

**UNIVERSIDADE FEDERAL DE MINAS GERAIS  
ESCOLA DE ENGENHARIA  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

**REDES NEURAIS ARTIFICIAIS  
ATIVIDADES 1 e 3 - PARTE 2: Problema Não-  
Linearmente Separável e Aproximação Polinomial**

**Aluna:** Priscila Aparecida Dias Nicácio

**Professores:** Prof. Antônio Braga e Prof. Frederico Gualberto.

Belo Horizonte, Agosto de 2025

## Sumário

1. INTRODUÇÃO.....	3
2. FUNDAMENTAÇÃO TEÓRICA .....	3
3. RESOLUÇÃO DA ATIVIDADE 1 - PARTE 2.....	5
3.1 CÓDIGO IMPLEMENTADO.....	6
3.2 ANÁLISE DOS RESULTADOS.....	7
3.3 RESOLUÇÃO DA ATIVIDADE 3 - PARTE 2 .....	8
3.4 RESULTADOS .....	9
3.5 CÓDIGO IMPLEMENTADO PARA 100 AMOSTRAS .....	12
3.6 RESULTADOS .....	14
3.7 ANÁLISE DOS RESULTADOS.....	17
4. CONCLUSÃO.....	17
5. REFERÊNCIAS .....	18

## 1. INTRODUÇÃO

A classificação de padrões e a aproximação de funções são problemas centrais em aprendizado de máquina e modelagem matemática. Em muitos casos, os dados não são linearmente separáveis, não é possível traçar uma linha reta (ou hiperplano em dimensões superiores) que separe perfeitamente as classes.

Problemas não-linearmente separáveis exigem técnicas que transformem os dados em um espaço no qual se tornem linearmente separáveis (BISHOP, 2006; HASTIE; TIBSHIRANI; FRIEDMAN, 2009; DUDA; HART; STORK, 2001).

Neste estudo, há dois casos complementares: Classificação não-linearmente separável: pontos distribuídos em um padrão circular, com uma classe localizada dentro do círculo (classe 0) e outra fora dele (classe 1).

Aplicou-se projeção não-linear baseada na distância ao centro,  $z = \sqrt{x^2 + y^2}$ , transformando o problema original em um problema linearmente separável em uma dimensão, permitindo o uso de métodos lineares de classificação.

Na aproximação polinomial houve a modelagem de uma função geradora  $f_g(x)$  a partir de amostras  $D = \{(x_i, y_i)\}_{i=1}^N$ .

Um polinômio de grau  $p$  é representado por:  $p(x) = w_p x^p + w_{p-1} x^{p-1} + \dots + w_1 x + w_0$  onde  $x$  é a variável de entrada e  $w_1$  os coeficientes do polinômio.

O objetivo é determinar  $w_1$  de forma que  $p(x) \approx y_i$  para todos os pontos de  $D$ , minimizando o erro quadrático:  $E(\omega) = \sum_{i=1}^N (y_i - p(x_i))^2$  ou, equivalentemente, resolvendo o sistema matricial:  $H\omega = y$  onde  $H$  é a matriz de Vandermonde,  $\omega$  o vetor de coeficientes e  $y$  o vetor de observações.

A solução que minimiza o erro quadrático é obtida usando a pseudoinversa  $H^+$ :  $\omega = H^+ y$ .

Apresentou-se o uso de projeção não-linear para linearizar um problema de classificação circular, bem como o ajuste polinomial de uma função quadrática com ruído gaussiano, variando o grau do polinômio de 1 a 8 e o número de amostras (20 e 100). Também foram analisados os fenômenos de underfitting e overfitting, relacionando-os ao grau do polinômio e à quantidade de dados.

## 2. FUNDAMENTAÇÃO TEÓRICA

Os classificadores lineares, como regressão logística ou perceptrons, funcionam bem quando os dados podem ser separados por uma linha ou hiperplano.

Quando as classes apresentam distribuições complexas, a separação linear direta não é possível, exigindo transformações não-lineares.

Para o problema circular, aplicamos a função:  $z = \sqrt{x^2 + y^2}$  que mede a distância de cada ponto ao centro do círculo. Esta transformação projeta os dados em um espaço 1D, permitindo que a separação entre classes seja realizada por um simples limiar.

Técnicas como o kernel trick em SVMs utilizam o mesmo princípio, projetando dados em espaços de maior dimensão para torná-los linearmente separáveis.

A aplicação de funções não-lineares simples, como a distância ao centro de um círculo, permite que problemas complexos se tornem lineares (SCHÖLKOPF; SMOLA, 2002).

A projeção não-linear evidencia claramente a separação das classes, demonstrando como funções simples podem resolver problemas complexos. Para o ajuste polinomial, a matriz H (Matriz de Vandermonde H) é construída com os termos  $x_i^j$  do polinômio até grau p, permitindo que o ajuste linear via mínimos quadrados encontre os coeficientes que melhor aproximam a função geradora.

A solução usando a pseudoinversa  $H^+$  fornece a minimização do erro quadrático mesmo em sistemas sobredeterminados ou singulares (GOLUB; VAN LOAN, 2013; NOCEDAL; WRIGHT, 2006).

$$H = \begin{bmatrix} x_1^p & x_1^{p-1} & \dots & x_1 & 1 \\ x_2^p & x_2^{p-1} & \dots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_N^p & x_N^{p-1} & \dots & x_N & 1 \end{bmatrix}$$

Cada coluna representa uma dimensão do espaço polinomial, permitindo que o ajuste linear via mínimos quadrados encontre os coeficientes  $\mathcal{W}$  que melhor aproximam a função geradora.

A pseudoinversa  $H^+$  fornece a solução de mínimos quadrados mesmo quando o sistema é sobredeterminado ( $N > p + 1$ ) ou singular:  $\mathcal{W} = H^+$  garantindo a minimização da soma dos erros quadráticos entre observações e previsões.

Underfitting ocorre quando o grau do polinômio é baixo, impedindo que ele capture a tendência da função. Overfitting ocorre quando o grau do polinômio é alto, fazendo-o seguir o ruído das amostras.

