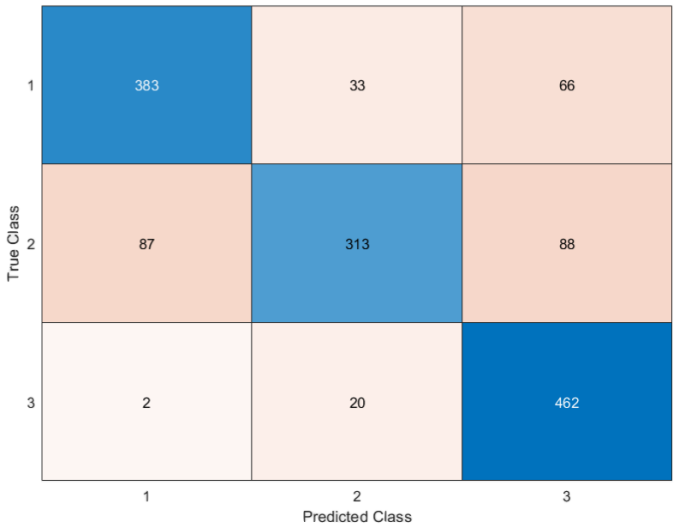
Norm	Samples	%Tests	Neurons	Spread	Hit1	Hit2	Hit3	Hits:	Acc
0	10000	0.15	700	40	383	313	462	1158	0.79642
0	12000	0.15	1000	40	418	391	581	1390	0.79202
0	3000	0.15	500	40	122	99	141	362	0.78867
0	15000	0.15	1200	50	545	483	699	1727	0.78109
1	6000	0.15	400	40	211	183	292	686	0.77955
0	10000	0.15	700	60	368	296	469	1133	0.76866
0	6000	0.15	500	50	223	173	301	697	0.76678
0	6000	0.15	800	80	222	191	255	668	0.76169
0	10000	0.15	400	60	353	320	454	1127	0.75435
0	3000	0.15	400	60	108	87	135	330	0.74830

Fonte: Elaborada pelos Autores (2021)

Através do arquivo *csv* com os testes executados, nota-se que o algoritmo se comportou melhor sem nenhum tipo de normalização, com 10000 amostras, 700 neurônios e um raio de 40. Foi utilizada 85% dados para treino e 15% dados para teste e a acurácia alcançou 79.642%.

Sua matriz de confusão é dada na Figura 25.

Figura 25 – Matriz de Confusão RBF



Fonte: Elaborada pelos Autores (2021)

4.2 Experimento 2 - abrangendo um maior número de amostras

4.2.1 Resultados MLP

A fim de obter o melhor resultado, o algoritmo precisou ser executado dentro de um *loop* de treinamento.A descrição das colunas apresentadas sobre os resultados, é mostrada abaixo:

1. i: Iteração de Treinamento;
2. j: Iteração dentro Treinamento (10 testes para cada modelo treinando);
3. lim_inf e lim_sup: variáveis de verificação do intervalo;
4. Hit1: Acertos na classe 1 (banana);
5. Hit2: Acertos na classe 2 (vinho);
6. Hit3: Acertos na classe 3 (*background* - ruído do fundo);
7. Hits: Acertos total;
8. Acc: Porcentagem de acertos.

Os melhores resultados da MLP serão exibidos na Figura 26. Os demais resultados podem ser averiguados no final deste trabalho na seção Anexos.

Figura 26 – Resultados MLP

i	j	lim_inf	lim_sup	Hit1	Hit2	Hit3	Hits	Acc
1	1	1	500	430	437	506	1373	91.533
2	**10**	**9501**	**10000**	**463**	**445**	**464**	**1372**	**91.467**
3	6	12501	13000	475	418	499	1392	92.8
4	5	17001	17500	459	458	492	1409	93.933
5	10	24501	25000	487	451	461	1399	93.267
6	**10**	**29501**	**30000**	**493**	**448**	**479**	**1420**	**94.667**
7	5	32001	32500	475	443	476	1394	92.933
8	6	37501	38000	470	419	495	1384	92.267
9	7	43001	43500	453	447	514	1414	94.267
10	8	48501	49000	464	422	508	1394	92.933
							Média:	93.030

