Trabalho de Redes Neurais Artificiais

Poda Aleatória de ELMs e regularização.

Aluno : Arthur Felipe Reis Souza.

Professor : Antonio de Padua Braga.

May 19, 2024

Sumário

Introdução

- Introduzir as Redes Neurais Artificias.
- Explicar a função de custo multiobjetiva, e a necessidade da regularização.

Desenvolvimento

- Explicar as redes neurais ELM (Extreme Learning Machine).
- Explicar os métodos de regularização que serão usados.
- Funções de treinamento com os metodos de regularização.

Resultados e Discussões

- Aplicar esses métodos de regularização nos datasets.
- Variar os hiperparâmetros e analisar os resultados.
- Discutir os resultados para a variação dos hiperparâmetros.

Conclusão

- Realizar conclusões sobre as análises feitas no trabalho.

Referências

- Referências bibliográficas utilizadas no trabalho.

Introdução

Atualmente, com a evolução tecnológica e o desenvolvimento em áreas da ciência e engenharia, é possível evidenciar pessoas e objetos gerando dados o tempo inteiro. Os algoritmos de Machine Learning são modelos estatísticos e matemáticos que, quando implementados em 1 computador, se tornam úteis para extrair informações relevantes acerca dessa grande quantidade de dados. A escolha do algoritmo de Machine Learning dependerá do tipo de problema a ser resolvido, também da complexidade e tamanho da base de dados. Algoritmos diferentes podem ter um desempenho semelhante, mesmo para uma grande diferença de tempo e recurso computacional gasto. Portanto, é necessário analisar profundamente o desempenho de cada algoritmo para um mesmo problema, e escolher o que terá um melhor equilíbrio entre custo e performance[3].

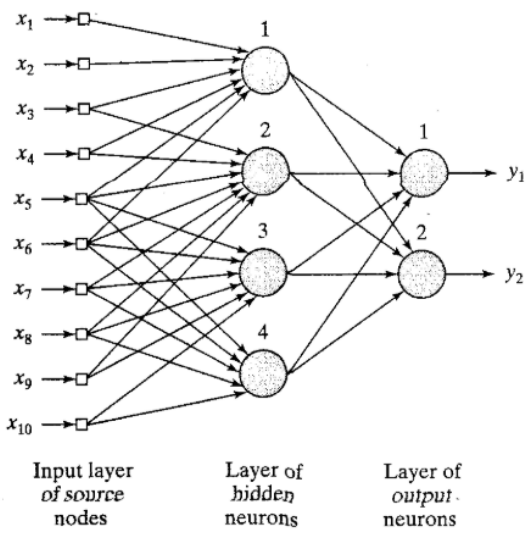

FIGURE 1.20 Illustrating the combined use of a receptive field and weight-sharing. All four hidden neurons share the same set of weights for their synaptic connections.

Figure 1: Imagem retirada do livro Neural Networks, Simon Haykin.[1]

As Redes Neurais Artificias são algoritmos de Machine Learning que consistem no ajuste dos parâmetros da rede para minimizar, parcialmente ou totalmente, uma função de custo J. Foi aprendido na fase inicial do curso a minimização de uma função de custo J que tem caráter uniobjetivo. Para isso, se ajustava os parâmetros da rede de forma a minimizar o erro de saída da rede, aproximando a saída da rede à saída desejada. O problema de minimizar apenas o erro de saída da rede é que, quando o modelo é muito complexo ele tenderá ao overfitting, que acontece quando a rede não aprende corretamente a função geradora dos dados e gera saídas incorretas para os dados de teste. As técnicas de regularização surgem para controlar a complexidade do modelo, e tendem a evitar esse overfitting. No trabalho serão estudadas as técnicas de regularização Ridge Regression (L2 Regression) e o Pruning para as Redes Neurais do Tipo ELM (Extreme Learning Machines). Serão utilizadas bases de dados que contém problemas de classificação e regressão, onde os resultados serão analisados. Com essa análise, uma Conclusão sobre esses métodos de regularização será feita.