A quantidade de amostras influencia diretamente esses fenômenos, sendo que mais pontos reduzem a sensibilidade ao ruído (HASTIE; TIBSHIRANI; FRIEDMAN, 2009; MONTGOMERY; PECK; VINING, 2012). Bom ajuste: polinômio que representa corretamente a função subjacente sem seguir o ruído.

O número de amostras N influencia diretamente esses fenômenos: mais pontos reduzem a sensibilidade ao ruído, diminuindo o overfitting.

### 3. RESOLUÇÃO DA ATIVIDADE 1 - PARTE 2

A atividade consiste em entender e aplicar projeções não-lineares para transformar um problema de classificação não-linear em linearmente separável. Deve-se analisar um conjunto de pontos de duas classes (vermelha e preta) e escolher uma ou duas funções não-lineares (como funções radiais, circulares ou combinações de retas) que permitam separar claramente as duas classes. Em seguida, implementar essa projeção, gerar o gráfico da superfície de separação no espaço original e mostrar também os pontos projetados no novo espaço, destacando como a transformação torna o problema linearmente separável.

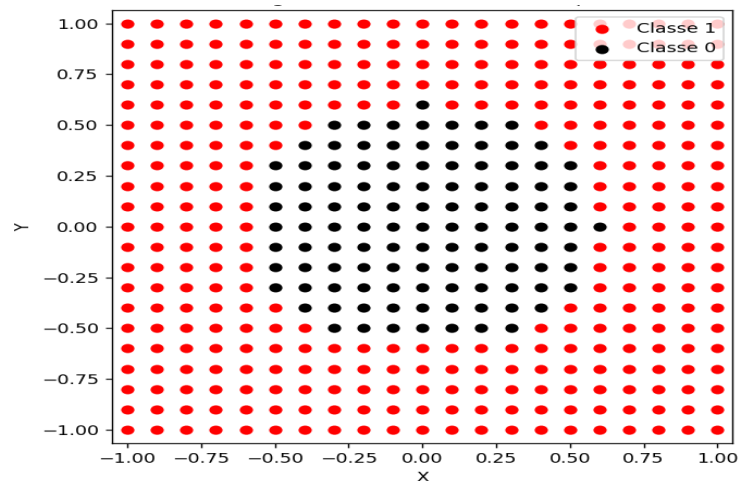


Figura 1 – Dados originais (não-linearmente separáveis), no espaço original os pontos em vermelho pertencem à classe positiva e pontos em preto pertencem à classe negativa.

```
x = seq(-1,1,by = 0.1)
y = seq(-1,1,by = 0.1)
create_grid <- expand.grid(x,y)

circle <- function(x,y) {
  return(sqrt(x^2+y^2))
}

raio = 0.6

classe = 1*(circle(create_grid$Var1,create_grid$Var2)>raio)
```

Figura 2 – Rotina que gerou os dados gerados em linguagem R.

Essa atividade contém um problema clássico de não-linearmente separável: os pontos pretos estão dentro de um círculo e os vermelhos fora. Para resolver, podemos criar uma projeção não-linear, por exemplo usando  $z = \sqrt{x^2 + y^2}$ , que transforma o problema circular em um problema linearmente separável em 1D.

### 3.1 CÓDIGO IMPLEMENTADO

```
import numpy as np

import matplotlib.pyplot as plt

# Criando os dados
x = np.arange(-1, 1.1, 0.1)
y = np.arange(-1, 1.1, 0.1)
X, Y = np.meshgrid(x, y)
X_flat = X.ravel()
Y_flat = Y.ravel()

# Função círculo
def circle(x, y):
    return np.sqrt(x**2 + y**2)

raio = 0.6
classe = np.where(circle(X_flat, Y_flat) > raio, 1, 0) # 1 = vermelho, 0 = preto

# Visualizando os dados originais
plt.figure(figsize=(6,6))
plt.scatter(X_flat[classe==1], Y_flat[classe==1], color='red', label='Classe 1')
plt.scatter(X_flat[classe==0], Y_flat[classe==0], color='black', label='Classe 0')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Dados originais (não-linearmente separáveis)')
plt.legend()
plt.axis('equal')
plt.show()

# Projeção não-linear
Z = circle(X_flat, Y_flat) # transf. os dados em 1D usando a distância ao centro

# Visualizando os dados projetados
plt.figure(figsize=(8,4))
plt.scatter(Z[classe==1], np.zeros_like(Z[classe==1]), color='red', label='Classe 1')
plt.scatter(Z[classe==0], np.zeros_like(Z[classe==0]), color='black', label='Classe 0')
plt.axvline(x=raio, color='blue', linestyle='--', label='Frente de separação linear')
```

```
plt.xlabel('z = sqrt(x^2 + y^2)')
plt.title('Dados projetados (linearmente separáveis)')
plt.legend()
plt.show()
```

O código implementa a separação de um problema não-linearmente separável através de uma projeção simples. Primeiro, é criado um grid bidimensional para x e y, sobre o qual se define a função circular  $z = \sqrt{x^2 + y^2}$ , que mede a distância dos pontos ao centro. A partir disso, pontos dentro do círculo ( $z < 0.6$ ) são classificados como classe 0 (pretos) e os fora como classe 1 (vermelhos). A transformação para o espaço unidimensional z torna o problema linearmente separável, já que a divisão das classes pode ser feita apenas com um limiar em  $z = 0.6$ . A visualização final mostra claramente a eficácia dessa projeção não-linear, revelando como uma fronteira circular no plano original se transforma em uma separação linear simples no espaço projetado.

