

# Relatório do Problema de Classificação Binária

## (a) Sorteio dos valores de $\epsilon_n$

- Valores de epsilon\_n sorteados: [0.37454012 0.95071431 0.73199394]

## (b)(i, ii) Geração e informação dos pontos $(\vec{x}_n, y(f(\vec{x}_n)))$

- $\vec{x}_n$ : [[ 5.91950905 -20.63888157 -20.64032878] [-26.51498327 21.97056875 6.0669007] [ 12.48435467 -28.76493034 28.19459113] ... [-25.66720888 11.06612166 0.19322489] [ 15.90893099 -0.88256196 -21.03710183] [ 8.89541724 -19.65681827 22.3436738 ]]
- $f(\vec{x}_n)$ : [7.75028721e+07 2.83573653e+05 5.03021363e+08 ... 1.56550243e+04 8.66795655e+07 1.24580062e+08]

Iremos estabelecer um threshold para que  $y(f(\vec{x}_n))$  consiga ter classes 0 e 1. Essa regra será a mediana, de tal forma que:

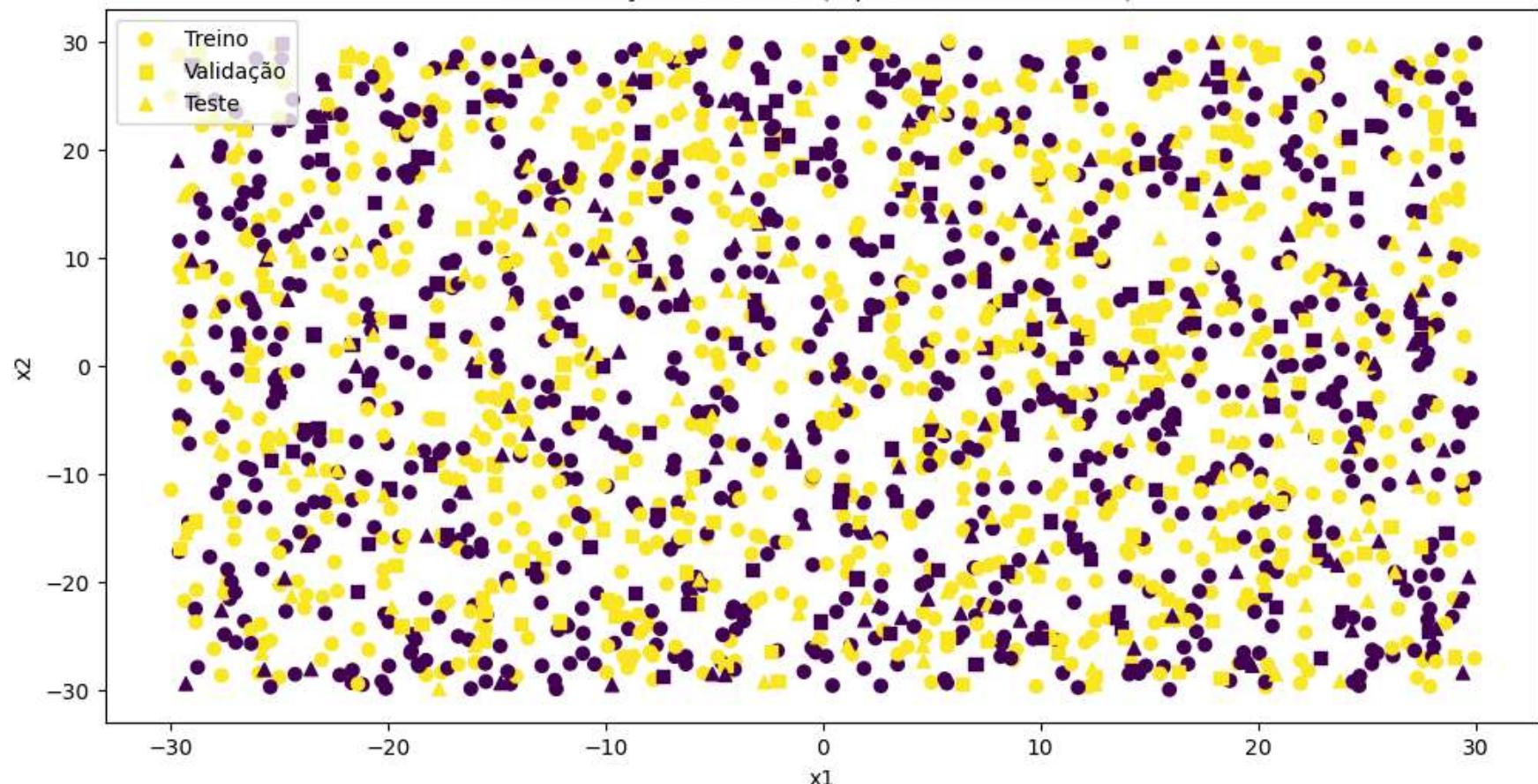
$$y(f(\vec{x}_n)) = \begin{cases} 1, & f(\vec{x}_n) \geqslant \text{Mediana}(\{f(\vec{x}_0), f(\vec{x}_1), \dots, f(\vec{x}_{1999})\}) \\ 0, & \text{caso não} \end{cases}$$