Metodologia

No trabalho, foi utilizada as redes neurais ELM (Extreme Learning Machine). As redes neurais ELM (Extreme Learning Machine) tem uma estrutura parecida com as redes do tipo MLP (Multi Layer Perceptron), onde as entradas são transformadas até a camada de saída. No entanto, elas se diferem na forma como são treinadas. As redes ELM se baseiam no Teorema de Cover, o qual afirma que uma expansão aleatória de alta dimensão da camada intermediária resulta em uma chance alta de linearização do problema. Sendo assim, com o problema linearmente separável na camada de saída basta aplicar um modelo linear que melhor irá se adequar ao problema.[4]

Após a criação das redes neurais ELM, métodos e heurísticas foram usadas no fito de tentar regularizar o modelo. Afim de encontrar os parâmetros que resultam em um melhor desempenho do modelo, foi utilizada a técnica Grid Search Cross-Validation. O Grid Search Cross-Validation é um algoritmo utilizado para encontrar os parâmetros que resultam em um melhor desempenho do modelo. Para isso, cria-se um grid com todas as possibilidades possíveis que os parâmetros podem assumir, em seguida, testa-se cada uma das configurações com o algoritmo Cross-Validation. Há maneiras diferentes de implementar o Cross-Validation, no trabalho foi usado o K-fold Cross-Validation.

O K-fold Cross-Validation é um algoritmo utilizado para treinar e testar o modelo. Ele se baseia em dividir os dados de entrada em K grupos. Dentro desses K grupos, (K - 1) serão utilizados para treino e 1 grupo para teste. Esse processo se repete até que todos os K grupos tenham sido usados para treino e teste, e ao final a média do desempenho será utilizada para avaliar o modelo com relação a uma métrica específica. Para os problemas de classificação, a métrica utilizada será a acurácia. Para os problemas de regressão será o MSE (erro quadrático médio).

```python
def grid_searchCV_L2(xin : np.ndarray, yin : np.ndarray, p : int, lam : np.ndarray, CV_groups : int, classification : bool):
    arr_lam = np.zeros(lam.shape[0])

    if classification == True:
        arr_acc = np.zeros(lam.shape[0])
        for index, value in enumerate(lam):
            arr_acc[index] = cross_validation_L2(xin = xin, yin = yin, p = p, lam = value, CV_groups = CV_groups, classification = True)
            arr_lam[index] = value
        idx = np.argmax(arr_acc)
        print(f"The model with best accuracy has the mean accuracy : {arr_acc[idx]}")
        print(f"The model parameters with best accuracy is using lambda : {arr_lam[idx]}")
        return arr_lam[idx], np.max(arr_acc), arr_acc
    else:
        arr_MSE = np.zeros(lam.shape[0])
        for index, value in enumerate(lam):
            arr_MSE[index] = cross_validation_L2(xin = xin, yin = yin, p = p, lam = value, CV_groups = CV_groups, classification = False)
            arr_lam[index] = value
        idx = np.argmin(arr_MSE)
        print(f"The model with lowest MSE is : {arr_MSE[idx]}")
        print(f"The model parameters with lowest MSE is using lambda : {arr_lam[idx]}")
        return arr_lam[idx], np.min(arr_MSE), arr_MSE
```

Figure 2: Código do Grid-Search para regularização Ridge Regression.

Além das heurísticas propostas no enunciado do trabalho (Ridge Regression e o Random Pruning), também foi utilizada a heurística de eliminação dos pesos com menos relevância e influência na saída do modelo. Para a análise do modelo, 6 bases de dados foram escolhidas de forma a visualizar bem a generalização do mesmo. 3 dessas bases são de regressão e 3 de classificação. As bases de dados de classificação são : base de dados Xor, base de dados Circle e o Breast Cancer. As bases de dados de regressão são : Função senoidal, Boston House e Wine Quality. Todas as bases que não foram geradas internamente em python foram extraídas do site de competição Kaggle.

Ridge Regression

O método de regularização Ridge Regression consiste em adicionar um termo de penalização na função de custo. Esse termo de penalização tem por objetivo limitar a magnitude dos pesos w e por consequência evitar o modelo do overfitting.

$$J = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \sum_{j=1}^{p}\lambda_j w_j^2$$

Figure 3: Função de custo adicionada do termo penalizador.[4]

Agora, a função de custo é multiobjetiva, pois envolve duas funções objetivos com um comportamento conflitante. Ou seja, ao minimizar uma das funções objetivos não minimiza a outra em conjunto. Para encontrar os parâmetros w que minimizam essa função de custo multiobjetiva, basta-se calcular a derivada dessa função de custo em relação aos parâmetros.

$$\sum_{i=1}^{N}\hat{y}_i h(\mathbf{x}_i, \mathbf{z}_j) + \lambda_j w_j = \sum_{i=1}^{N} y_i h(\mathbf{x}_i, \mathbf{z}_j)$$

Figure 4: Equação que resulta na minimização da função de custo multiobjetiva.[4]