### 3.2 ANÁLISE DOS RESULTADOS

O gráfico gerado mostra claramente a diferença entre os espaços antes e depois da projeção:

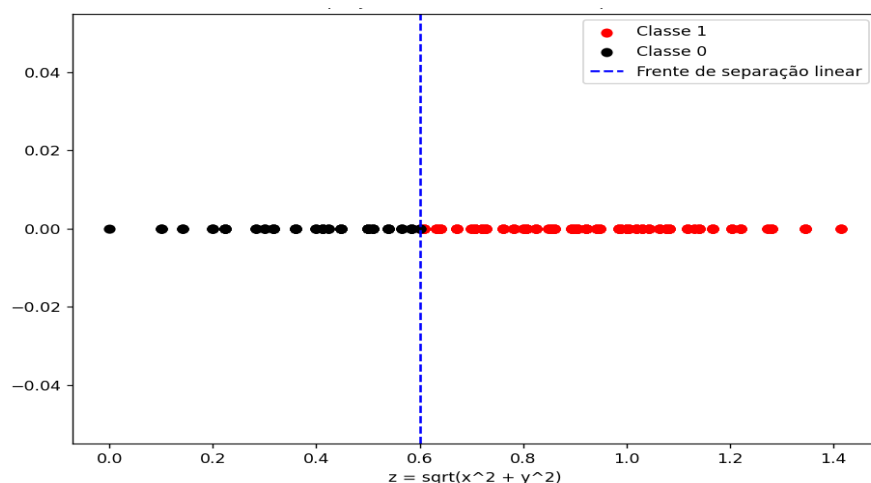


Figura 3 – Dados projetados (linearmente separáveis).

1. Espaço original (2D): os pontos da classe 0 (preto) estão agrupados no centro, enquanto os pontos da classe 1 (vermelho) estão distribuídos ao redor. Não existe uma linha reta que consiga separar as duas classes, caracterizando o problema como não-linearmente separável.

2. Espaço projetado (1D): após aplicar a projeção  $z = \sqrt{x^2 + y^2}$ , os dados foram transformados em uma dimensão, onde os pontos da classe 0 estão à esquerda do limiar  $z = 0.6$  e os pontos da classe 1 estão à direita. Com isso, a separação linear se torna possível e visualmente evidente, demonstrando a eficácia da transformação.

Este resultado confirma que, para problemas com estrutura circular, a distância ao centro é uma característica discriminativa eficiente e pode ser utilizada para construir classificadores lineares simples, evitando complexidade desnecessária.

### 3.3 RESOLUÇÃO DA ATIVIDADE 3 - PARTE 2

A tarefa de aproximação polinomial consiste em gerar polinômios que estimem uma função geradora a partir de um conjunto de amostras ruidosas. Deve-se criar conjuntos de dados com no mínimo 10 amostras e depois ampliar para 100 amostras da função  $fg(x) = 0.5x^2 + 3x + 10$  somada a ruído gaussiano, variando o grau do polinômio de 1 a 8. Para cada caso, é necessário calcular os coeficientes do polinômio usando a pseudoinversa da matriz de Vandermonde, plotar gráficos que mostrem a função original, as amostras e o polinômio ajustado, e analisar se ocorreu overfitting ou underfitting. Por fim, deve-se comparar como o número de amostras influencia a qualidade da aproximação.

#### 3.3.1 CÓDIGO IMPLEMENTADO

```
import numpy as np
import matplotlib.pyplot as plt
# Configurações iniciais
np.random.seed(42) # para resultados reproduzíveis
N = 20 # número de amostras
x = np.linspace(-15, 10, N)
ruído = np.random.normal(0, 4, N) # ruído gaussiano (mean=0, sd=4)
y = 0.5*x**2 + 3*x + 10 + ruído # função geradora com ruído

# Para plotar suavemente os polinômios e função real
x_plot = np.linspace(-15, 10, 200)
y_real = 0.5*x_plot**2 + 3*x_plot + 10
# Ajuste polinomial
graus = range(1, 9) # polinômios do grau 1 ao 8

plt.figure(figsize=(20, 15))
```



```

for i, p in enumerate(graus, 1):
    # Ajuste polinomial
    coef = np.polyfit(x, y, p) # encontra os coeficientes w
    y_poly = np.polyval(coef, x_plot) # avalia o polinômio nos pontos para plot
    # Plotagem
    plt.subplot(3, 3, i)
    plt.scatter(x, y, color='red', label='Amostras')
    plt.plot(x_plot, y_real, color='black', label='Função geradora')
    plt.plot(x_plot, y_poly, color='blue', label=f'Polinômio grau {p}')
    # Anotação sobre underfitting/overfitting
    if p <= 2:
        nota = 'Underfitting'
    elif p >= 6:
        nota = 'Overfitting'
    else:
        nota = 'Bom ajuste'
    plt.title(f'Grau {p} - {nota}')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.tight_layout()
    plt.show()
# Observações
print("Observações:")
print("- Underfitting ocorre para grau 1 e 2: polinômio não consegue capturar a curva quadrática.")
print("- Bom ajuste ocorre para grau 3 a 5: segue bem a tendência da função geradora.")
print("- Overfitting ocorre para grau 6 a 8: polinômio tenta passar exatamente por todos os pontos, capturando o ruído.")

```

### 3.4 RESULTADOS

Foram gerados os gráficos a seguir:

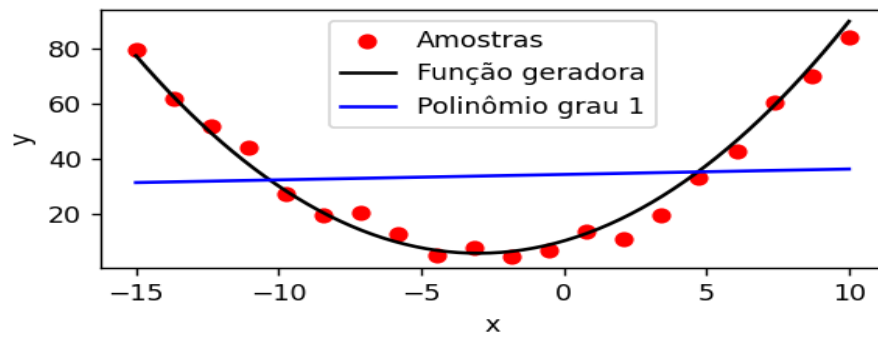


Figura 4 – Grau 1: Underfitting.