Assim, binarizamos o problema em um classificador com classes 0 e 1.

- $y(f(\vec{x}_n))$ : [1 0 1 ... 0 1 1]

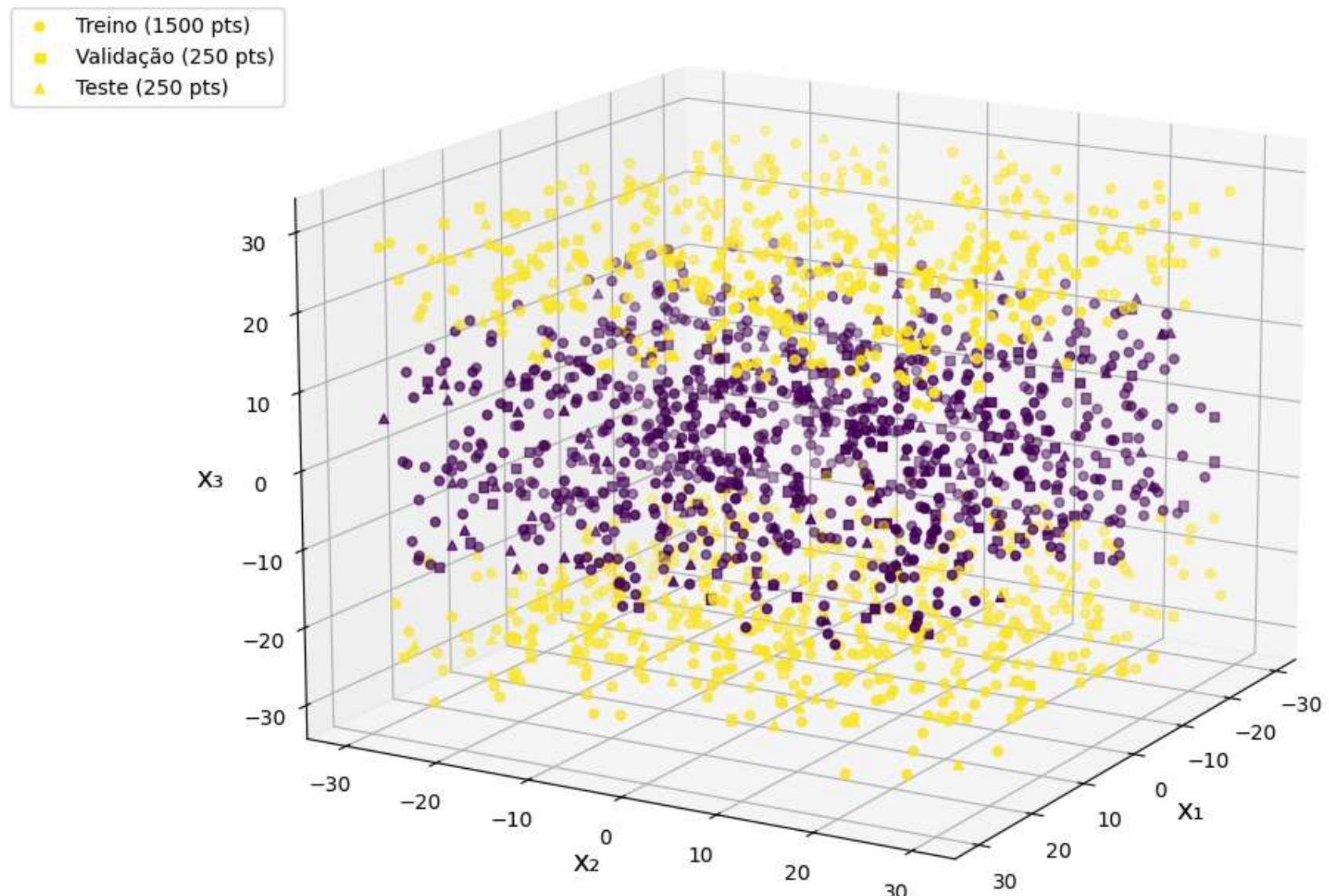
## (b)(iii) Visualização dos pontos $(\vec{x}_n, y(f(\vec{x}_n)))$ em 2D