A equação acima mostra a minimização para uma dimensão caracterizada por wj. Para abranger todos os parâmetros w que serão ajustados, o problema será mostrado como um conjunto de equações lineares, e resolvido através da álgebra linear.

$$\mathbf{w} = (\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}_p)^{-1}\mathbf{H}^T\mathbf{y}$$

Figure 5: Regularização descrita por uma relação de álgebra linear.[4]

O termo penalizador pode ser decidido seguindo diferentes heurísticas. Cada heurística diferente tem como objetivo somar um termo no cálculo da pseudoinversa. Esse termo penalizador também é útil para evitar a existência de matrizes singulares. No desenvolvimento do trabalho, a matriz de parâmetros W estava explodindo para valores muito altos, esse problema foi resolvido com a utilização de um termo penalizador na matriz H que gerou a pseudoinversa.

No trabalho, um valor λ que será somado a todos os elementos da diagonal principal no cálculo da pseudoinversa.

```python
def train_ELM_L2_REG(xin : np.ndarray, yin : np.ndarray, p : int, control : bool, lam : float) -> list:
    try:
        if yin.shape[1] == 1:
            pass
    except IndexError:
        yin = yin.reshape(-1, 1)
    try:
        if xin.shape[1] == 1:
            pass
    except IndexError:
        xin = xin.reshape(-1, 1)

    n = xin.shape[1]

    if control == True:
        Z = np.array([np.random.uniform(-0.5, 0.5) for _ in range((n + 1) * p)]).reshape(n + 1, -1)
        ones = np.ones((xin.shape[0], 1))
        xin = np.concatenate((xin,ones), axis = 1)
    else:
        Z = np.array([np.random.uniform(-0.5, 0.5) for _ in range(n * p)]).reshape(n , -1)

    # Fazendo a matriz H, que será 1 função sigmoidal sobre as entradas e os pesos intermediários aleatórios.
    H = np.tanh(np.dot(xin, Z))
    ones = np.ones((H.shape[0], 1))
    H = np.concatenate((H, ones), axis = 1)

    # Realizando a regularização e adicionando o termo de regularização, alterando a função de custo : w = ((HT)H + λIp)^(-1)(HT)y.
    diagonal_matrix = lam * np.eye(H.shape[1])
    w1 = np.linalg.inv(np.dot(np.transpose(H), H) + diagonal_matrix)
    w = np.dot(w1, np.dot(np.transpose(H), yin))

    return_list = []
    return_list.append(w)
    return_list.append(H)
    return_list.append(Z)
    return  return_list
```

Figure 6: Função que implementa o treino das redes ELM com regularização L2.

Pruning

As técnicas de pruning consistem na remoção de parâmetros individuais ou neurônios inteiros de uma rede neural. Essa remoção segue uma heurística e um conjunto de regras que, quando executada corretamente provoca várias melhorias no modelo como : melhor generalização do modelo, menos gasto computacional, e uma singificante melhoria na velocidade de teste.[5]

Duas heurísticas serão utilizadas para a remoção dos parâmetros da rede. A primeira é a poda aleatória que consiste em escolher N pesos aleatórios e removê-los da rede, ou seja, alterar esses pesos aleatórios para o valor 0. A segunda é a remoção dos parâmetros que tem menor influência na saída da rede, ou seja, os N menores pesos serão alterados para 0. Essa alteração dos parâmetros para 0 economiza recurso computacional na hora de realizar a validação do modelo, visto que menos cálculos precisarão ser realizados.

