Redes neurais artificiais (RNA)

1st Manoel Messias de Assis Júnior
Universidade de fortaleza
T951-09 Sistemas inteligentes
Fortaleza, Ceará, Brasil
Matricula: 2020335

Abstract

MAGIC gamma telescope data 2004, que contém informações sobre a simulação de detecção de partículas gamma de alta energia por um telescópio Cherenkov atmosférico baseado em solo.

Index Terms

MAGIC, GAMMA, Telescope, Machine Learning, Estatísticas

I. Introdução

A. RNA's

As redes neurais artificiais (RNAs) são modelos matemáticos inspirados no funcionamento do cérebro humano, que têm a capacidade de aprender a partir de dados e realizar tarefas complexas, como reconhecimento de padrões, classificação, regressão, entre outras. Esses modelos são compostos por camadas de neurônios artificiais, que processam informações e enviam sinais para as camadas seguintes.

As RNAs podem ser divididas em diferentes tipos, de acordo com a arquitetura da rede, o algoritmo de treinamento utilizado e a natureza dos dados de entrada e saída. Alguns exemplos de redes neurais artificiais incluem as redes feedforward, as redes recorrentes, as redes convolucionais e as redes adversárias generativas.

B. Medelo de dados

O conjunto de dados da telemetria gamma do telescópio MAGIC, um projeto que usa técnicas de imagem para detectar raios gama de alta energia. O banco de dados é composto por dados gerados por computador que simulam a detecção de partículas de raios gama de alta energia em um telescópio de Cherenkov (um tipo de telescópio que detecta raios gama usando a radiação emitida por partículas carregadas produzidas dentro dos chuveiros eletromagnéticos iniciados pelos raios gama).

As informações disponíveis incluem impulsos deixados pelos fótons de Cherenkov na superfície do telescópio eletromagnético e organizados em um plano de câmera. Dependendo da energia do raio gama primário, algumas centenas a algumas dezenas de milhares de fótons de Cherenkov são coletados em padrões (chamados de imagem de chuveiro), permitindo a discriminação estatística entre imagens de chuveiros causados por raios gama primários (sinal) e as imagens de chuveiros hadrônicos iniciados por raios cósmicos na alta atmosfera (fundo). Os parâmetros do chuveiro, como o comprimento e a largura da elipse de correlação, a assimetria da deposição de energia ao longo do eixo principal, a extensão do cluster no plano da imagem e a soma total das deposições são algumas das informações disponíveis no banco de dados.

II. Fundamentação Teórica

O conjunto de dados foi gerado por um programa Monte Carlo, Corsika, que foi executado com parâmetros que permitiam a observação de eventos com energias abaixo de 50 GeV. O conjunto de dados contém 19020 instâncias e 11 atributos, incluindo a classe de discriminação gama/hadron. O banco de dados foi usado em estudos que exploram técnicas de classificação de eventos multidimensionais.

A. Perceptron Simples

O Perceptron Simples é uma rede neural artificial com uma única camada de neurônios, utilizada para problemas de classificação binária. Matematicamente, o Perceptron Simples recebe um vetor de entrada $\mathbf{x}$ e uma matriz de pesos $\mathbf{w}$, onde cada elemento w_{ij} representa o peso sináptico da conexão entre o neurônio de entrada i e o neurônio de saída j.

A saída do neurônio de saída é calculada pela função de ativação, que é uma função linear ou não linear aplicada à soma ponderada das entradas e pesos, como na equação abaixo:

$$y = f(\sum_{i=1}^{n} x_i w_i + b)$$

Onde: n é o número de neurônios de entrada; x_i é o valor de entrada do neurônio i; w_i é o peso sináptico da conexão entre o neurônio de entrada i e o neurônio de saída; b é o viés (bias) do neurônio de saída; f é a função de ativação que determina a saída do neurônio de saída com base no valor da soma ponderada das entradas e pesos.