Fonte: Elaborada pelos Autores (2021)

Para análise e comparação dos resultados optou-se por criar um arquivo do tipo *csv* capaz de armazenar as informações de cada novo teste. Através desse estudo, nota-se que o algoritmo conquista resultados satisfatórios usando 15.000 exemplos de treinamento (onde existem cerca de 5.000 para cada classe), divididos em 10 execuções, ou seja, cada teste existem 1500 exemplos, cerca de 500 para cada classe. Assim, o primeiro experimento com o modelo, apesar de usar uma parte pequena, refletiu bem sobre toda a base. A melhor acurácia atingida

pelo algoritmo foi de 94,667% com a normalização do tipo z-score.

Além disso, o conjunto de dados utilizado nos experimentos contém um número muito elevado de amostras usando muito poder computacional. Dessa forma, para efetuar os experimentos, optou-se por usar apenas uma parte desse conjunto. Para que esta parte faça uma boa representação do todo, optou-se por executar 10 ciclos com diferentes dados a fim de verificar a constância do algoritmo. A Figura 27 exibe a média da acurácia da MLP para cada ciclo de treinamento, mostrando que as médias se mantém com valores constantes ao longo do ciclo.

Figura 27 – Média Acurácias MLP

MLP	
Ciclo de Treinamento	**Média das Acurácias**
1	89.733
2	89.567
3	90.767
4	92.533
5	92.533
6	92.667
7	91.533
8	90.333
9	91.400
10	91.533

Fonte: Elaborada pelos Autores (2021)

4.2.2 Resultados RBF

A fim de obter o melhor resultado, o algoritmo precisou ser executado dentro de um *loop* de treinamento. A descrição das colunas apresentadas sobre os resultados, é mostrada abaixo:

1. i: Iteração de Treinamento;
2. j: Iteração dentro Treinamento (10 testes para cada modelo treinado);
3. lim_inf e lim_sup: variáveis de verificação do intervalo;
4. Hit1: Acertos na classe 1 (banana);
5. Hit2: Acertos na classe 2 (vinho);
6. Hit3: Acertos na classe 3 (*background* - ruído do fundo);
7. Hits: Acertos total;
8. Acc: Porcentagem de acertos.

Os melhores resultados da RBF serão exibidos na Figura 28. Os demais resultados podem ser averiguados no final deste trabalho na seção Anexos.

Figura 28 – Resultados RBF

i	j	lim_inf	lim_sup	Hit1	Hit2	Hit3	Hits	Acc
1	5	2001	2500	356	323	499	1178	78.533
2	2	5501	6000	388	322	475	1185	79
3	6	12501	13000	352	351	475	1178	78.533
4	**5**	**17001**	**17500**	**371**	**331**	**493**	**1195**	**79.667**
5	1	20001	20500	379	298	495	1172	78.133
6	9	29001	29500	394	281	500	1175	78.333
7	**4**	**31501**	**32000**	**372**	**294**	**502**	**1168**	**77.867**
8	1	35001	35500	372	330	472	1174	78.267
9	1	40001	40500	351	336	503	1190	79.333
10	10	49501	50000	382	332	463	1177	78.467
							Média:	78.400

Fonte: Elaborada pelos Autores (2021)

Através do arquivo *csv* com os testes executados, nota-se que o algoritmo se comportou melhor sem nenhum tipo de normalização, com 10000 amostras, 700 neurônios e um raio de 50. Foi utilizada 85% dados para treino e 15% dados para teste e a acurácia alcançou 79.667%.

Ademais, como a base de dados apresenta um número muito grande de amostras, para efetuar os experimentos viu-se a necessidade de usar apenas uma parte devido ao seu grande processamento computacional. Assim, para que esta parte faça uma boa representação do todo, optou-se por executar 10 ciclos com diferentes dados a fim de verificar a constância do algoritmo. A Figura 29 exibe a média da acurácia da RBF para cada ciclo de treinamento, mostrando que as médias se mantém com valores constantes ao longo do ciclo.