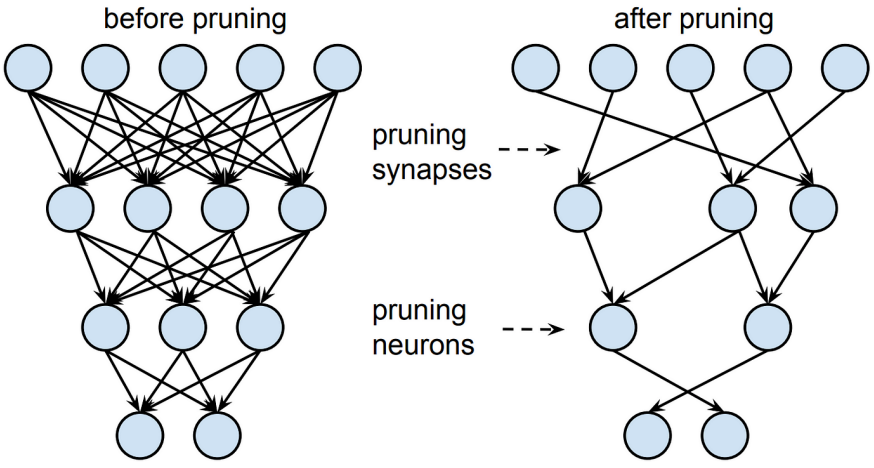

Figure 7: Rede Neural antes e depois do pruning.

```python
def train_ELM_PRUNING(xin : np.ndarray, yin : np.ndarray, p : int, keep_rate : float, control : bool) -> list:
    np.random.seed(np.random.randint(0, 10000))


    n = xin.shape[1] # Pegando o número de valores de cada entrada.


    if control == True:
        Z = np.array([np.random.uniform(-0.5, 0.5) for _ in range((n + 1) * p)]).reshape(n + 1, -1)
        ones = np.ones((xin.shape[0], 1))
        xin = np.concatenate((xin,ones), axis = 1)
    else:
        Z = np.array([np.random.uniform(-0.5, 0.5) for _ in range(n * p)]).reshape(n , -1)


    H = np.tanh(np.dot(xin, Z))
    scaler = StandardScaler()
    H = scaler.fit_transform(H)


    ones = np.ones((H.shape[0], 1))
    H = np.concatenate((H, ones), axis = 1)
    #print(f"det : {np.linalg.det(np.dot(np.transpose(H), H))}")

    w = np.dot(np.linalg.pinv(H), yin) # w = H*y

    try:
        N_col_W = w.shape[1]
    except IndexError:
        w = w.reshape(-1, 1)
        N_col_W = w.shape[1]

    for i in range(N_col_W): # Removendo os pesos menos relevantes da camada de saída.

        N_dropped_neurons = int(np.ceil((1 - keep_rate) * w[:, i].shape[0])) # Pegando o número de neurônios que serão dropados.
        sequence = np.arange(start = 0, stop = w[:, i].shape[0], step = 1).tolist()
        out_pos = random.sample(population = sequence, k = N_dropped_neurons)
        for j in out_pos: # Zerando os pesos menos relevantes.
            w[j, i] = 0


    return_list = list()
    return_list.append(w)
    return_list.append(H)
    return_list.append(Z) # Conexões são desligadas apenas no treino, portanto tenho que mandar a matriz Z completa.
    return  return_list
```

Ln 12,

Figure 8: Função que implementa o treino das redes ELM com a poda aleatória.

```python
def train_ELM_PRUNING(xin : np.ndarray, yin : np.ndarray, p : int, keep_rate : float, control : bool) -> list:
    np.random.seed(np.random.randint(0, 10000))

    n = xin.shape[1] # Pegando o número de valores de cada entrada.


    if control == True:
        Z = np.array([np.random.uniform(-0.5, 0.5) for _ in range((n + 1) * p)]).reshape(n + 1, -1)
        ones = np.ones((xin.shape[0], 1))
        xin = np.concatenate((xin,ones), axis = 1)
    else:
        Z = np.array([np.random.uniform(-0.5, 0.5) for _ in range(n * p)]).reshape(n , -1)


    H = np.tanh(np.dot(xin, Z))
    scaler = StandardScaler()
    H = scaler.fit_transform(H)


    ones = np.ones((H.shape[0], 1))
    H = np.concatenate((H, ones), axis = 1)
    #print(f"det : {np.linalg.det(np.dot(np.transpose(H), H))}")

    w_aux = np.dot(np.transpose(H), H) + 0.01*np.eye(H.shape[1])

    w_aux = np.dot(np.linalg.inv(w_aux), np.transpose(H))

    w = np.dot(w_aux, yin)

    #w = np.dot(np.linalg.pinv(H), yin) # w = H*y

    N_col_W = w.shape[1]

    for i in range(N_col_W): # Removendo os pesos menos relevantes da camada de saída.
        N_dropped_neurons = int(np.ceil((1 - keep_rate) * w[:, i].shape[0])) # Pegando o número de neurônios que serão dropados.
        idx = np.array(np.argsort(np.abs(w[:, i]))) # Pegandos os índices dos neurônios que serão dropados.
        lowest_val = idx[:N_dropped_neurons]
        for j in lowest_val: # Zerando os pesos menos relevantes.
            w[j, i] = 0



    return_list = list()
    return_list.append(w)
    return_list.append(H)
    return_list.append(Z) # Conexões são desligadas apenas no treino, portanto tenho que mandar a matriz Z completa.
    return  return_list
```

Figure 9: Função que implementa o treino das redes ELM com a poda do peso menos influente.

