



CMC-15 - Inteligência Artificial

Relatório do Laboratório 2 - O Problema de Classificação entre Democratas e Republicanos com Rede Neural com Arquitetura RBF

Grupo:

Gabriel Telles Missailidis - gabriel.missailidis@ga.ita.br

João Lucas Rocha Rolim - joao.rolim@ga.ita.br

Samir Nunes da Silva - samir.silva@ga.ita.br

Professor:

Paulo Marcelo Tasinaffo

07/11/2024

Instituto Tecnológico de Aeronáutica – ITA

São José dos Campos, SP

1 Introdução

O problema de classificação entre democratas e republicanos encontra desafios específicos na escolha e ajuste de modelos de aprendizado. Em especial, considerando o contexto de redes neurais, como as redes com regularização e as redes baseadas em funções de base radial (Radial Basis Function - RBF), é importante realizar estudos relativos ao seu desempenho nesse problema de classificação.

Nesse contexto, as redes RBF são particularmente úteis, pois permitem um mapeamento de características não lineares em uma nova representação linear, facilitando a separação entre as classes. Além disso, a utilização de regularização é essencial para lidar com o sobreajuste (overfitting), principalmente em cenários onde o número de parâmetros de ajuste pode se tornar elevado em comparação ao número de dados de treinamento.

O presente trabalho se propõe a investigar o impacto de diferentes configurações de rede neural sobre as acurácias nos conjuntos de treino e de teste. Em particular, explora-se a utilização da matriz pseudo-inversa na determinação dos pesos para a rede RBF e os efeitos do termo de regularização λ nas redes de regularização. A implementação computacional segue princípios estabelecidos por literaturas especializadas e considera experimentos para análise de diferentes valores de λ e do número de neurônios na camada RBF. Dessa forma, visa-se fornecer resultados acerca da influência dessas variáveis no desempenho de redes neurais voltadas à classificação, utilizando conjuntos de dados balanceados e a divisão de treino/teste de 80/20.

2 Objetivos

O objetivo do trabalho é resolver o problema de classificação entre democratas e republicanos utilizando redes de regularização e RBF, em especial considerando o conceito da matriz pseudo-inversa no caso da formulação da camada da rede RBF. Além disso, tem-se por objetivo realizar uma série de tarefas para se estudar a variação das acurácias nos conjuntos de treino e de teste (divididos na proporção de 80/20) conforme se altera o coeficiente λ de regularização na rede de regularização formulada considerando uma função de Green gaussiana (Radial Basis Function - RBF) e o número de neurônios da camada RBF no caso da rede RBF. Finalmente, também se estuda como a matriz de pesos tem sua dimensão alterada conforme se muda o número de neurônios na rede RBF.

3 Metodologia

3.1 Teoria

Para a resolução do problema proposto, baseou-se em duas principais referências:

- <https://innovationyourself.com/radial-basis-function-networks-rbfn/>
- <https://towardsdatascience.com/most-effective-way-to-implement-radial-basis-function-neural-network-for-classification-problem-33c467803319>

Por meio de ambas, tem-se um norte acerca da implementação computacional. Além delas, considerou-se a teoria apresentada no roteiro do laboratório. A Figura 1 mostra como o cálculo dos pesos (treinamento) de redes de regularização e de redes RBF é feito, bem como o valor da largura σ da função de base radial é calculado considerando o número total de neurônios na camada interna e a distância máxima entre todos os centróides c_i . Já a Figura 2 mostra como o cálculo dos pesos de uma rede de regularização é feito quando se considera o termo de regularização de Tikonov.

$$\underbrace{\begin{bmatrix} \exp\left(-\frac{\|\vec{x}_1 - \vec{c}_1\|^2}{2\sigma^2}\right) & \exp\left(-\frac{\|\vec{x}_1 - \vec{c}_2\|^2}{2\sigma^2}\right) & \dots & \exp\left(-\frac{\|\vec{x}_1 - \vec{c}_m\|^2}{2\sigma^2}\right) \\ \exp\left(-\frac{\|\vec{x}_2 - \vec{c}_1\|^2}{2\sigma^2}\right) & \exp\left(-\frac{\|\vec{x}_2 - \vec{c}_2\|^2}{2\sigma^2}\right) & \dots & \exp\left(-\frac{\|\vec{x}_2 - \vec{c}_m\|^2}{2\sigma^2}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \exp\left(-\frac{\|\vec{x}_p - \vec{c}_1\|^2}{2\sigma^2}\right) & \exp\left(-\frac{\|\vec{x}_p - \vec{c}_2\|^2}{2\sigma^2}\right) & \dots & \exp\left(-\frac{\|\vec{x}_p - \vec{c}_m\|^2}{2\sigma^2}\right) \end{bmatrix}}_{p \times m} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}}_{m \times 1} = \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_p \end{bmatrix}}_{p \times 1}$$

onde,

p ... número total de padrões de treinamento

m ... número total de neurônios na camada interna

$f(\vec{x}_i) = d_i$ para $i=1, 2, \dots, p$ $\vec{x}_i \in \mathbb{R}^n$ e $d_i \in \mathbb{R}$