Figura 29 – Média das Acurácias RBF

RBF	
Ciclo de Treinamento	**Média das Acurácias**
1	76.467
2	76.667
3	76.867
4	76.667
5	76.533
6	76.533
7	75.133
8	76.733
9	77.467
10	76.000

Fonte: Elaborada pelos Autores (2021)

4.3 Comparação de Resultados com demais autores

Sharma (2017) utiliza Regressão Logística para a classificação da base de dados *Gas Sensors for Home Activity Monitoring*. A Regressão Logística é um classificador linear, onde durante seu experimento foi utilizado 70% dos dados para treino e 30% dos dados para teste. Além disso, o modelo obteve uma acurácia de 63%. No trabalho de Kumar Shanur Rahman (2020) utiliza *Random Forest* para classificar a mesma base, utilizando a divisão 67% dos dados para treino e 33% dos dados para teste, obtendo uma acurácia de 87%.

Mihaylova *et al.* (2019) aplicou Deep Autoencoder Neural Network (DANN) e uma rede neural não-profunda (NN). No algoritmo foi aplicado uma normalização de dados (0 e 1), em seguida houve uma redução de espaço de recurso baseado em PCA e também foram aplicados hiper-parâmetros de otimização. No modelo DANN foi utilizada duas camadas de *autoenconder* e para o modelo da NN somente uma única camada oculta, obtendo uma precisão do modelo a acurácia de 65.67% para NN e 55.89% para DANN.

Por fim, em (HUERTA *et al.*, 2016) é usado a técnica SVM's inibitórios (ISVM), um classificador não linear neste mesmo cenário. Ao aplicar a classificação apenas possuindo como entrada os sensores de gás, o modelo apresenta uma acurácia de 78,5%, já quando inclui os sensores de temperatura e umidade há uma queda para 76,5%. Os dados foram divididos em 80% para treino e 20% para teste.

Tabela 1 – Resultados Demais Autores

Autores	Técnicas	Acurácia
(SHARMA, 2017)	Regressão Logística	63%
(KUMAR SHANUR RAHMAN, 2020)	Random Forest	87%
(MIHAYLOVA *et al.*, 2019)	DANN e NN	55,44% e 68,78%
(HUERTA *et al.*, 2016)	SVM's inibitórios	78,5%

Fonte: Elaborada pelos autores (2021)

Tabela 2 – Resultados deste Trabalho

Autores	Técnicas	Acurácia
Experimento 1	**MLP** e RBF	**95,89%** e 79,642%
Experimento 2	**MLP** e RBF	**94,667%** e 79,667%

Fonte: Elaborada pelos autores (2021)

5 CONCLUSÕES

O objetivo deste trabalho foi em abordar o uso das redes MLP e RBF, através de implementações no software MATLAB, para classificação da base de dados *Gas Sensors for Home Activity Monitoring*, uma base sobre um conjunto de sensores sob diferentes condições em um ambiente doméstico: *background*, vinho e banana. Incluindo 8 sensores de gás MOX e sensores de umidade e temperatura.

No primeiro experimento, usou-se uma divisão de 85% para treino e 15 % para teste. Nele, observou-se que, a rede MLP se comporta melhor do que a rede RBF quando aplicada nesta base de dados, apresentando uma acurácia de 95.887% e de 79.642%, respectivamente. A rede MLP apresenta melhor desempenho quando se é aplicada 50 neurônios ou mais em suas camadas ocultas e com 15.000 quantidades de amostras, utilizando a normalização z-score. Já a RBF, em seu melhor desempenho, utiliza uma menor quantidade de mostras, cerca de 10.000, e 700 neurônios em sua camada oculta, usando nenhum tipo de normalização. No segundo experimento, quando são usados uma maior quantidade de amostras, a rede MLP atinge uma acurácia de 94,667% e a RBF 79.642%. Nele, são mantidos os mesmos parâmetros do experimento anterior, a alteração é na divisão de dados, onde são executados 10 ciclos (15000 amostras para treino e 1500 amostras para teste) a fim de verificar a constância do algoritmo.