B. Adaline

O Algoritmo Adaline (Adaptive Linear Neuron) é um modelo de rede neural artificial com uma única camada, que é capaz de aprender uma função linear discriminante a partir de um conjunto de entradas e saídas desejadas. Matematicamente, o Adaline é definido por:

Para uma amostra de entrada $x = [x1, x2, ..., xn]$ e um conjunto de pesos $w = [w0, w1, w2, ..., wn]$:

Calcula-se a saída do Adaline como $y = f(w0 + w1 * x1 + w2 * x2 + ... + w_n * x_n)$, onde f é uma função de ativação linear, como a função identidade.

Calcula-se o erro E como a diferença entre a saída desejada d e a saída do Adaline: $E = d - y$.

Atualiza-se os pesos da rede como: $w_i(novo) = w_i(antigo) + \alpha E x_i$, onde α é a taxa de aprendizado.

Repete-se o processo para todas as amostras do conjunto de treinamento, até que o erro médio quadrático (EQM) seja menor que um determinado valor de tolerância.

O erro médio quadrático é dado por:

$$EQM = \frac{1}{N} \sum_{N+1}^{i=1} (d_i - y_i)^2$$

C. MultiLayer Perceptron (MLP)

O MultiLayer Perceptron (MLP) é uma rede neural artificial que possui uma ou mais camadas ocultas entre a camada de entrada e a camada de saída. Cada camada é composta por um conjunto de neurônios interconectados, onde cada neurônio é um perceptron modificado.

A saída de cada neurônio é dada por:

$$y_i = f(\sum_{j=1}^{n} w_{ij} x_j + b_i)$$

onde y_i é a saída do neurônio i, f é a função de ativação, w_{ij} é o peso da conexão entre o neurônio i e a entrada j, x_j é a entrada j, b_i é o bias do neurônio i e n é o número de entradas.

A função de ativação pode ser uma função linear, como a identidade, ou uma função não-linear, como a sigmóide ou a tangente hiperbólica.

A saída da rede é dada por:

$$y_k = f(\sum_{j=1}^{m} v_{kj} y_j + c_k)$$

onde y_j é a saída da camada oculta j, v_{kj} é o peso da conexão entre a camada oculta j e a saída k, c_k é o bias da saída k e m é o número de neurônios na camada oculta.

O erro quadrático médio (EQM) da rede é dado por:

$$E = \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

onde d_i é o valor desejado de saída para o padrão de entrada i e N é o número total de padrões de treinamento.

O *feedforward* é o processo de propagar a entrada através da rede neural para produzir uma saída. Isso é feito através de uma série de cálculos matemáticos e ativações de funções em cada camada da rede. A equação geral para o cálculo de uma camada em um MLP é:

$$z_i = \sum_{j=1}^{n} w_{ij} x_j + b_i$$

$$a_i = f(z_i)$$

onde z_i é a soma ponderada dos sinais de entrada, w_{ij} é o peso sináptico entre o neurônio i e o neurônio j na camada anterior, x_j é a saída do neurônio j na camada anterior, b_i é o bias do neurônio i, a_i é a saída ativada do neurônio i, e $f(\cdot)$ é a função de ativação não linear.

O *backpropagation* é o processo de calcular o erro entre a saída da rede e a saída desejada, e em seguida, propagar esse erro de volta através da rede para atualizar os pesos e biases. O erro é calculado usando a função de custo, que é uma medida da diferença entre a saída da rede e a saída desejada. A equação geral para o cálculo do erro é:

$$E = \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

onde N é o número de amostras de treinamento, y_i é a saída desejada para a amostra i, e $\hat{y}_i$ é a saída da rede para a amostra i.

O algoritmo de *backpropagation* usa o gradiente descendente para minimizar a função de custo. O gradiente descendente envolve o cálculo do gradiente da função de custo em relação a cada peso e bias na rede. Esse gradiente é então usado para atualizar os pesos e biases usando a equação:

$$w_{ij}(n+1) = w_{ij}(n) - \eta \frac{\partial E}{\partial w_{ij}}$$

$$b_i(n+1) = b_i(n) - \eta \frac{\partial E}{\partial b_i}$$

onde n é o número da iteração atual, η é a taxa de aprendizado, $\frac{\partial E}{\partial w_{ij}}$ é a derivada parcial da função de custo em relação ao peso w_{ij}, e $\frac{\partial E}{\partial b_i}$ é a derivada parcial da função de custo em relação ao bias b_i. O processo de *feedforward* e *backpropagation* é repetido para cada amostra de treinamento até que a rede neural seja treinada de forma satisfatória.