Rede de Regularização:

$$\sigma = \frac{\text{dist}_{\max}}{\sqrt{2m}}$$

1. $m = p$

2. $\vec{w} = \Phi^{-1} \cdot \vec{d}$

3. Os centros \vec{c}_i são os próprios \vec{x}_i , ou seja, $\vec{c}_i = \vec{x}_i$ para $i=1, \dots, p$

Redes RBF

1. $m \ll p$

2. $\vec{w} = \Phi^+ \cdot \vec{d}$ para $\Phi^+ = (\Phi^T \Phi)^{-1} \cdot \Phi^T$ (pseudo-inversa)

3. Os centros \vec{c}_i não coincide com os \vec{x}_i (há vários métodos para calculá-los)

Figura 1: Cálculo dos pesos de uma rede de regularização e de uma rede RBF.

Determinação dos Coeficientes da Expansão

Introduzimos agora as seguintes definições:

$$F_\lambda = [F_\lambda(x_1), F_\lambda(x_2), \dots, F_\lambda(x_n)]^T \quad (21)$$

$$d = [d_1, d_2, \dots, d_n]^T \quad (22)$$

$$G = \begin{bmatrix} G(x_1, x_1) & G(x_1, x_2) & \dots & G(x_1, x_N) \\ G(x_2, x_1) & G(x_2, x_2) & \dots & G(x_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ G(x_N, x_1) & G(x_N, x_2) & \dots & G(x_N, x_N) \end{bmatrix} \quad (23)$$

$$W = [w_1, w_2, \dots, w_n]^T \quad (24)$$

As classes de funções de Green cobertas pelo teorema de Micchelli incluem *multiquádricas inversas* e *funções gaussianas*, mas não *multiquádricas*. Na prática, podemos sempre escolher λ suficientemente grande para garantir que $G + \lambda I$ seja definida positivamente e assim inversiva. Isto, por sua vez, significa que o sistema linear de Equações (27) terá uma única solução dada por

$$w = (G + \lambda I)^{-1} d \quad (28)$$

Assim, tendo selecionado o operador diferencial D e portanto tendo identificado a função de Green associada $G(x_j, x_i)$, onde $i = 1, 2, \dots, N$, podemos usar a Eq. (28) para obter o vetor de peso w para um vetor resposta desejada específico d e um valor apropriado de parâmetro de regularização λ .

Concluindo, podemos afirmar que a solução do problema de regularização é dada pela expansão:

$$F_\lambda(x) = \sum_{i=1}^N \omega_i G(x, x_i) \quad (29)$$

Figura 2: Cálculo dos pesos de uma rede de regularização considerando o termo de regularização de Tikonov.

3.2 Implementação

Os códigos listados no Apêndice B referem-se à implementação realizada para a resolução do problema proposto.

Em conformidade com o que foi explicado anteriormente, implementa-se uma rede neural radial baseada em função de base radial (RBF) com regularização e suas variações. A funcionalidade principal é dividida entre os arquivos `rbf_net.py`, `reg_net.py`, `tasks.py` e `utils.py`. O código utiliza a biblioteca de logging do Python, definida em `logger.py`, para registrar as mensagens e resultados no console.

Em `rbf_net.py`, a classe `RBFLayer` define uma camada que utiliza funções de base radial, sendo responsável por calcular os centros e a largura (**sigma**) das funções de base radial para uma rede neural RBF. No método `fit`, os centros são selecionados aleatoriamente a partir do conjunto de dados de entrada, e o valor de **sigma** é calculado com base na distância euclidiana máxima entre os centros. A previsão é feita pelo método `predict`, que aplica a função de base radial gaussiana entre as amostras de entrada e os centros.

A classe `RBFNetwork` em `rbf_net.py` utiliza a `RBFLayer` como camada de entrada e calcula pesos para aproximar as saídas desejadas no treinamento. No método `fit`, a matriz de saída `phi` é calculada a partir da camada RBF, e os pesos são obtidos através da pseudoinversa de `phi` multiplicada pelo vetor de saídas esperadas, conforme formulação da Figura 1. O método

`predict` usa a camada RBF e os pesos treinados para prever as saídas de novas amostras, retornando uma classificação binária.

Em seguida, o arquivo `reg_net.py` contém classes para redes com e sem regularização. A classe `NoRegularizationNetwork` herda de `BaseNetwork` e implementa uma rede sem regularização, onde a matriz de pesos é obtida pela inversa de `phi`, conforme formulações da Figura 1. Caso a matriz seja singular, o logger informa o erro. A `RegularizationNetwork`, por outro lado, inclui o parâmetro `lambda` de regularização, que é usado no cálculo da inversa regularizada da matriz `G`, conforme a teoria da Figura 2.