Dessa forma, nota-se em embora a RBF não tenha apresentado um resultado satisfatório, ela apresenta resultados superiores em relação a algumas literaturas, como em Sharma (2017) que apresenta uma acurácia de 63%, Mihaylova *et al.* (2019) que apresenta uma acurácia de 55.44% e 68.78% e Huerta *et al.* (2016) que apresenta uma acurácia de 78,5%. Por sua vez, a MLP consegue os melhores resultados, superando a literatura de Kumar Shanur Rahman (2020) que apresenta uma acurácia de 87%.É importante ressaltar que o ganho de acurácia é fundamental para uma maior eficácia na classificação.

As MLPs podem apresentar melhor performance devido sua capacidade de poder conter mais de uma camada oculta, isso porque, cada camada é uma rede perceptron para cada grupo de entradas linearmente separáveis, o que facilita a tarefa de classificação. Ademais, a MLP apresenta uma maior capacidade de generalização que a RBF, lidando melhor com *outliers*.

REFERÊNCIAS

BARRETO, G. de A.; CAMPELLO, R. J. G. B.; ARAúJO, A. F. R.; VASCONCELOS, G. C.; ADEODATO, P. J. **Redes Neurais**. [S.l.], 2018. Disponível em: <chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/viewer.html?pdfurl=http%3A%2F%2Fwww.facom.ufu.br%2F~backes%2Fpgc204%2FAula07-RedesNeurais.pdf&clen=1201632&chunk=true>. Acesso em: 14 dez. 2021.

BONIFÁCIO, F. N. Comparação entre as redes neurais artificiais mlp, rbf e lvq na classificação de dados. **Paraná: Universidade Estadual do Oeste do Paraná**, 2010.

BRITO, R. X. de; FERNANDES, C. A. R.; AMORA, M. A. B. Análise de desempenho com redes neurais artificiais, arquiteturas mlp e rbf para um problema de classificação de crianças com autismo. **iSys-Revista Brasileira de Sistemas de Informação**, v. 13, n. 1, p. 60–76, 2020.

CARLOS, R. **Normalizando dados com R e Python**. [S.l.], 2020. Dados publicados pelo blog Medium. Disponível em: <https://datalivre.medium.com/normalizando-dados-com-r-e-python-fc048dd78ede#:~:text=A%20normaliza%C3%A7%C3%A3o%20de%20dados%20Min,0%20para%20o%20maior%20valor.> Acesso em: 14 dez. 2021.

HUERTA, R.; MOSQUEIRO, T.; FONOLLOSA, J.; RULKOV, N. F.; RODRIGUEZ-LUJAN, I. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. **Chemometrics and Intelligent Laboratory Systems**, Elsevier, v. 157, p. 169–176, 2016.

JAITLEY, U. **You have 2 free member-only stories left this month. Sign up for Medium and get an extra one**. [S.l.], 2018. Dados publicados pelo blog Master of Science - Business Analytics. Disponível em: <https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029>. Acesso em: 12 dez. 2021.

KALMAN, E. E. C. A. À. Rede neural com função de base radial com treinamento usando filtro de. S.D.

KUMAR SHANUR RAHMAN, N. A. **Aplicação de Random Forest em Gas Sensor for Home Activity Monitoring Dataset**. [S.l.], 2020. Dados publicados pela comunidade online Kaggle. Disponível em: <https://www.kaggle.com/nitinguptadu0621/kernel565f1f2b91>. Acesso em: 14 dez. 2021.

MELO, M. P. S. de; MAGALHÃES, M. L. C.; FERREIRA, M. M.; NUNES, H.; OLIVA, P.; MAIA, N. A.; TORCHELSEN, F. T. V.; PEREIRA, D. V. de C. Uma comparação entre as redes neurais artificiais mlp e rbf na classificação de dados. 2014.

MIHAYLOVA, P.; MANOLOVA, A.; GEORGIEVA, P. Data analytics for home air quality monitoring. In: SPRINGER. **International Conference on Future Access Enablers of Ubiquitous and Intelligent Infrastructures**. [S.l.], 2019. p. 79–88.