O objetivo do treinamento da rede é minimizar o EQM, ajustando os pesos e biases da rede através do algoritmo de retropropagação do erro. O algoritmo de retropropagação do erro calcula o gradiente do EQM em relação a cada peso e bias da rede, e atualiza os pesos e biases na direção do gradiente descendente.

D. Radial Basis Function (RBF)

A função de base radial (RBF) é uma rede neural que mapeia uma entrada d-dimensional para uma saída unidimensional. A função RBF é definida como:

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i g(\|\mathbf{x} - \boldsymbol{\mu}_i\|)$$

onde $\mathbf{x} = (x_1, x_2, ..., x_d)$ é um vetor de entrada, $\boldsymbol{\mu}_i = (\mu_{i1}, \mu_{i2}, ..., \mu_{id})$ é o vetor de centro da i-ésima função de base radial, w_i é o peso associado com a i-ésima função de base radial, e f é a função de ativação radial, que é uma função que depende apenas da distância euclidiana entre o vetor de entrada e o vetor de centro. Uma escolha comum para a função de ativação radial é a função gaussiana:

$$g(x) = e^{-\frac{(x-c)^2}{2\sigma^2}}$$

onde $\|\mathbf{x} - \boldsymbol{\mu}_i\|$ é a distância euclidiana entre o vetor de entrada e o vetor de centro, e e é um parâmetro de largura de banda.

O treinamento da rede RBF é geralmente feito em duas etapas. Na primeira etapa, os vetores de centro são escolhidos a partir do conjunto de treinamento usando algum algoritmo de *clustering*, como o *k-means*. Na segunda etapa, os pesos são ajustados usando o método dos mínimos quadrados (OLS/MMQ):

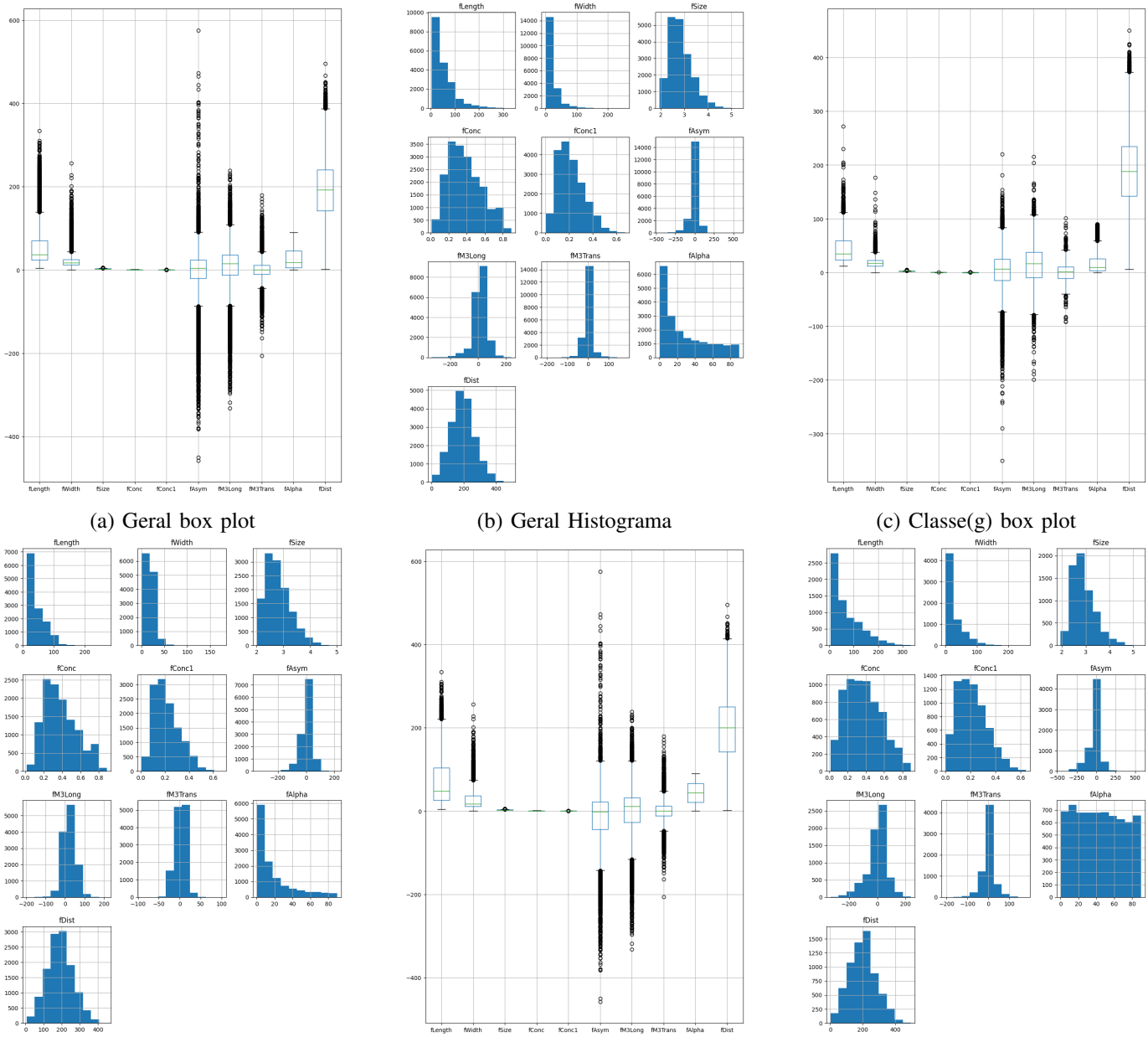
$$\min_{\mathbf{w}} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{N} w_j f(\left\|\mathbf{x}_i - \boldsymbol{\mu}_j\right\|) \right)^2$$