## Visualização dos Dados (2 primeiras dimensões)



(b)(iii) Visualização dos pontos  $(\vec{x}_n, y(f(\vec{x}_n)))$  em 3D

## Visualização 3D dos Dados Gerados com as classes 0 e 1 como cor



## Métricas Escolhidas para avaliação dos modelos

Como estamos tratando de um problema de classificação binária, temos que ajustar as métricas e os modelos para atuarem como classificadores.

- Modelos:
  - MLP-BP de Três Camadas: usaremos o [MLPRegressor da Scikit-Learn](#)
  - RBF de Três Camadas: construí uma rotina própria de RBF com k-means
  - SVM com Kernel: usaremos o [SVR da Scikit-Learn](#)
- Métricas:
  - Mean Squared Error: usaremos o [MSE da Scikit-Learn](#) para analisar o quanto o modelo está errando
  - Acurácia: usaremos o [accuracy\\_score da Scikit-Learn](#) para analisar a porcentagem de acertos dos modelos

## Normalização dos Dados

Fazemos isso para que os valores numéricos não estourem. Para tal, usamos o [StandardScaler](#) que atua como um z-score:

$$z = \frac{x - \mu}{\sigma}$$

## (c) MLP-BP de três camadas

Implementei um MLP de 3 camadas em que:

- Camada escondida contém apenas 20 neurônios;
- Função de ativação para o cálculo do Net é a 'ReLU' ≡ Rectified Linear Unit Function;
- Regra de Atualização dos Pesos: 'adam';
- Número Máximo de Iterações: 1000;
- Resultados do MLP-BP de três camadas:

- Accuracy - Treino: 0.996 | Validação: 0.996 | Teste: 1.0
- MSE - Treino: 0.004 | Validação: 0.004 | Teste: 0.0
- Pesos Finais Utilizados: [array([-1.49449615e-02, 2.07447635e-02, 5.54207655e-01, 3.14050960e-01, -5.57145537e-01, 1.09665687e-01, -3.18957546e-02, 4.11631227e-04, 2.42979570e-83, -3.03305110e-02, -8.77204735e-01, 1.66063964e-02, -8.83890258e-03, -4.36595105e-02, -7.12597133e-02, -3.12212500e-02, -2.34073103e-02, -2.27746909e-02, -9.82814102e-02, -1.50420632e-01], [5.32210428e-02, 5.02498127e-02, -3.73592062e-01, -5.12913958e-01, -5.56080500e-01, 8.77239538e-01, 5.52923645e-02, 4.83143488e-02, -1.30009027e-84, 5.16791968e-02, 1.79156023e-01, 2.11675151e-02, 1.83060280e-03, 4.36165033e-02, 1.16407070e-01, 2.01647892e-02, 6.33771188e-02, 5.76858778e-02, 6.45646764e-01, -1.29234143e-02], [-1.48857655e+00, -2.19913530e+00, -2.63334256e-01, 2.11427286e-02, -5.57462870e-01, -1.30646795e-01, -1.86829877e+00, -1.66824860e+00, -4.62656363e-93, -2.36252874e+00, -1.46026354e-01, 2.01489643e+00, 1.98625006e+00, 2.24674987e+00, 1.74005138e+00, 2.42120906e+00, -2.06031724e+00, -2.27502248e+00, -1.15382649e-01, -1.19774493e-01]], array([1.66573072e+00, [2.14861245e+00], [-1.30217813e+00], [-9.54611642e-01], [-7.78432137e-01], [-1.32111253e+00], [2.19579496e+00], [1.52104656e+00], [2.54879449e-27], [1.54630585e+00], [-1.22191531e+00], [2.30134466e+00], [2.16744243e+00], [1.60972848e+00], [1.55037131e+00], [1.64631088e+00], [1.36784806e+00], [2.11766008e+00], [-3.34216632e-01], [-3.73624069e-01]]])]
- Biases Finais Utilizados: [array([-0.65178449, -0.87803386, 2.15579083, 1.00264065, 0.74006954, 1.55711706, -0.78994953, -0.61730583, -0.43459985, -0.83325752, 1.54435811, -1.00608172, -0.98750568, -0.88911274, -0.66493495, -1.04027693, -0.80246429, -0.89984683, 0.18112299, -0.4495661]), array([-1.34811352])]