MOTA, J. da; SIQUEIRA, P.; SOUZA, L. de; VITOR, A. Uma rede neural de base radial baseada em computaçao evolucionária. **XXXII CILAMCE**, 2011.

RODRIGUES, V. **Métricas de Avaliação**. [S.l.], 2019. Dados publicados pelo blog Medium. Disponível em: <https://vitorborbarodrigues.medium.com/m%C3%A9tricas-de-avalia%

C3%A7%C3%A3o-acur%C3%A1cia-precis%C3%A3o-recall-quais-as-diferen%C3%A7as-c8f05e0a513c>. Acesso em: 12 dez. 2021.

SANTOS, G. S. dos; ARAUJO, A. F. de. Estudo sobre classificacao de lesoes de pele com rna-mlp. **Proceeding Series of the Brazilian Society of Computational and Applied Mathematics**, v. 6, n. 2, 2018.

SHARMA, S. **Aplicação de Regressão Logística em Gas Sensor for Home Activity Monitoring Dataset**. [S.l.], 2017. Dados publicados pelo repositório Github. Disponível em: <https://github.com/shubham0420/Gas-Sensor-for-Home-Activity-Monitoring-Dataset/blob/master/code/main_code_notebook.ipynb>. Acesso em: 14 dez. 2021.

TAVARES, J. T. S. *et al.* Sistema automático de negociação para a bolsa de valores utilizando redes neurais multilayer perceptron e regressão linear. Universidade Estadual de Feira de Santana, 2018.

TOCCI, A. **Técnicas de Detecção de Anomalias**. [S.l.], 2018. Dados publicados pelo blog oficial da comunidade DP6 Team. Disponível em: <https://blog.dp6.com.br/t%C3%A9cnicas-de-detec%C3%A7%C3%A3o-de-anomalias-3d9e216bf82e>. Acesso em: 12 dez. 2021.

VINICIUS, A. **Introdução ao Aprendizado de Máquina**. [S.l.], 2017. Dados publicados pelo blog Medium. Disponível em: <https://medium.com/@avinicius.adorno/introdu%C3%A7%C3%A3o-a-aprendizado-de-m%C3%A1quina-e39ec5ef459b>. Acesso em: 12 dez. 2021.

ZUBEN, I.-P. F. J. V. Redes neurais com função de ativação de base radial. 2013.

6 ANEXO

Resultados MLP

i	j	lim_inf	lim_sup	Hit1	Hit2	Hit3	Hits	Acc
1	0	1	5000	4515	4469	4929	13913	92.753
1	**1**	**1**	**500**	**430**	**437**	**506**	**1373**	**91.533**
1	2	501	1000	450	411	476	1337	89.133
1	3	1001	1500	409	439	486	1334	88.933
1	4	1501	2000	428	406	494	1328	88.533
1	5	2001	2500	432	382	525	1339	89.267
1	6	2501	3000	437	436	473	1346	89.733
1	7	3001	3500	457	424	474	1355	90.333
1	8	3501	4000	451	414	481	1346	89.733
1	9	4001	4500	423	440	500	1363	90.867
1	10	4501	5000	426	443	481	1350	90
2	0	5001	10000	4691	4307	4886	13884	92.56
2	1	5001	5500	473	379	484	1336	89.067
2	2	5501	6000	442	442	480	1364	90.933
2	3	6001	6500	449	392	498	1339	89.267
2	4	6501	7000	455	376	510	1341	89.4
2	5	7001	7500	437	424	496	1357	90.467
2	6	7501	8000	448	400	483	1331	88.733
2	7	8001	8500	431	413	509	1353	90.2
2	8	8501	9000	477	405	463	1345	89.667
2	9	9001	9500	449	403	490	1342	89.467
2	**10**	**9501**	**10000**	**463**	**445**	**464**	**1372**	**91.467**
3	0	10001	15000	4776	4474	4895	14145	94.3
3	1	10001	10500	482	364	511	1357	90.467
3	2	10501	11000	437	433	519	1389	92.6
3	3	11001	11500	439	415	497	1351	90.067
3	4	11501	12000	493	413	480	1386	92.4
3	5	12001	12500	478	394	499	1371	91.4
3	**6**	**12501**	**13000**	**475**	**418**	**499**	**1392**	**92.8**
3	7	13001	13500	457	426	503	1386	92.4
3	8	13501	14000	459	430	477	1366	91.067
3	9	14001	14500	457	409	508	1374	91.6
3	10	14501	15000	497	411	474	1382	92.133
4	0	15001	20000	4737	4558	4976	14271	95.14
4	1	15001	15500	438	434	489	1361	90.733
4	2	15501	16000	508	425	455	1388	92.533
4	3	16001	16500	434	433	514	1381	92.067
4	4	16501	17000	465	434	484	1383	92.2
4	**5**	**17001**	**17500**	**459**	**458**	**492**	**1409**	**93.933**
4	6	17501	18000	457	398	530	1385	92.333
4	7	18001	18500	481	441	480	1402	93.467
4	8	18501	19000	432	448	501	1381	92.067
4	9	19001	19500	446	414	534	1394	92.933
4	10	19501	20000	477	447	470	1394	92.933