Por sua vez, o arquivo `tasks.py` define várias funções de tarefas (de `a` a `f`) para experimentar as diferentes configurações de redes neurais solicitadas. Cada função realiza treinamento e teste usando uma das redes definidas em `rbf_net.py` e `reg_net.py`. Para cada configuração, os pesos e as acurácias de treino e teste são registradas com o logger. A tarefa `b`, por exemplo, experimenta diferentes valores de `lambda` para a rede com regularização e registra os resultados para cada valor.

A função `import_data` em `utils.py` realiza a importação dos dados de um arquivo CSV e transforma as variáveis categóricas para valores inteiros. Após o pré-processamento, os dados são divididos em conjuntos de treino e teste pelo método `_split_data`, que aplica `train_test_split` para um particionamento de 80/20.

O arquivo `main.py` executa o conjunto de tarefas importado de `tasks.py` ao listar cada uma das funções e executá-las em sequência. Isso permite realizar todos os experimentos definidos com diferentes configurações de rede neural ao executar o script `main.py`.

4 Resultados

Com base no exposto anteriormente, determine:

- a) Os pesos desta rede utilizando a Rede de Regularização de acordo com a Figura 02;
- b) Os pesos da Rede de Regularização de acordo com a Figura 03 e equação (28). Realizar 5 estudos de caso, a saber, para `lambda=0`, `lambda=1`, `lambda=10`, `lambda=100` e `lambda=1000`.
- c) Os pesos desta rede utilizando a Rede RBF com 14 neurônios na camada interna;
- d) Repita o item anterior com 25 neurônios na camada interna;
- e) Repita o item anterior com 50 neurônios na camada interna;
- f) Repita o item anterior com 80 neurônios na camada interna.

Figura 3: Tarefas solicitadas no roteiro do trabalho.

A seguir, são apresentados os resultados de cada um dos itens solicitados no roteiro do laboratório (vide Figura 3).

```
1
2 a)
3 Matrix phi is singular (determinant = 0), generating the error: Singular
  matrix.
4 This happens because there's no regularization (lambda = 0).
5 "Regularization" network with no regularization
6 Network weights:
7 None
8
9
10 b)
```

```

11 Matrix phi is singular (determinant = 0), generating the error: Singular
    matrix.
12 This happens because there's no regularization (lambda = 0).
13 Regularization network with lambda = 0
14 Network weights:
15 None
16
17 Regularization network with lambda = 1
18 Network weights:
19 [ 0.08464088  0.08464088 -0.01100928  0.23076148  0.04537817 -0.02095925
20    0.00799129  0.00651917  0.04220471  0.08025859  0.29364197  0.19406771
21   -0.22010325 -0.06933864 -0.0341914  0.00936621 -0.02110735 -0.00275415
22    0.20974423  0.15086966 -0.0295201  -0.00393459  0.04557795  0.19493987
23    0.00323406  0.24053468 -0.02871195 -0.06685137 -0.19795732 -0.02095925
24   -0.0104385  0.00799129 -0.02703709  0.00323406  0.00936621 -0.00679505
25   -0.05073412  0.43881737  0.03614517 -0.08935179  0.02662353  0.09334059
26    0.04537817  0.03614517  0.02662353 -0.11748128  0.23546815 -0.23254447
27    0.02662353 -0.12480445  0.00324975  0.02662353 -0.00679505 -0.01903425
28    0.00638078 -0.00562854 -0.03369634  0.00073208  0.03614517  0.01514873
29    0.18079406 -0.06570931  0.15424157  0.08464088 -0.23597608  0.04557795
30   -0.01432496  0.03614517  0.33891664  0.28512327  0.28500228  0.31870843
31   -0.1133878  -0.18336908 -0.01100928  0.00324975  0.28262545 -0.04125404
32   -0.03240205 -0.05746574]
33 Train set accuracy: 1.0
34 Test set accuracy: 1.0
35
36 Regularization network with lambda = 10
37 Network weights:
38 [ 0.04231009  0.04231009 -0.00154093  0.06314162  0.04215423 -0.0028434
39   -0.00102224 -0.002035  0.0391684  0.04836593  0.07143414  0.05884712
40   -0.02552513 -0.00815074 -0.00569997  0.03665439 -0.00332157 -0.00135473
41    0.05664605  0.05000344 -0.01047833 -0.00173178  0.04137863  0.0567882
42   -0.00169283  0.06077167 -0.00306922 -0.01081996 -0.02019772 -0.0028434
43   -0.00231132 -0.00102224 -0.00794573 -0.00169283  0.03665439 -0.00219812
44   -0.00407825  0.083858  0.0358594  -0.0094301  0.03618622  0.05426001
45    0.04215423  0.0358594  0.03618622 -0.0155238  0.0595087  -0.02601846
46    0.03618622 -0.0075975  -0.00146058  0.03618622 -0.00219812 -0.00435337
47   -0.00138288 -0.00221931 -0.00382392 -0.00165401  0.0358594  -0.00305613
48    0.0521601  -0.00557767  0.05686846  0.04231009 -0.02217698  0.04137863
49   -0.00248376  0.0358594  0.07871732  0.07368209  0.0757208  0.07806111
50   -0.00878878 -0.02497713 -0.00154093 -0.00146058  0.0633956  -0.0082386
51   -0.00772705 -0.01603764]
52 Train set accuracy: 0.7875
53 Test set accuracy: 0.85
54
55 Regularization network with lambda = 100
56 Network weights:
57 [ 8.78054065e-03  8.78054065e-03 -5.86614633e-05  9.29379767e-03
58    8.80154610e-03 -8.25267183e-05 -5.32464086e-05 -7.53649787e-05
59    8.72086877e-03  8.95884678e-03  9.48797122e-03  9.17874816e-03
60   -6.00540449e-04 -1.92005759e-04 -1.47687285e-04  8.66618401e-03
61   -9.12222380e-05 -5.43520333e-05  9.12105955e-03  8.95545402e-03
62   -2.87192108e-04 -6.47371144e-05  8.78262924e-03  9.13137902e-03
63   -6.63633465e-05  9.21352137e-03 -8.05267540e-05 -2.70175474e-04
64   -4.61429694e-04 -8.25267183e-05 -7.46674327e-05 -5.32464086e-05
65   -2.23854620e-04 -6.63633465e-05  8.66618401e-03 -7.13094379e-05
66   -9.70140483e-05  9.75055234e-03  8.62293282e-03 -2.08450430e-04
67    8.63629317e-03  9.11239273e-03  8.80154610e-03  8.62293282e-03
68    8.63629317e-03 -3.76806917e-04  9.17665725e-03 -5.96326486e-04
69    8.63629317e-03 -1.49043173e-04 -6.36492710e-05  8.63629317e-03