## (d) RBF de três camadas

Construí uma RBF de forma que:

- Usa K-means para encontrar centros representativos nos dados em que  $K = 50$  centros;
- Calcula a ativação de cada neurônio RBF usando função Gaussiana:  $\varphi(\|\vec{x}_n - c_i\|) = \exp(-\|\vec{x}_n - c_i\|^2/(2\sigma^2))$ , o que faz com que cada neurônio responda mais intensamente a pontos próximos ao seu centro;
- Camada de saída com combinação linear das ativações RBF;
- Usa pseudoinversa para resolver o sistema linear (mais estável que inversão direta);
- Resultados do RBF de três camadas:

- Accuracy - Treino: 0.7166666666666667 | Validação: 0.696 | Teste: 0.732
- MSE - Treino: 0.2833333333333333 | Validação: 0.304 | Teste: 0.268
- Centros finais utilizados: [[ 0.46057846 0.80667904 -0.48209908] [-1.12096508 -0.50048307 1.36988296] [ 0.55209187 0.25296892 1.35018956] [-0.97716343 -0.23870371 -1.37668172] [ 0.44629816 -1.12993786 -0.93862846] [-1.39797151 1.01527053 0.3170208] [ 0.17727225 -1.2814488 0.74570554] [ 0.63036203 -0.38260551 0.29072563] [-0.66592946 0.87629654 -0.12471984] [-1.33550674 -1.43748992 -0.05658016] [-1.39657733 1.15400183 -1.3318278] [ 1.22895625 1.29348949 0.81496228] [ 1.26485206 0.19411633 -1.11954386] [ 1.34691819 -0.81981882 -0.0292166] [-0.35147353 1.03619091 1.47464359] [-1.05242236 -1.3740765 0.59329821] [-1.36420467 -1.08527224 -1.27206692] [-0.20335398 -0.04853397 -0.28487743] [-0.55933835 1.11385022 -1.33220225] [ 1.3183862 1.41131188 -0.21811388] [ 0.4846463 -0.66474595 1.34084913] [-1.30422189 -0.53611684 -0.16294738] [-1.3336035 0.3813164 -0.6792554] [ 1.28151367 1.23657816 -1.17597802] [ 1.20627435 -1.43333979 -0.06327635] [ 0.19361677 1.50641172 0.08969605] [ 1.18043592 -1.00416369 -1.27757327] [ 0.28272739 0.04678045 -1.37124074] [-0.13629348 -1.17720707 0.01713341] [ 1.40436821 -0.03219523 -0.28739446] [-0.19598484 1.64606164 -1.21414266] [ 1.14146745 -1.35083632 1.25992219] [-0.43185477 -1.33891155 -1.39596256] [-1.45297857 -1.17696623 1.35046319] [-1.15104237 1.41016413 -0.58560968] [ 0.58313732 -0.21055254 -0.60946469] [-0.30018078 -0.27949251 1.07632484] [-0.70783101 -1.1792492 -0.74705196] [-1.45697297 0.40331522 1.21990172] [ 1.43408837 0.50653655 1.40248075] [-1.2865405 1.40003685 1.35583463] [-0.32846925 0.46561854 -0.98674189] [ 1.36083124 0.3166733 0.48004559] [-0.80682908 -0.10607417 0.48862153] [ 0.64964423 1.27276724 1.38282704] [ 0.18852653 0.65797224 0.61043924] [ 1.2953145 -0.60951677 0.84539558] [ 0.52215632 1.17748815 -1.23827974] [-0.26761052 -1.11587013 1.46214598] [-0.703207 1.22862954 0.78430594]]]

## (e) SVM com Kernel

Implementei um SVM Regressor (SVR) de tal forma que:

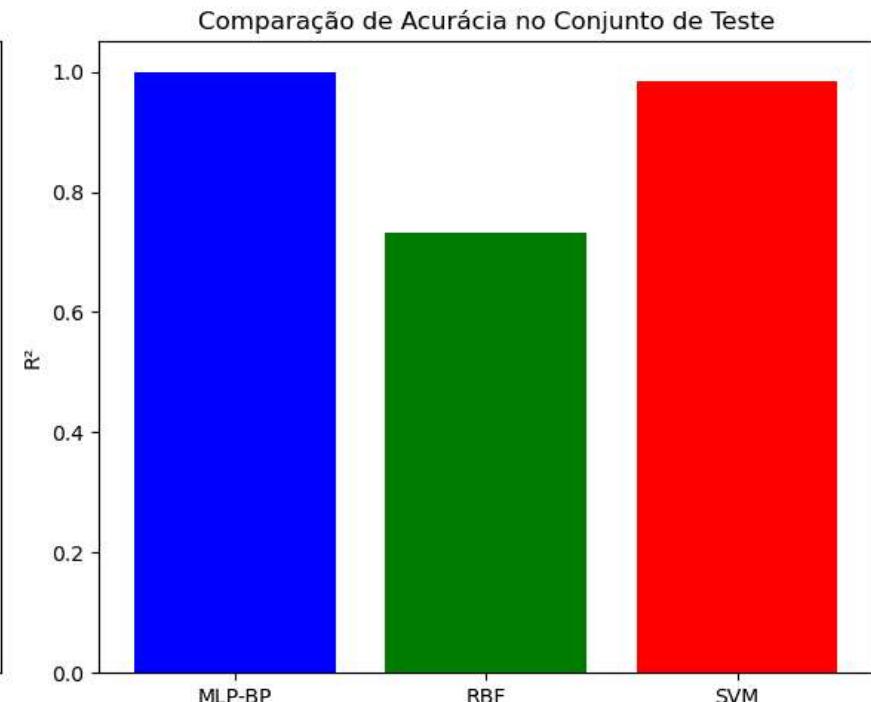
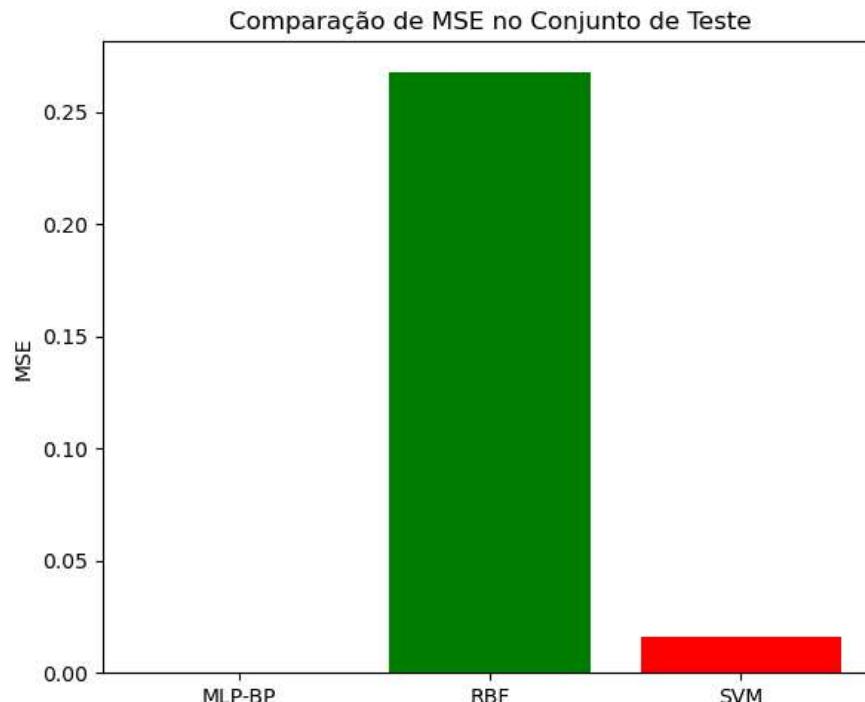
- Kernel do SVR é RBF;
  - Kernel Gaussiano:  $\mathbf{K}(x, x') = \exp(-\gamma * \|x - x'\|^2)$ 
    - $\gamma$ : controla a "largura";
    - $\|x - x'\|^2$ : é a distância euclidiana ao quadrado.
  - Com essa Transformação de Núcleo, conseguimos mapear o problema para um espaço de dimensão superior e fazer a regressão
- gamma: Controla o alcance da influência de cada ponto de treinamento
  - Valores baixos: decisão mais suave (considera pontos mais distantes);