5	0	20001	25000	4673	4608	4913	14194	94.627
5	1	20001	20500	442	478	472	1392	92.8
5	2	20501	21000	463	427	498	1388	92.533
5	3	21001	21500	463	429	493	1385	92.333
5	4	21501	22000	461	438	466	1365	91
5	5	22001	22500	468	429	465	1362	90.8
5	6	22501	23000	446	453	489	1388	92.533
5	7	23001	23500	461	390	517	1368	91.2
5	8	23501	24000	432	421	528	1381	92.067
5	9	24001	24500	458	407	531	1396	93.067
5	**10**	**24501**	**25000**	**487**	**451**	**461**	**1399**	**93.267**
6	0	25001	30000	4877	4631	4932	14440	96.267
6	1	25001	25500	490	398	493	1381	92.067
6	2	25501	26000	476	459	473	1408	93.867
6	3	26001	26500	457	447	475	1379	91.933
6	4	26501	27000	455	430	513	1398	93.2
6	5	27001	27500	436	480	494	1410	94
6	6	27501	28000	483	429	510	1422	94.8
6	7	28001	28500	484	419	487	1390	92.667
6	8	28501	29000	487	427	489	1403	93.533
6	9	29001	29500	513	429	453	1395	93
6	**10**	**29501**	**30000**	**493**	**448**	**479**	**1420**	**94.667**
7	0	30001	35000	4680	4563	4882	14125	94.167
7	1	30001	30500	419	427	525	1371	91.4
7	2	30501	31000	446	428	494	1368	91.2
7	3	31001	31500	452	432	499	1383	92.2
7	4	31501	32000	439	463	483	1385	92.333
7	**5**	**32001**	**32500**	**475**	**443**	**476**	**1394**	**92.933**
7	6	32501	33000	418	454	501	1373	91.533
7	7	33001	33500	464	414	502	1380	92
7	8	33501	34000	446	397	511	1354	90.267
7	9	34001	34500	473	405	485	1363	90.867
7	10	34501	35000	420	439	482	1341	89.4
8	0	35001	40000	4619	4518	4900	14037	93.58
8	1	35001	35500	440	401	495	1336	89.067
8	2	35501	36000	447	418	480	1345	89.667
8	3	36001	36500	433	440	497	1370	91.333
8	4	36501	37000	476	416	491	1383	92.2
8	5	37001	37500	480	398	481	1359	90.6
8	**6**	**37501**	**38000**	**470**	**419**	**495**	**1384**	**92.267**
8	7	38001	38500	427	427	470	1324	88.267
8	8	38501	39000	446	440	482	1368	91.2
8	9	39001	39500	453	444	494	1391	92.733
8	10	39501	40000	457	422	476	1355	90.333
9	0	40001	45000	4571	4514	5003	14088	93.92
9	1	40001	40500	398	436	508	1342	89.467