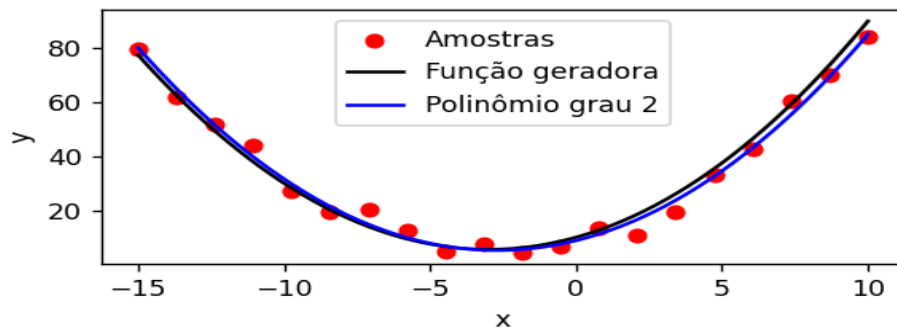


Figura 5 – Grau 2: Underfitting.

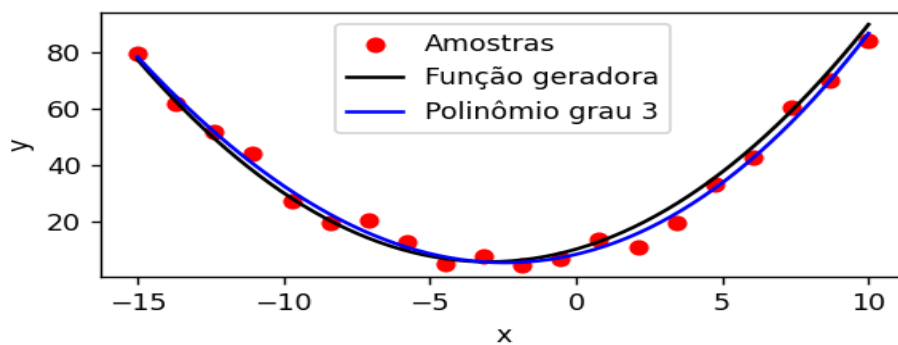


Figura 6 – Grau 3: Bom Ajuste.

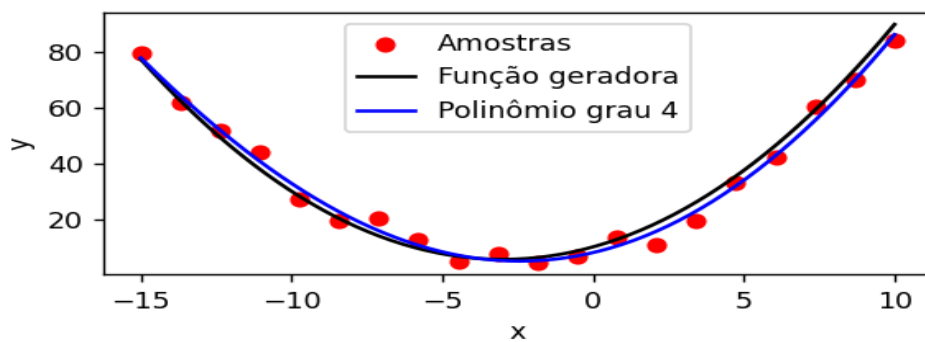


Figura 7 – Grau 4: Bom Ajuste.

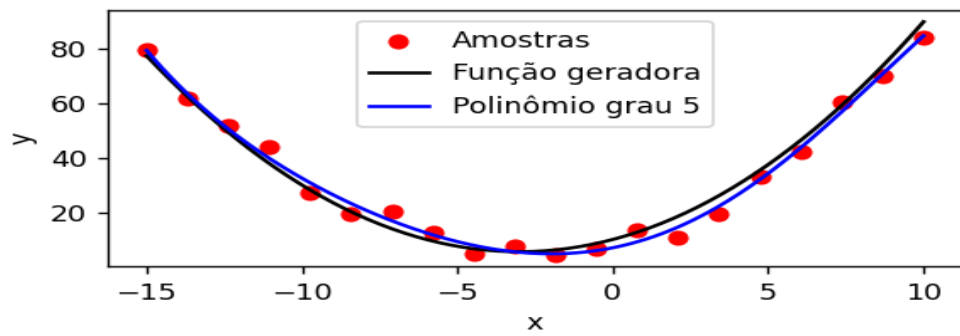


Figura 8 – Grau 5: Bom Ajuste.

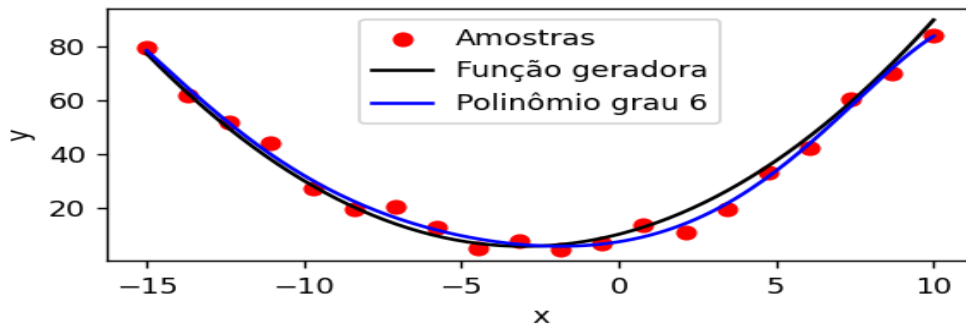


Figura 9 – Grau 6: Overfitting.