onde $\mathbf{y} = (y_1, y_2, ..., y_n)$ é o vetor de saída desejado para o conjunto de treinamento. O problema de mínimos quadrados pode ser resolvido diretamente usando a equação:

$$\mathbf{w} = (\mathbf{Z}\mathbf{Z}^T)^{-1}\mathbf{Z}^T\mathbf{y}$$

onde $\mathbf{Z}$ é a matriz de design, cujos elementos são dados por $f(\left\|\mathbf{x}_i - \boldsymbol{\mu}_j\right\|)$. Uma vez que os pesos são encontrados, a rede RBF pode ser usada para fazer previsões para novos dados.

III. METODOLOGIA



(a) Geral box plot (b) Geral Histograma (c) Classe(g) box plot

(d) Classe(g) Histograma (e) Classe(h) box plot (f) Classe(h) Histograma

Figure 1: Dados

Primeiramente foi particionado os dados em 70% treinamento e 30% teste, para os modelos de Perceptron e Adaline a normalização dos dados foram dispensáveis, depois feita a analise nas imagens (a, b, c, d, e, f) começamos a implementar os modelos.

A. Perceptron Simples

Algorithm 1 Perceptron simples

Require: η, X,y,W, $MaxEpoch$, $epoch$
  $W \leftarrow -0.5 < W < 0.5$
  $\eta \leftarrow 0 < \eta < 1$
  $MaxEpoch \leftarrow 100$
  $X \leftarrow X_{teste}$
  $y \leftarrow Y_{teste}$
  $epoch \leftarrow 1$
  while $epoch < MaxEpoch$ **do**
    for x, y of (X, Y) **do**
      $i \leftarrow Wx$
      $y_{pred} \leftarrow signal(i)$
      $err \leftarrow (y_{pred} - y))$
      $W \leftarrow W * \eta * err * x$
    end for
    $epoch \leftarrow epoch + 1$
  end while