9	2	40501	41000	454	429	469	1352	90.133
9	3	41001	41500	405	443	515	1363	90.867
9	4	41501	42000	447	424	508	1379	91.933
9	5	42001	42500	448	419	483	1350	90
9	6	42501	43000	479	428	469	1376	91.733
9	**7**	**43001**	**43500**	**453**	**447**	**514**	**1414**	**94.267**
9	8	43501	44000	438	445	483	1366	91.067
9	9	44001	44500	446	448	496	1390	92.667
9	10	44501	45000	451	435	504	1390	92.667
10	0	45001	50000	4634	4549	4851	14034	93.56
10	1	45001	45500	482	393	481	1356	90.4
10	2	45501	46000	469	413	485	1367	91.133
10	3	46001	46500	429	449	495	1373	91.533
10	4	46501	47000	446	435	499	1380	92
10	5	47001	47500	486	431	465	1382	92.133
10	6	47501	48000	461	407	509	1377	91.8
10	7	48001	48500	477	411	485	1373	91.533
10	**8**	**48501**	**49000**	**464**	**422**	**508**	**1394**	**92.933**
10	9	49001	49500	479	448	451	1378	91.867
10	10	49501	50000	466	446	444	1356	90.4

Resultados RBF

i	j	lim_inf	lim_sup	Hit1	Hit2	Hit3	Hits	Acc
1	0	1	5000	3759	3189	4827	11775	78.5
1	1	1	500	351	291	489	1131	75.4
1	2	501	1000	351	295	474	1120	74.667
1	3	1001	1500	367	299	481	1147	76.467
1	4	1501	2000	344	309	495	1148	76.533
1	**5**	**2001**	**2500**	**356**	**323**	**499**	**1178**	**78.533**
1	6	2501	3000	370	314	469	1153	76.867
1	7	3001	3500	378	269	478	1125	75
1	8	3501	4000	340	341	465	1146	76.4
1	9	4001	4500	338	270	509	1117	74.467
1	10	4501	5000	356	308	449	1113	74.2
2	0	5001	10000	3654	3243	4757	11654	77.693
2	1	5001	5500	347	324	500	1171	78.067
2	**2**	**5501**	**6000**	**388**	**322**	**475**	**1185**	**79**
2	3	6001	6500	388	273	469	1130	75.333
2	4	6501	7000	351	308	420	1079	71.933
2	5	7001	7500	338	289	484	1111	74.067
2	6	7501	8000	401	270	482	1153	76.867
2	7	8001	8500	320	319	511	1150	76.667
2	8	8501	9000	341	296	485	1122	74.8
2	9	9001	9500	365	336	463	1164	77.6
2	10	9501	10000	372	329	461	1162	77.467
3	0	10001	15000	3700	3061	4905	11666	77.773
3	1	10001	10500	370	311	472	1153	76.867
3	2	10501	11000	318	344	486	1148	76.533
3	3	11001	11500	371	325	466	1162	77.467
3	4	11501	12000	379	268	495	1142	76.133
3	5	12001	12500	352	309	508	1169	77.933
3	**6**	**12501**	**13000**	**352**	**351**	**475**	**1178**	**78.533**
3	7	13001	13500	358	318	491	1167	77.8
3	8	13501	14000	376	306	468	1150	76.667
3	9	14001	14500	370	287	469	1126	75.067
3	10	14501	15000	359	270	502	1131	75.4
4	0	15001	20000	3532	3435	4770	11737	78.247
4	1	15001	15500	330	317	494	1141	76.067
4	2	15501	16000	357	330	495	1182	78.8
4	3	16001	16500	372	276	485	1133	75.533
4	4	16501	17000	360	328	485	1173	78.2
4	**5**	**17001**	**17500**	**371**	**331**	**493**	**1195**	**79.667**
4	6	17501	18000	356	315	464	1135	75.667
4	7	18001	18500	353	292	498	1143	76.2
4	8	18501	19000	358	351	465	1174	78.267
4	9	19001	19500	387	298	473	1158	77.2
4	10	19501	20000	357	305	488	1150	76.667