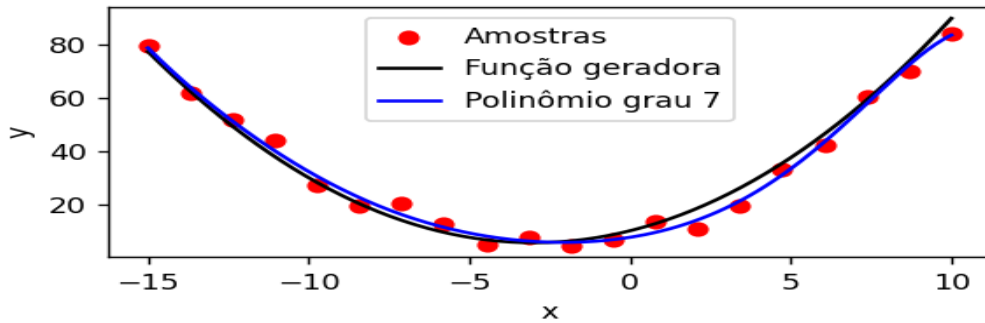


Figura 10 – Grau 7: Overfitting.

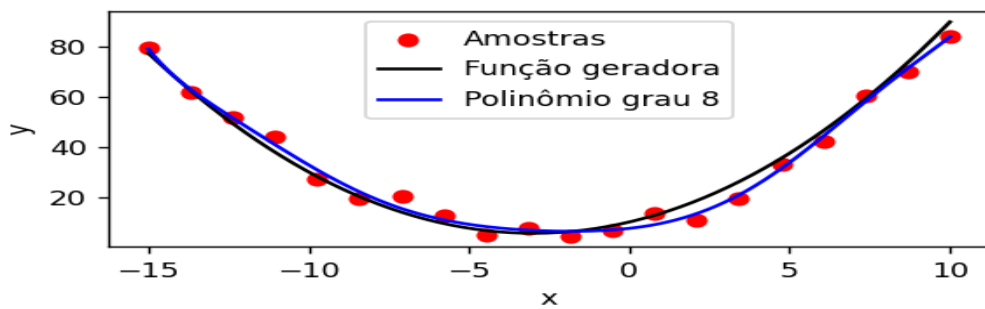


Figura 11 – Grau 8: Overfitting.

O código gera 20 amostras da função  $fg(x) = 0.5x^2 + 3x + 10$  com ruído gaussiano, ajusta polinômios de grau 1 a 8 usando least squares (np.polyfit); Plota cada polinômio, a função real e os pontos amostrados. Marca no título se o polinômio está em underfitting, bom ajuste ou overfitting e exibe observações sobre cada caso.

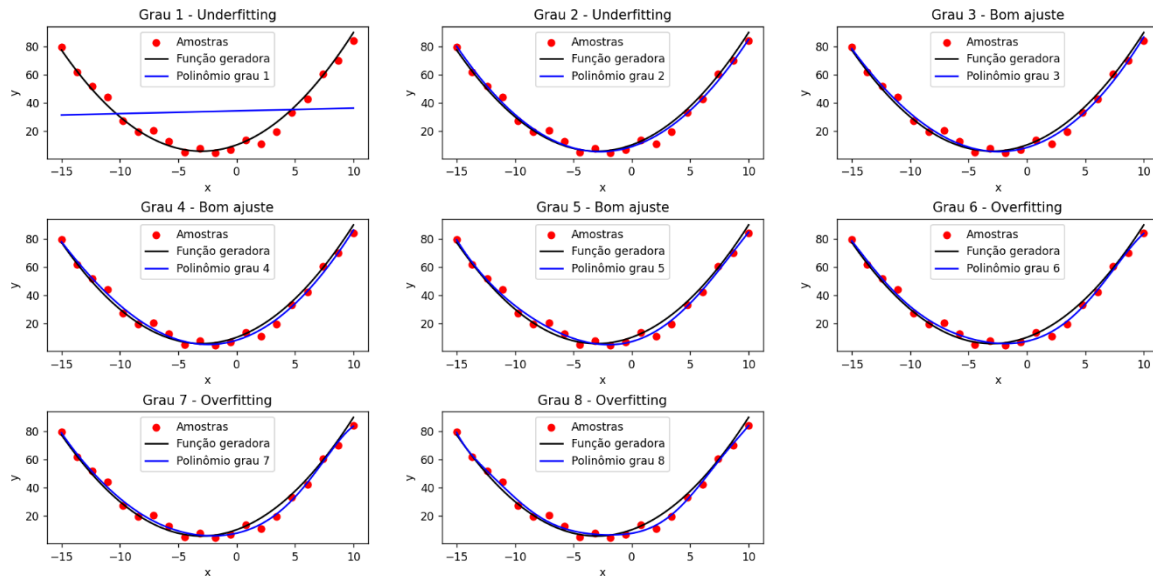


Figura 12 – Todos os casos gerados pelo código implementado.

- Underfitting ocorre para grau 1 e 2: polinômio não consegue capturar a curva quadrática.
- Bom ajuste ocorre para grau 3 a 5: segue bem a tendência da função geradora.
- Overfitting ocorre para grau 6 a 8: polinômio tenta passar exatamente por todos os pontos, capturando o ruído.

### 3.5 CÓDIGO IMPLEMENTADO PARA 100 AMOSTRAS

Na sequência, o processo para 100 amostras:

```
import numpy as np
import matplotlib.pyplot as plt
# Configurações iniciais
np.random.seed(42)
N = 100 # número de amostras
x = np.linspace(-15, 10, N)
ruído = np.random.normal(0, 4, N)
y = 0.5*x**2 + 3*x + 10 + ruído # função geradora com ruído
```

```

# Para plotar suavemente
x_plot = np.linspace(-15, 10, 400)
y_real = 0.5*x_plot**2 + 3*x_plot + 10

# Função para montar matriz H
def montar_H(x, grau):
    """Constrói a matriz H para polinômio de dado grau"""
    H = np.vander(x, grau+1, increasing=True) # colunas: x^0, x^1, ..., x^p
    return H

# Ajuste polinomial usando pseudoinversa
graus = range(1, 9)
plt.figure(figsize=(20, 15))

for i, p in enumerate(graus, 1):
    H = montar_H(x, p)
    w = np.linalg.pinv(H) @ y # pseudoinversa de H
    H_plot = montar_H(x_plot, p)
    y_poly = H_plot @ w # avaliação do polinômio nos pontos para plot
    # Plotagem
    plt.subplot(3, 3, i)
    plt.scatter(x, y, color='red', label='Amostras')
    plt.plot(x_plot, y_real, color='black', label='Função geradora')
    plt.plot(x_plot, y_poly, color='blue', label=f'Polinômio grau {p}')
    # Anotação sobre underfitting/overfitting
    if p <= 2:
        nota = 'Underfitting'
    elif p >= 7:
        nota = 'Overfitting'
    else:
        nota = 'Bom ajuste'
    plt.title(f'Grau {p} - {nota}')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
plt.tight_layout()
plt.show()