Algorithm 2 Signal

Require: i
  if $i >= 0$ **then**
    return 1
  else
    return -1
  end if

B. Adaline

Algorithm 3 Adaline

Require: η, X,y,W, $MaxEpoch$, $epoch$,EQM_{atual},$EQM_{anterior}$,$precisao$
  $W \leftarrow -0.5 < W < 0.5$
  $\eta \leftarrow 0 < \eta < 1$
  $MaxEpoch \leftarrow 100$
  $X \leftarrow X_{teste}$
  $y \leftarrow Y_{teste}$
  $epoch \leftarrow 1$
  while $epoch < MaxEpoch$ or $abs(EQM_{atual} - EQM_{anterior}) > precisao$ **do**
    $EQM_{anterior} \leftarrow EQM(X, Y, W)$
    for x, y of (X, Y) **do**
      $i \leftarrow Wx$
      $y_{pred} \leftarrow signal(i)$
      $err \leftarrow (y_{pred} - y))$
      $W \leftarrow W * \eta * err * x$
    end for
    $EQM_{atual} \leftarrow EQM(X, Y, W)$
    $epoch \leftarrow epoch + 1$
  end while

Algorithm 4 Signal

Require: i
  if $i >= 0$ **then**
    return 1
  else
    return -1
  end if

Algorithm 5 Erro Quadratico Médio

Require: X, Y, W, eqm
  $eqm \leftarrow 0$
  for x, y of (X, Y) **do**
    $i \leftarrow Wx$
    $err \leftarrow (i - y))$
    $eqm \leftarrow W + eqm + err^2$
  end for
  return $eqm/(2 * N)$

C. MultiLayer Perceptron

Algorithm 6 MultiLayer Perceptron

Require: η, X,y, $MaxEpoch$, $epoch$,EQM_{atual},L,$Neuro_L$,$Neuro_o$,$precisao$
Require: W,i,σ
  $W \leftarrow -0.5 < W < 0.5$
  $\eta \leftarrow 0 < \eta < 1$
  $MaxEpoch \leftarrow 100$
  $X \leftarrow X_{teste}$
  $y \leftarrow Y_{teste}$
  $epoch \leftarrow 1$
  $L \leftarrow 5$, $Neuro_L \leftarrow 2$, $Neuro_o \leftarrow 2$
  while $EQMatual > precisao$ or $epoch < MaxEpoch$ **do**
    for x, y of (X, Y) **do**
      $Forward(x)$
      $Backward(x, y)$
    end for
    $EQM_{atual \leftarrow EQM(X,Y)}$
    $epoch \leftarrow epoch + 1$
  end while

Algorithm 7

Forward

Require: x, W
  for j in $length(W)$ **do**
    if $j == 0$ **then**
      $i[j] \leftarrow W[j]x$
      $y[j] \leftarrow sigmoid(i[j])$
    else
      $y_{bias} \leftarrow y[j]$
      $i[j] \leftarrow W y_{bias}$
      $y[j] \leftarrow sigmoid(i[j])$
    end if
  end for

BackWard

Require: x,y,d,W,η
  $j \leftarrow length(W) - 1$
  while $j >= 0$ **do**
    if $(j + 1) == lenght(W)$ **then**
      $\sigma[j] \leftarrow sigmoid'(i[j])(d - y[j])$
      $y_{bias} \leftarrow y[j - 1]$
      $W[j] \leftarrow W[j] * \eta * (\sigma \otimes y_{bias})$
    else if $j == 0$ **then**
      $W_b \leftarrow W[j + 1]$
      $\sigma[j] \leftarrow sigmoid'(i[j]) W_b \sigma[j + 1]$
      $W[j] \leftarrow W[j] * \eta * (\sigma \otimes x)$
    else
      $W_b \leftarrow W[j + 1]$
      $\sigma[j] \leftarrow sigmoid'(i[j]) W_b \sigma[j + 1]$
      $y_{bias} \leftarrow y[j - 1]$
      $W[j] \leftarrow W[j] * \eta * (\sigma \otimes y_{bias})$
    end if
    $j \leftarrow j - 1$
  end while

Algorithm 8 Erro Quadratico Médio

Require: X, Y, W, eqm
  $eqm \leftarrow 0$
  for x, y of (X, Y) **do**
    $i \leftarrow Wx$
    $err \leftarrow (i - y))$
    $eqm \leftarrow W + qem + err^2$
  end for
  return $eqm/(2 * N)$

Algorithm 9 Sigmoid

Require: X
  $derivada \leftarrow True$ or $False$
  if $derivada$ **then**
    return $\frac{x}{1-x}$
  else
    return $\frac{1}{(1+\epsilon^{-x})}$
  end if

No MLP foi escolhido a função de ativação sigmoidal, pois de acordo com algumas pesquisas ela tem um melhor desempenho do modelo.