```

```

70 -7.13094379e-05 -1.17409474e-04 -6.26293912e-05 -7.54400569e-05
71 -9.85976650e-05 -6.44518374e-05 8.62293282e-03 -1.08831226e-04
72 9.00464460e-03 -1.24064822e-04 9.15801352e-03 8.78054065e-03
73 -4.97668198e-04 8.78262924e-03 -7.89825097e-05 8.62293282e-03
74 9.66279258e-03 9.54963176e-03 9.60738229e-03 9.65491693e-03
75 -1.84799705e-04 -6.04491489e-04 -5.86614633e-05 -6.36492710e-05
76 9.25440761e-03 -2.09548461e-04 -2.01557017e-04 -4.12789739e-04]
77 Train set accuracy: 0.55
78 Test set accuracy: 0.65
79
80 Regularization network with lambda = 1000
81 Network weights:
82 [ 9.86283533e-04 9.86283533e-04 -7.20038283e-07 9.92154255e-04
83 9.86555485e-04 -9.82184641e-07 -6.68374727e-07 -9.07906755e-07
84 9.85623128e-04 9.88358084e-04 9.94359781e-04 9.90823761e-04
85 -6.83493738e-06 -2.19190560e-06 -1.70499675e-06 9.85006058e-04
86 -1.07280332e-06 -6.65368654e-07 9.90164297e-04 9.88264270e-04
87 -3.32484365e-06 -7.82274073e-07 9.86339823e-04 9.90290243e-04
88 -8.12401601e-07 9.91207071e-04 -9.43614632e-07 -3.09794013e-06
89 -5.23601180e-06 -9.82184641e-07 -8.93610747e-07 -6.68374727e-07
90 -2.60126814e-06 -8.12401601e-07 9.85006058e-04 -8.59267682e-07
91 -1.12068589e-06 9.97305352e-04 9.84480667e-04 -2.35721908e-06
92 9.84639827e-04 9.90122260e-04 9.86555485e-04 9.84480667e-04
93 9.84639827e-04 -4.30628118e-06 9.90777795e-04 -6.76593204e-06
94 9.84639827e-04 -1.67000347e-06 -7.87474860e-07 9.84639827e-04
95 -8.59267682e-07 -1.36228968e-06 -7.72355235e-07 -9.07113589e-07
96 -1.14984059e-06 -7.86117309e-07 9.84480667e-04 -1.29504071e-06
97 9.88823160e-04 -1.41435636e-06 9.90622770e-04 9.86283533e-04
98 -5.63612241e-06 9.86339823e-04 -9.45251037e-07 9.84480667e-04
99 9.96351616e-04 9.95071382e-04 9.95740100e-04 9.96269706e-04
100 -2.08286195e-06 -6.90274234e-06 -7.20038283e-07 -7.87474860e-07
101 9.91644601e-04 -2.40900662e-06 -2.32382957e-06 -4.74232210e-06]
102 Train set accuracy: 0.55
103 Test set accuracy: 0.65
104
105
106 c)
107 RBF network with 14 neurons
108 Network weights:
109 [-0.6632162 -1.21975653 0.7091163 -0.74267526 -1.62010784 0.91190387
110 -1.57045037 1.89000413 1.39029724 1.26033234 1.39029724 -1.26443106
111 0.42571842 -0.06809647]
112 Train set accuracy: 0.975
113 Test set accuracy: 0.95
114
115
116 d)
117 RBF network with 25 neurons
118 Network weights:
119 [-0.12678128 -0.26634102 -0.88779964 -0.72297732 -0.27321834 2.2710429
120 0.97548542 1.26685796 0.35600438 0.2471123 0.35600438 0.74490074
121 1.56253934 -1.09281851 -1.11772519 1.180662 -1.42220868 1.62990986
122 1.01537072 -1.11726526 -0.95080604 -0.9898839 0.60437542 -1.41871108
123 -0.90343659]
124 Train set accuracy: 1.0
125 Test set accuracy: 1.0
126
127
128 e)
129 RBF network with 50 neurons