- Valores altos: decisão mais complexa (considera pontos mais próximos);
- Resultados do SVM com Kernel:
  - Accuracy - Treino: 0.986 | Validação: 0.992 | Teste: 0.984
  - MSE - Treino: 0.014 | Validação: 0.008 | Teste: 0.016
  - Vetores de Suporte finais utilizados:
    - Índices [ 1 4 5 21 24 61 68 77 84 93 101 114 127 138 148 156 166 168 174 185 189 191 210 213 238 244 247 261 269 276 278 283 289 292 297 298 305 318 320 330 337 340 350 357 367 368 394 398 403 405 408 411 420 423 436 437 439 440 444 447 452 453 475 479 480 500 511 520 524 530 543 561 567 569 570 574 576 577 581 585 588 590 592 620 627 633 637 639 642 651 656 658 678 685 709 721 738 745 754 756 765 769 775 790 815 816 817 819 825 827 836 840 868 877 883 889 904 905 937 940 966 969 970 974 989 993 1002 1005 1021 1032 1060 1071 1074 1090 1111 1114 1140 1149 1152 1169 1170 1201 1223 1225 1229 1241 1245 1250 1251 1267 1270 1282 1298 1302 1303 1307 1319 1323 1326 1329 1341 1342 1363 1370 1378 1381 1383 1386 1388 1394 1402 1404 1411 1412 1418 1424 1425 1426 1429 1431 1437 1439 1443 1456 1474 1484 1485 1488 1489 1492 1499 14 16 34 43 54 56 79 82 92 129 133 137 143 146 149 170 182 184 201 208 212 220 223 239 249 255 257 263 279 290 293 321 325 328 333 335 339 345 356 360 384 387 389 395 399 407 410 412 431 438 443 446 448 455 472 474 476 477 478 485 489 491 492 498 504 505 506 522 523 539 541 566 568 572 579 584 593 596 599 603 609 618 632 643 644 648 654 657 659 672 681 711 720 724 727 746 749 751 755 761 762 774 782 809 811 812 813 820 831 835 837 839 849 862 865 870 876 881 882 887 890 908 909 914 915 919 920 929 933 950 954 960 982 984 987 991 994 1006 1008 1012 1015 1017 1022 1043 1046 1057 1068 1069 1070 1072 1085 1094 1098 1100 1110 1117 1129 1142 1162 1174 1185 1190 1192 1199 1208 1210 1212 1215 1230 1236 1243 1255 1257 1263 1276 1289 1299 1317 1348 1358 1403 1410 1413 1417 1421 1430 1433 1461 1469 1471 1478 1483]
    - Vetores de Suporte [[ 1.11557204 0.98519407 0.77387656] [ 0.13695967 -1.13327575 0.71300274] [ 1.17424398 -0.15939722 0.77331593] ... [ 0.31896098 1.72590156 0.98506006] [-1.1951046 0.27001591 1.11024264] [-0.83611845 0.57638212 1.14331788]]]
    - Número de Vetores de Suporte: 383

## (f) Comparação dos resultados no conjunto de teste

- MLP-BP:

- MSE Teste: 0.0000 | Acc Teste: 1.0000
- RBF:
  - MSE Teste: 0.2680 | Acc Teste: 0.7320
- SVM:
  - MSE Teste: 0.0160 | Acc Teste: 0.9840



## Considerações Finais

1. O MLP-BP apresentou o melhor desempenho geral, com menor MSE e maior Acurácia. Isso era esperado, pois redes neurais multcamadas são aproximadores universais de funções. Seu principal destaque vai para a rápida convergência do modelo que com menos robustez que os outros modelos (converge com menos de 1000 iterações e possui apenas 20 neurônios na camada oculta), consegue ter um desempenho superior aos outros dois. Contudo, conseguiríamos ajustar os outros modelos para que desempenhassem de forma parecida (ponto 4. deste breve relatório), porém aumentando suas complexidades.

2. O RBF teve um "pior" desempenho. Ele funciona bem para problemas com características radialmente simétricas, mas pode ter dificuldade com funções mais complexas.
3. O SVM com kernel RBF teve o segundo melhor desempenho neste problema. SVMs são poderosos para classificação binária e, como binarizamos as classes, teve um excelente desempenho.
4. Pelos desempenhos nas etapas de validação, os modelos não me parecem ter perdido a capacidade de generalização.
5. Há modos de aumentar as respectivas acurárias (aqui, vistas pela acurácia). O problema passa a ser analisar se não há overfitting dos modelos ao aumentar seus desempenhos:
  - MLP-BP: não há melhoria aparente na minha configuração
  - RBF: Aumentar o número de centros na região de decisão
  - SVM com Kernel RBF: Aumentar o parâmetro gamma que controla o alcance da influência dos pontos no problema
6. A função proposta parece ser melhor aproximada pela arquitetura do MLP-BP.