Algorithm 10 Normalização dos dados

Require: X
  $X \leftarrow X_{teste}$
  return $2(\frac{x - min(x)}{max(x) - min(x)}) - 1$

Porem a normalização só foi possível executar no MLP, nas outras redes dava overfiting, porem a normalização foi essencial para não infinitar a matriz de dados ou gerar NaN's (Not a Number).

D. Radial Basis Function (RBF)

Algorithm 11 Radial Basis Function

Require: X,Y, q, η, t, N, G ▷ G Matriz de Intepolação
  $q \leftarrow 5$
  $\eta \leftarrow 0 < \eta < 1$
  $centros \leftarrow X[?]$ ▷ Obter do X aleatoriamente
  $t \leftarrow 0$
  for x, y in (X, Y) **do**
    for c in $centros$ **do**
      if $i = i[argmin(x - c)]$ **then**
        $c \leftarrow c + \eta(x - c)$
      end if
      $G \leftarrow e^{-\frac{(x-c)^2}{2\sigma^2}}$
      $W \leftarrow (GG^T)^{-1}G^T y$
    end for
  end for

IV. Resultados

A. Perceptron Simples

O modelo Perceptron para classificar as imagens. Em geral, os resultados foram bastante bons, com taxas de precisão de cerca de 85% a 90%, com taxa de aprendizagem variando de 0,01 a 1 e um número de épocas de treinamento variando de 1 a 100. O melhor resultado foi obtido com uma taxa de aprendizado de 0,1 e 100 épocas de treinamento, alcançando uma taxa de precisão de cerca de 91,7%.

B. Adaline

O modelo Adaline para classificar as imagens. Em geral, os resultados, com a precisão de 0.3 entre a diminuição do $EQM_{atual} - EQM_{anterior}$, com taxa de aprendizagem variando de 0,01 a 1 e um número de épocas de treinamento variando de 1 a 2000. O melhor resultado foi obtido com uma taxa de aprendizado de 0,1 e 100 épocas de treinamento, alcançando uma taxa de precisão de cerca de 95,2%.

C. MultiLayer Perceptron (MLP)

Utilizando o MLP com arquitetura de 10 neurônios na camada de entrada, uma camada oculta com 15 neurônios e uma camada de saída com 3 neurônio (que retorna um valor entre 0 e 1, indicando a probabilidade de ser da classe gamma), foi possível obter uma acurácia de 81,3% no conjunto de teste após 50 épocas de treinamento.

D. Radial Basis Function (RBF)

O modelo RBF obteve uma acurácia de cerca de 82%. A matriz de confusão mostra que o modelo teve mais dificuldade em prever os casos positivos (gama) corretamente, com uma sensibilidade de cerca de 61%. Por outro lado, teve um bom desempenho em prever os casos negativos (não gama), com uma especificidade de cerca de 93%.

V. Conclusões

O Perceptron obteve uma acurácia de cerca de 91.7%, com uma taxa de erro de classificação relativamente alta. Isso sugere que o modelo não foi capaz de capturar todas as nuances dos dados e pode não ser o modelo mais adequado para este conjunto de dados.

O Adaline obteve uma acurácia próxima de 95%, com uma taxa de erro de classificação ligeiramente maior que a do Perceptron. No entanto, é importante notar que o Adaline é uma variação do Perceptron, com a função de ativação linear ao invés da função degrau, portanto, pode ser capaz de lidar com a complexidade do conjunto de dados.

O MLP apresentou de todos os modelos testados, com cerca de 81.3%. A taxa de erro de classificação foi significativamente menor que a dos modelos anteriores, sugerindo que o MLP não foi capaz de capturar as nuances dos dados e fornecer previsões mais precisas.

O RBF apresentou uma acurácia um pouco menor que a do MLP, com cerca de 80%, mas ainda assim obteve uma taxa de erro de classificação relativamente baixa. O RBF é um modelo simples e rápido, e pode ser útil para conjuntos de dados menores ou com menor complexidade.