```



```

130 Network weights:
131 [ 0.40517746  0.14315587  0.35479075  0.08106163 -0.4525487  0.98818288
132 -0.49166515  1.23128663  0.09019109  0.51563672  0.09019109 -0.09300734
133 0.73992459 -0.3495039  1.17348993 -0.65279331 -0.28887568 0.3356492
134 0.3190445 -0.59074263 0.32550032 0.38957002 -0.04584974 -1.06309899
135 0.19970519 -0.14645848 0.32327589 0.05363561 -0.50110079 -0.28245281
136 -0.74315103 0.32550032 -0.47276697 -0.07397572 -0.99923099 0.40007796
137 -0.12428876 0.03408903 0.62699977 -0.50110079 0.38957002 -0.75142911
138 -0.09300734 0.47170145 0.40517746 -0.2018319 0.15027514 0.87487634
139 -0.77020923 -0.09967739]
140 Train set accuracy: 1.0
141 Test set accuracy: 1.0
142
143
144 f)
145 RBF network with 80 neurons
146 Network weights:
147 [-5.43225788e-02 -1.04657292e-01 -1.24836745e-02 -2.35444709e-03
148 -3.03466209e-01 5.09998675e-01 -4.50567659e-02 6.07310072e-01
149 -5.84283874e-02 3.76759183e-01 -5.84283874e-02 1.70910264e-01
150 -9.87811663e-02 -3.49405195e-01 5.52196906e-02 -1.43622636e-01
151 5.06768304e-01 1.55969232e-02 1.95620314e-02 -8.29262896e-02
152 -1.33728356e-01 6.16109240e-02 -2.63890442e-01 3.29809344e-01
153 -1.85766576e-04 1.37612983e-02 5.17851508e-01 3.47056040e-01
154 6.35544313e-02 -1.43341312e-01 -3.43581238e-01 -1.33728356e-01
155 1.11921144e-01 -3.49506235e-05 -5.81941158e-01 2.26611837e-01
156 9.11969687e-03 4.32166240e-01 6.09865467e-01 6.35544313e-02
157 6.16109240e-02 -1.73674325e-01 1.70910264e-01 -3.55191311e-02
158 -5.43225788e-02 -7.87907096e-02 3.20082251e-02 2.85356150e-01
159 -2.92527086e-02 -5.39100402e-01 1.11921144e-01 5.52196906e-02
160 5.03431336e-01 1.70910264e-01 6.78273794e-02 -1.43622636e-01
161 -2.92527086e-02 1.19209439e-01 8.35875529e-02 -5.84283874e-02
162 5.13791021e-02 -9.87811663e-02 1.74293964e-01 -1.72610297e-01
163 5.63562317e-01 1.03324512e+00 5.48150664e-01 -1.38029808e-01
164 1.55969232e-02 6.16109240e-02 -7.74502681e-01 3.46196163e-01
165 -1.92902453e-01 -4.80870205e-02 6.38470708e-02 6.04746564e-01
166 6.16109240e-02 -7.63876886e-01 -5.07316804e-01 -5.84283874e-02]
167 Train set accuracy: 1.0
168 Test set accuracy: 1.0

```

4.1 Item a

No item a, verifica-se que a matriz ϕ é singular, isto é, com determinante igual a zero. Como consequência, ela não é invertível, tornando o sistema de equações impossível de ser resolvido no caso sem que se utiliza a rede sem regularização.

4.2 Item b

No item b, há vários casos. Neles, quanto maior λ , maior a força da regularização aplicada à rede neural.

- $\lambda = 0$: equivalente ao caso do item a, com a rede sem regularização.
- $\lambda = 1$: nesse caso, tem-se regularização e, portanto, a matriz torna-se invertível, podendo-se assim resolver a equação matricial para a rede de regularização, conforme a Figura 1. Tanto no conjunto de treino quanto no de teste a acurácia foi de 100%.

- $\lambda = 10$: Analogamente, por se ter regularização, pode-se resolver a equação e determinar os pesos da rede. No entanto, como tem-se um λ maior, a maior regularização da rede faz com que ela tenha menor capacidade de aprendizado (isto é, não é capaz de aprender tão bem os padrões ocultos do conjunto de treinamento). Por consequência, ambas as acurácias de treino e de teste são menores.
- $\lambda = 100$: Com λ ainda maior, as acurácias diminuem ainda mais.
- $\lambda = 1000$: Finalmente, como λ já possui um valor bastante elevado, verifica-se que a acurácia no treino e no teste são iguais ao caso de $\lambda = 100$. Isso pode ser explicado pelo fato da regularização ter "saturado", ou seja, não adianta mais aumentar o valor de λ , pois a capacidade de aprendizado da rede não pode ser mais reduzida.