```

```
# Observações
print("Observações com pseudoinversa:")
print("- Underfitting: grau 1 e 2.")
print("- Bom ajuste: grau 3 a 6.")
print("- Overfitting: grau 7 e 8, embora menos pronunciado devido ao maior número de amostras (100).")
```

### 3.6 RESULTADOS

Foram gerados os gráficos a seguir:

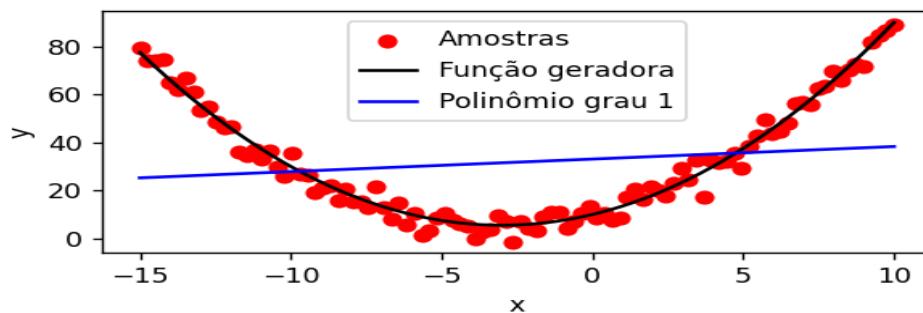


Figura 13 – Grau 1: Underfitting

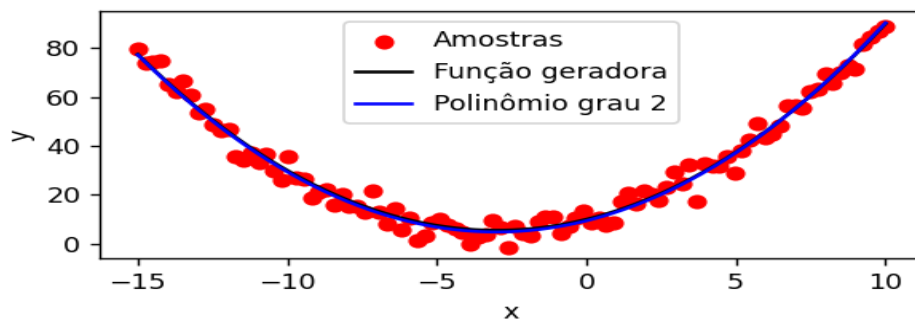


Figura 14 – Grau 2: Underfitting.

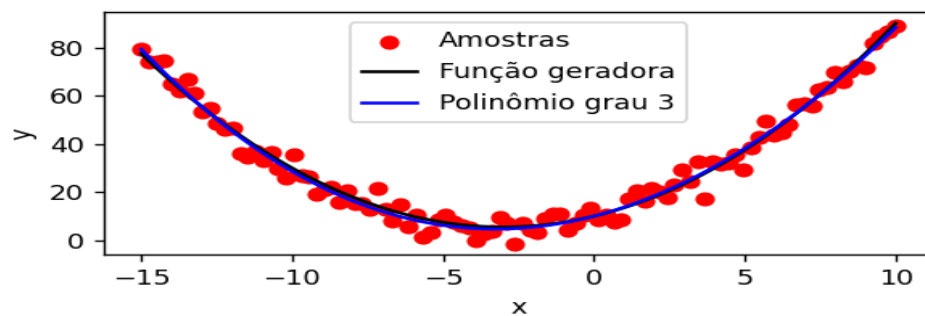


Figura 15 – Grau 3: Bom Ajuste.

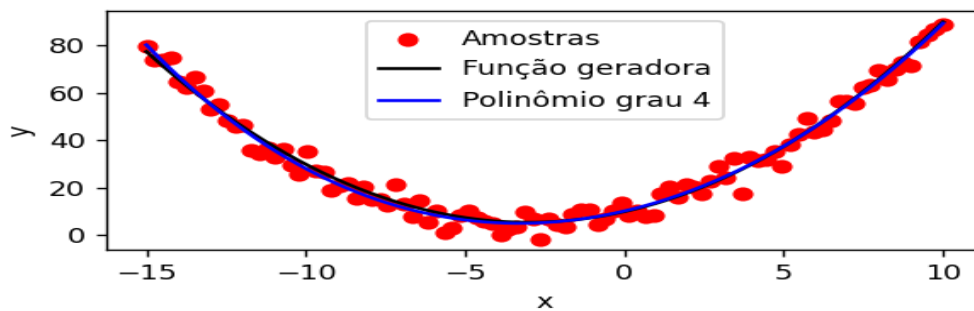


Figura 16 – Grau 4: Bom Ajuste.

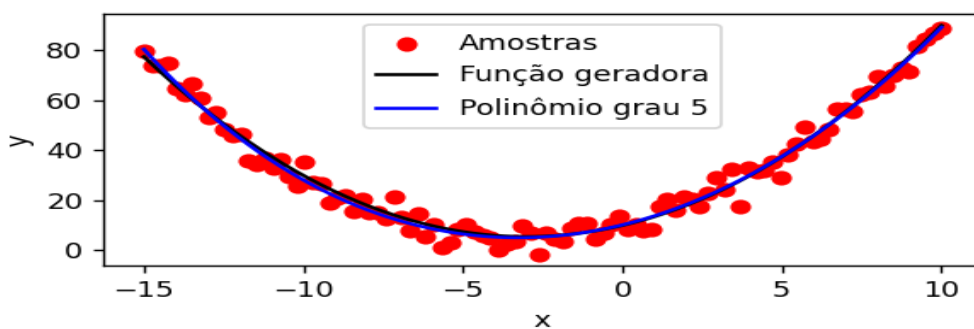


Figura 17 – Grau 5: Bom Ajuste.