5	0	20001	25000	3702	3161	4795	11658	77.72
5	**1**	**20001**	**20500**	**379**	**298**	**495**	**1172**	**78.133**
5	2	20501	21000	368	311	492	1171	78.067
5	3	21001	21500	389	292	480	1161	77.4
5	4	21501	22000	328	301	496	1125	75
5	5	22001	22500	367	306	475	1148	76.533
5	6	22501	23000	352	311	495	1158	77.2
5	7	23001	23500	343	329	488	1160	77.333
5	8	23501	24000	350	333	462	1145	76.333
5	9	24001	24500	346	328	447	1121	74.733
5	10	24501	25000	342	329	477	1148	76.533
6	0	25001	30000	3806	3052	4751	11609	77.393
6	1	25001	25500	395	267	465	1127	75.133
6	2	25501	26000	400	295	455	1150	76.667
6	3	26001	26500	375	292	490	1157	77.133
6	4	26501	27000	371	273	485	1129	75.267
6	5	27001	27500	372	316	476	1164	77.6
6	6	27501	28000	322	316	489	1127	75.133
6	7	28001	28500	380	284	484	1148	76.533
6	8	28501	29000	387	284	474	1145	76.333
6	**9**	**29001**	**29500**	**394**	**281**	**500**	**1175**	**78.333**
6	10	29501	30000	361	275	483	1119	74.6
7	0	30001	35000	3670	3095	4783	11548	76.987
7	1	30001	30500	397	296	453	1146	76.4
7	2	30501	31000	371	290	466	1127	75.133
7	3	31001	31500	379	310	477	1166	77.733
7	**4**	**31501**	**32000**	**372**	**294**	**502**	**1168**	**77.867**
7	5	32001	32500	374	293	503	1170	78
7	6	32501	33000	339	283	475	1097	73.133
7	7	33001	33500	356	305	493	1154	76.933
7	8	33501	34000	364	295	469	1128	75.2
7	9	34001	34500	389	273	465	1127	75.133
7	10	34501	35000	367	301	455	1123	74.867
8	0	35001	40000	3669	3217	4788	11674	77.827
8	**1**	**35001**	**35500**	**372**	**330**	**472**	**1174**	**78.267**
8	2	35501	36000	350	319	482	1151	76.733
8	3	36001	36500	350	367	445	1162	77.467
8	4	36501	37000	368	339	469	1176	78.4
8	5	37001	37500	367	315	442	1124	74.933
8	6	37501	38000	366	306	478	1150	76.667
8	7	38001	38500	348	291	477	1116	74.4
8	8	38501	39000	347	295	488	1130	75.333
8	9	39001	39500	369	304	501	1174	78.267
8	10	39501	40000	358	324	427	1109	73.933
9	0	40001	45000	3769	3099	4833	11701	78.007
9	**1**	**40001**	**40500**	**351**	**336**	**503**	**1190**	**79.333**

9	2	40501	41000	360	287	495	1142	76.133
9	3	41001	41500	371	290	470	1131	75.4
9	4	41501	42000	373	306	462	1141	76.067
9	5	42001	42500	347	281	497	1125	75
9	6	42501	43000	355	320	488	1163	77.533
9	7	43001	43500	369	292	474	1135	75.667
9	8	43501	44000	380	319	463	1162	77.467
9	9	44001	44500	354	343	440	1137	75.8
9	10	44501	45000	360	301	502	1163	77.533
10	0	45001	50000	3733	3029	4736	11498	76.653
10	1	45001	45500	367	303	481	1151	76.733
10	2	45501	46000	353	301	478	1132	75.467
10	3	46001	46500	338	305	505	1148	76.533
10	4	46501	47000	389	301	433	1123	74.867
10	5	47001	47500	348	313	449	1110	74
10	6	47501	48000	349	284	506	1139	75.933
10	7	48001	48500	366	285	453	1104	73.6
10	8	48501	49000	374	309	458	1141	76.067
10	9	49001	49500	321	281	519	1121	74.733
10	**10**	**49501**	**50000**	**382**	**332**	**463**	**1177**	**78.467**