4.3 Item c

A partir do item c, trata-se da rede RBF, com quantidade de neurônios variável. No item c, em específico, utiliza-se uma camada RBF com 14 neurônios, gerando, portanto, um vetor de pesos de tamanho 14. As acurácias de treino e de teste não atingem 100%, mas são próximas a esse valor. Isso acontece porque não se tem um número elevado o suficiente de neurônios na camada RBF para que a rede consiga aprender os padrões ocultos do conjunto de treinamento.

4.4 Item d

Aumentando-se, no item d, o número de neurônios para 25, verifica-se que ambas as acurácias atingem 100%, conforme esperado. O vetor de pesos aumenta seu tamanho para 25, como previsto.

4.5 Item e

Aumentando-se mais ainda o número de neurônios, não se obtém melhorias nas acurácias, pois a rede já possuía a capacidade de aprendizado necessária com 25 neurônios. Na verdade, utilizar, por exemplo, 50 neurônios tende a levar a rede a um overfitting, afetando negativamente sua capacidade de generalização em outros conjuntos de dados.

4.6 Item f

Analogamente ao item e, agora com 80 neurônios, a rede continua com acurácias de treino e de teste máximas, mas agora com vetor de pesos de tamanho 80, conforme esperado.

5 Conclusão

Após a implementação realizada e a resolução das tarefas propostas, foi possível observar que a variação nos parâmetros de regularização λ e no número de neurônios na camada RBF influenciam de forma significativa o desempenho das redes neurais nos conjuntos de treino e de teste no problema de classificação entre democratas e republicanos. A inclusão do termo de regularização demonstrou-se eficaz para evitar problemas de sobreajuste, sendo capaz de estabilizar a matriz de pesos e melhorar a capacidade de generalização da rede, especialmente em valores intermediários de λ . Por outro lado, o aumento no número de neurônios na camada RBF contribuiu para a precisão da rede até certo ponto, após o qual observou-se um efeito

de saturação, onde mais neurônios já não traziam benefícios adicionais e, em alguns casos, conduziam a sobreajuste.

A análise dos resultados permite concluir que há um equilíbrio necessário entre a regularização e o dimensionamento da rede, dependendo da complexidade do problema e do tamanho do conjunto de dados. A metodologia aplicada e os experimentos realizados contribuíram para uma melhor compreensão sobre como otimizar o desempenho de redes RBF e de regularização em problemas de aprendizado, em especial no conjunto de dados de classificação entre democratas e republicanos.

6 Apêndice

6.1 Apêndice A

Utilizou-se a linguagem Python para a codificação do laboratório. O código se encontra no seguinte repositório:

- <https://github.com/SamirNunes/cmc-15-ia/tree/main/lab5>

Para rodá-lo, instale o poetry (<https://python-poetry.org/docs/>) e crie um environment por meio da instalação das dependências:

```
> poetry install
```

Em seguida, partindo da raiz, rode os comandos no terminal:

```
> cd src/lab5
> poetry run python main.py
```

6.2 Apêndice B

6.2.1 logger.py

```
1 import sys
2 from logging import INFO, StreamHandler, getLogger
3
4 logger = getLogger("lab5")
5 logger.setLevel(INFO)
6 logger.addHandler(StreamHandler(sys.stdout))
```

6.2.2 main.py

```
1 from lab5.tasks import a, b, c, d, e, f
2
3 if __name__ == "__main__":
4     tasks = [a, b, c, d, e, f]
5     for task in tasks:
6         task()
```

6.2.3 rbf_net.py

```
1 import numpy as np
2 from scipy.spatial.distance import cdist
3 from sklearn.exceptions import NotFittedError
4
5
```

```

6 class RBFLayer:
7     def __init__(self, n_centers: int, random_state: int = 0):
8         self._rand = np.random.RandomState(random_state)
9         self._n_centers: int = n_centers
10        self._centers: np.ndarray = None
11        self._sigma: float = None
12        self._fitted = False
13
14        def fit(self, X: np.ndarray):
15            self._centers = self._calculate_centers(X)
16            self._sigma = self._calculate_sigma()
17            self._fitted = True
18            return self
19
20        def predict(self, X: np.ndarray):
21            if self._fitted:
22                return RBFLayer._gaussian_rbf(X, self._centers, self._sigma)
23            raise NotFittedError
24
25        def _calculate_centers(self, X: np.ndarray):
26            indices = self._rand.choice(X.shape[0], self._n_centers, replace=
False)
27            return X[indices]
28
29        def _calculate_sigma(self):
30            return np.max(cdist(self._centers, self._centers, "sqeuclidean")) /
np.sqrt(
31                2 * self._n_centers
32            )
33
34        @staticmethod
35        def _gaussian_rbf(X: np.ndarray, centers: np.ndarray, sigma: float):
36            return np.exp(-cdist(X, centers, "sqeuclidean") / (2 * sigma**2))
37
38
39 class RBFNetwork:
40     def __init__(self, n_neurons: int, random_state: int = 0):
41         self.n_neurons = n_neurons
42         self.random_state = random_state
43         self.layer = None
44         self.weights = None
45         self.fitted = False
46
47        def fit(self, X_train: np.ndarray, y_train: np.ndarray):
48            self.layer = RBFLayer(self.n_neurons, self.random_state).fit(X_train
)
49            phi: np.ndarray = self.layer.predict(X_train)
50            self.weights: np.ndarray = np.linalg.pinv(phi) @ y_train
51            self.fitted = True
52            return self
53
54        def predict(self, X: np.ndarray):
55            if self.fitted:
56                phi = self.layer.predict(X)
57                y_pred = phi @ self.weights
58                return (y_pred >= 0.5).astype(int)
59            raise NotFittedError