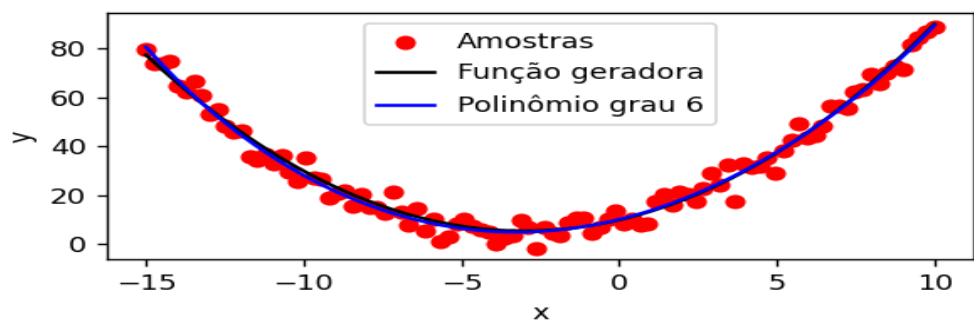


Figura 18 – Grau 6: Bom Ajuste.

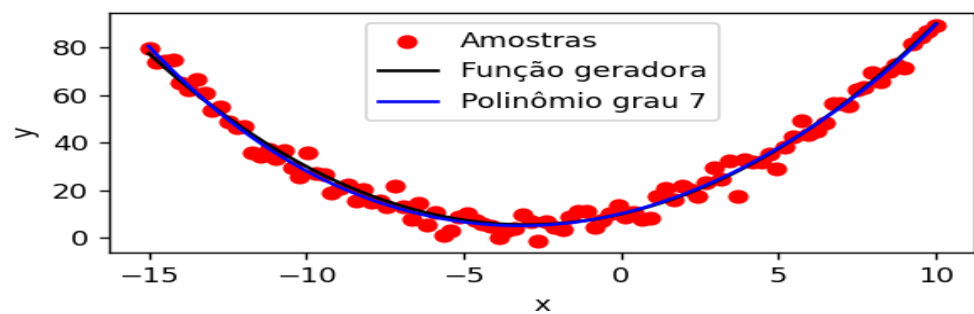


Figura 19 – Grau 7: Overfitting.

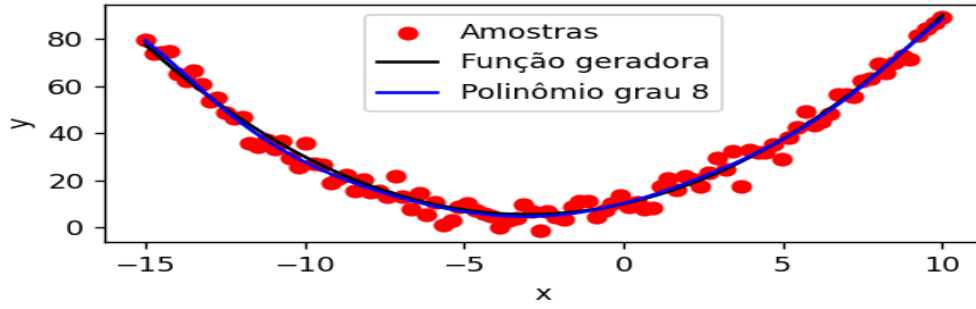


Figura 20 – Grau 8: Overfitting.