```

6.2.4 reg_net.py

```

1 from abc import ABC

```

```

2
3 import numpy as np
4 from lab5.logger import logger
5 from scipy.spatial.distance import cdist
6 from sklearn.exceptions import NotFittedError
7
8
9 class RegularizationLayer:
10     def __init__(self, random_state: int = 0):
11         self._rand = np.random.RandomState(random_state)
12         self._n_centers: int = None
13         self._centers: np.ndarray = None
14         self._sigma: float = None
15         self._fitted = False
16
17     def fit(self, X: np.ndarray):
18         self._n_centers = X.shape[0]
19         self._centers = self._calculate_centers(X)
20         self._sigma = self._calculate_sigma()
21         self._fitted = True
22         return self
23
24     def predict(self, X: np.ndarray):
25         if self._fitted:
26             return RegularizationLayer._gaussian_rbf(X, self._centers, self._sigma)
27         raise NotFittedError
28
29     def _calculate_centers(self, X: np.ndarray):
30         return X
31
32     def _calculate_sigma(self):
33         return np.max(cdist(self._centers, self._centers, "sqeuclidean")) /
34         np.sqrt(
35             2 * self._n_centers
36         )
37
38     @staticmethod
39     def _gaussian_rbf(X: np.ndarray, centers: np.ndarray, sigma: float):
40         return np.exp(-cdist(X, centers, "sqeuclidean") / (2 * sigma**2))
41
42 class BaseNetwork(ABC):
43     def __init__(self, random_state: int = 0):
44         self.random_state: int = random_state
45         self.layer: RegularizationLayer = None
46         self.weights: np.ndarray = None
47         self.fitted: bool = False
48
49     def predict(self, X: np.ndarray):
50         if self.fitted:
51             phi = self.layer.predict(X)
52             y_pred = phi @ self.weights
53             return (y_pred >= 0.5).astype(int)
54         raise NotFittedError
55
56
57 class NoRegularizationNetwork(BaseNetwork):
58     def __init__(self, random_state: int = 0):
59         super().__init__(random_state)

```

```

60
61     def fit(self, X_train: np.ndarray, y_train: np.ndarray):
62         self.layer = RegularizationLayer(self.random_state).fit(X_train)
63         phi: np.ndarray = self.layer.predict(X_train)
64         try:
65             self.weights: np.ndarray = np.linalg.inv(phi) @ y_train
66             self.fitted = True
67             return self
68         except np.linalg.LinAlgError as e:
69             logger.error(
70                 f"Matrix phi is singular (determinant = 0), generating the
error: {e}.\nThis happens because there's no regularization (lambda = 0).
"
71             )
72
73
74 class RegularizationNetwork(BaseNetwork):
75     def __init__(self, lambda_param: int, random_state: int = 0):
76         super().__init__(random_state)
77         self.lambda_param = lambda_param
78
79     def fit(self, X_train: np.ndarray, y_train: np.ndarray):
80         self.layer = RegularizationLayer(self.random_state).fit(X_train)
81         G = self.layer.predict(X_train)
82         try:
83             self.weights = self._calculate_regularized_inverse(G) @ y_train
84             self.fitted = True
85             return self
86         except np.linalg.LinAlgError as e:
87             logger.error(
88                 f"Matrix phi is singular (determinant = 0), generating the
error: {e}.\nThis happens because there's no regularization (lambda = 0).
"
89             )
90
91     def _calculate_regularized_inverse(self, G: np.ndarray):
92         return np.linalg.inv(G + self.lambda_param * np.identity(G.shape[0])
)

```

6.2.5 tasks.py

```

1 from lab5.logger import logger
2 from lab5.rbf_net import RBFNetwork
3 from lab5.reg_net import NoRegularizationNetwork, RegularizationNetwork
4 from sklearn.metrics import accuracy_score
5 from utils import import_data
6
7 RANDOM_STATE = 0
8
9
10 def a():
11     logger.info("\na")
12
13     X_train, X_test, y_train, y_test = import_data()
14     reg_net = NoRegularizationNetwork(RANDOM_STATE)
15     reg_net.fit(X_train, y_train)
16
17     logger.info(
18         f'"Regularization" network with no regularization\nNetwork weights:\n{reg_net.weights}'
19     )

```

```

20
21     if reg_net.fitted:
22         y_train_pred = reg_net.predict(X_train)
23         logger.info(f"Train set accuracy: {accuracy_score(y_train,
24 y_train_pred)}")
25
26         y_test_pred = reg_net.predict(X_test)
27         logger.info(f"Test set accuracy: {accuracy_score(y_test, y_test_pred
28 )}\n")
29
30 def b():
31     logger.info("\nb")
32
33     X_train, X_test, y_train, y_test = import_data()
34     lambda_values = [0, 1, 10, 100, 1000]
35
36     for lambda_param in lambda_values:
37         reg_net = RegularizationNetwork(lambda_param, RANDOM_STATE)
38         reg_net.fit(X_train, y_train)
39         logger.info(
40             f"Regularization network with lambda = {lambda_param}\nNetwork
41 weights:\n{reg_net.weights}"
42 )
43
44         if reg_net.fitted:
45             y_train_pred = reg_net.predict(X_train)
46             logger.info(f"Train set accuracy: {accuracy_score(y_train,
47 y_train_pred)}")
48
49             y_test_pred = reg_net.predict(X_test)
50             logger.info(f"Test set accuracy: {accuracy_score(y_test,
51 y_test_pred)}\n")
52
53 def c():
54     logger.info("\nc")
55
56     X_train, X_test, y_train, y_test = import_data()
57     n_neurons = 14
58     rbf_net = RBFNetwork(n_neurons, RANDOM_STATE)
59     rbf_net.fit(X_train, y_train)
60
61     logger.info(
62         f"RBF network with {n_neurons} neurons\nNetwork weights:\n{rbf_net.
63 weights}"
64 )
65
66     y_train_pred = rbf_net.predict(X_train)
67     logger.info(f"Train set accuracy: {accuracy_score(y_train, y_train_pred)
68 }")
69
70     y_test_pred = rbf_net.predict(X_test)
71     logger.info(f"Test set accuracy: {accuracy_score(y_test, y_test_pred)}\n

```

```

72 X_train, X_test, y_train, y_test = import_data()
73 n_neurons = 25
74 rbf_net = RBFNetwork(n_neurons, RANDOM_STATE)
75 rbf_net.fit(X_train, y_train)
76
77 logger.info(
78     f"RBF network with {n_neurons} neurons\nNetwork weights:\n{rbf_net.
weights}"
79 )
80
81 y_train_pred = rbf_net.predict(X_train)
82 logger.info(f"Train set accuracy: {accuracy_score(y_train, y_train_pred)
}")
83
84 y_test_pred = rbf_net.predict(X_test)
85 logger.info(f"Test set accuracy: {accuracy_score(y_test, y_test_pred)}\n
")
86
87
88 def e():
89     logger.info("\ne")
90
91     X_train, X_test, y_train, y_test = import_data()
92     n_neurons = 50
93     rbf_net = RBFNetwork(n_neurons, RANDOM_STATE)
94     rbf_net.fit(X_train, y_train)
95
96     logger.info(
97         f"RBF network with {n_neurons} neurons\nNetwork weights:\n{rbf_net.
weights}"
98     )
99
100     y_train_pred = rbf_net.predict(X_train)
101     logger.info(f"Train set accuracy: {accuracy_score(y_train, y_train_pred)
}")
102
103     y_test_pred = rbf_net.predict(X_test)
104     logger.info(f"Test set accuracy: {accuracy_score(y_test, y_test_pred)}\n
")
105
106
107 def f():
108     logger.info("\nf")
109
110     X_train, X_test, y_train, y_test = import_data()
111     n_neurons = 80
112     rbf_net = RBFNetwork(n_neurons, RANDOM_STATE)
113     rbf_net.fit(X_train, y_train)
114
115     logger.info(
116         f"RBF network with {n_neurons} neurons\nNetwork weights:\n{rbf_net.
weights}"
117     )
118
119     y_train_pred = rbf_net.predict(X_train)
120     logger.info(f"Train set accuracy: {accuracy_score(y_train, y_train_pred)
}")
121
122     y_test_pred = rbf_net.predict(X_test)
123     logger.info(f"Test set accuracy: {accuracy_score(y_test, y_test_pred)}\n
")

```



```
)
```

6.2.6 utils.py

```
1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3
4
5 def import_data():
6     data = pd.read_csv("../data/data.csv")
7     data = data.replace(
8         {"n": "0", "y": "1", "democrat": "0", "republican": "1"}
9     ).astype(int)
10    X = data.drop(["Alvo"], axis=1)
11    y = data["Alvo"]
12    return _split_data(X, y)
13
14
15 def _split_data(X: pd.DataFrame, y: pd.DataFrame):
16     X_train, X_test, y_train, y_test = train_test_split(
17         X, y, test_size=0.2, random_state=0
18     )
19     return X_train.values, X_test.values, y_train.values, y_test.values
```