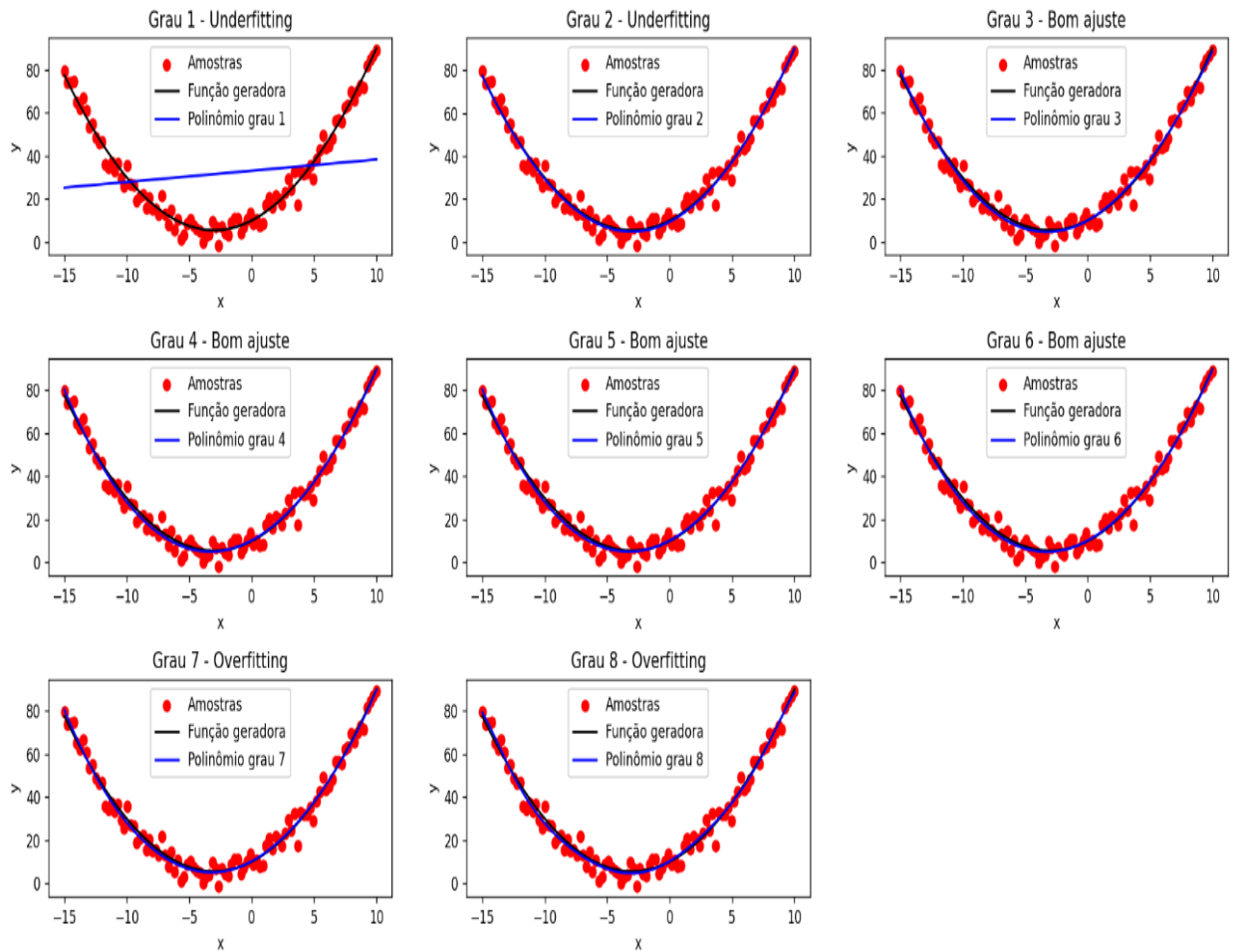


Figura 21 – Todos os casos gerados pelo código implementado.

A matriz  $H$  é construída com `np.vander(x, grau+1, increasing=True)`, de forma que cada coluna representa um termo do polinômio, desde o constante até o de grau  $p$ . A solução dos coeficientes é obtida pela pseudoinversa, `np.linalg.pinv(H)`, que fornece  $H^+$  e garante a minimização do erro quadrático. Para avaliar o polinômio ajustado em novos pontos, calcula-se  $y_{polly} = H_{plot} @ \omega$ , onde  $H_{plot}$  é a matriz de Vandermonde construída para os pontos de visualização. Observa-se que, ao aumentar o número de amostras de 20 para 100, o overfitting é



reduzido: polinômios de grau elevado ainda apresentam oscilações, mas em menor intensidade, mostrando como a quantidade de dados suaviza o ajuste e melhora a generalização.

Abaixo as observações com pseudoinversa:

- Underfitting: grau 1 e 2.
- Bom ajuste: grau 3 a 6.
- Overfitting: grau 7 e 8, embora menos pronunciado devido ao maior número de amostras (100).

### 3.7 ANÁLISE DOS RESULTADOS

Amostras  $N = 20$

- Grau 1-2: underfitting. Polinômio linear e quadrático simples não conseguem capturar a curvatura da função geradora  $f(x) = 0.5x^2 + 3x + 10$ .
- Grau 3-5: bom ajuste. O polinômio segue bem a função quadrática e representa corretamente a tendência.
- Grau 6-8: overfitting. O polinômio oscila para passar por todos os pontos, capturando o ruído gaussiano das amostras.

O gráfico confirma a relação entre o grau do polinômio e o ajuste: graus intermediários são ideais para este número de amostras.

Amostras  $N = 100$

- Grau 1-2: underfitting ainda presente, mas a tendência geral é mais visível devido ao maior número de pontos.
- Grau 3-6: bom ajuste. O polinômio consegue representar a função geradora com precisão, sem seguir o ruído.
- Grau 7-8: overfitting menos pronunciado. O aumento do número de amostras diminui a influência do ruído, suavizando as oscilações do polinômio de grau alto.

Esses resultados reforçam que mais amostras permitem polinômios mais complexos sem causar overfitting significativo.

## 4. CONCLUSÃO

O estudo do problema circular não-linearmente separável mostrou que a aplicação de uma projeção não-linear pode transformar dados complexos em um espaço linearmente separável. A função  $z = \sqrt{x^2 + y^2}$  permitiu separar as classes de forma simples e intuitiva, evidenciando a importância de técnicas de transformação de dados em aprendizado de máquina. A principal contribuição deste trabalho é a demonstração de como problemas aparentemente difíceis podem ser simplificados com uma escolha adequada de características ou projeções, reforçando o papel da engenharia de características na construção de classificadores eficientes. O estudo mostrou que

a aproximação polinomial de uma função quadrática com ruído depende de dois fatores fundamentais:

1. Grau do polinômio  $p$ :
  - Baixo ( $p = 1, 2$ )  $\rightarrow$  underfitting.
  - Médio ( $p = 3 - 5$  ou  $p = 3 - 6$ )  $\rightarrow$  bom ajuste.
  - Alto ( $p \geq 6$ )  $\rightarrow$  overfitting, mais evidente com poucas amostras.
2. Número de amostras  $N$ :
  - Poucas amostras amplificam o efeito do ruído, aumentando o overfitting.
  - Muitas amostras reduzem a influência do ruído, permitindo que polinômios de grau

alto ainda representem bem a função geradora.

O uso da matriz  $H$  e da pseudoinversa  $H^+$  garante a solução ótima de mínimos quadrados para qualquer conjunto de amostras, tornando a técnica robusta e aplicável a diferentes situações de aproximação polinomial.

## 5. REFERÊNCIAS

1. BISHOP, C. M. *Pattern Recognition and Machine Learning*. New York: Springer, 2006.
2. HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2. ed. New York: Springer, 2009.
3. DUDA, R. O.; HART, P. E.; STORK, D. G. *Pattern Classification*. 2. ed. New York: Wiley, 2001.
4. SCHÖLKOPF, B.; SMOLA, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press, 2002.
5. GOLUB, G. H.; VAN LOAN, C. F. *Matrix Computations*. 4. ed. Baltimore: Johns Hopkins University Press, 2013.
6. NOCEDAL, J.; WRIGHT, S. J. *Numerical Optimization*. 2. ed. New York: Springer, 2006.
7. MONTGOMERY, D. C.; PECK, E. A.; VINING, G. G. *Introduction to Linear Regression Analysis*. 5. ed. Hoboken: Wiley, 2012.