Portanto, ao utilizar os dois métodos de regularização, é de se esperar uma redução no termo variance do modelo (erro médio sobre os dados de teste), enquanto há um aumento no bias (erro médio nos dados de treino). Essa relação bias-variance indica um tradeoff entre o erro médio sobre os dados de treino e o erro médio sobre os dados de teste. Deixar o modelo bem regularizado e menos complexo reduz o termo variance enquanto aumenta o bias. Deixar o modelo pouco regularizado aumenta o termo variance e levará o modelo ao overfitting.

Resultados e Discussões

As imagens abaixo irão evidenciar a saída de cada um dos 3 modelos (Ridge Regression, Random Pruning, Pruning Lowest Value) para todos as bases de dados de classificação. O número de neurônios usados na camada intermediária em cada um dos modelos foi 100. Os valores de λ e também do número de neurônios a serem cortados no pruning foram obtidos com o Grid Search Cross-Validation.

XOR

Ridge Regression

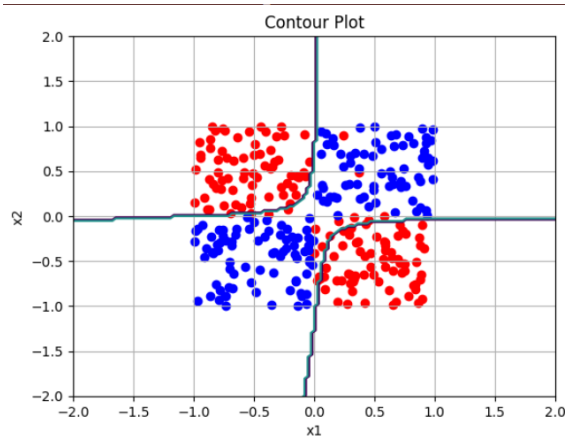

Figure 10: Gráfico de separação usando a regularização L2.

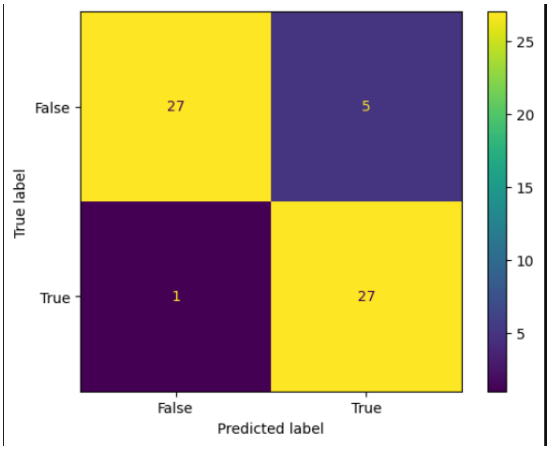

Figure 11: Matriz de confusão usando a regularização L2.

Random Pruning

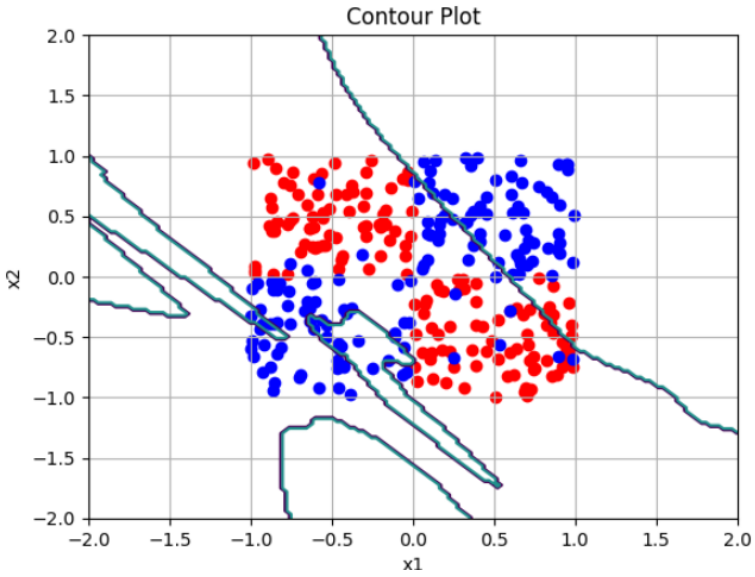

Figure 12: Gráfico de separação usando o random pruning.

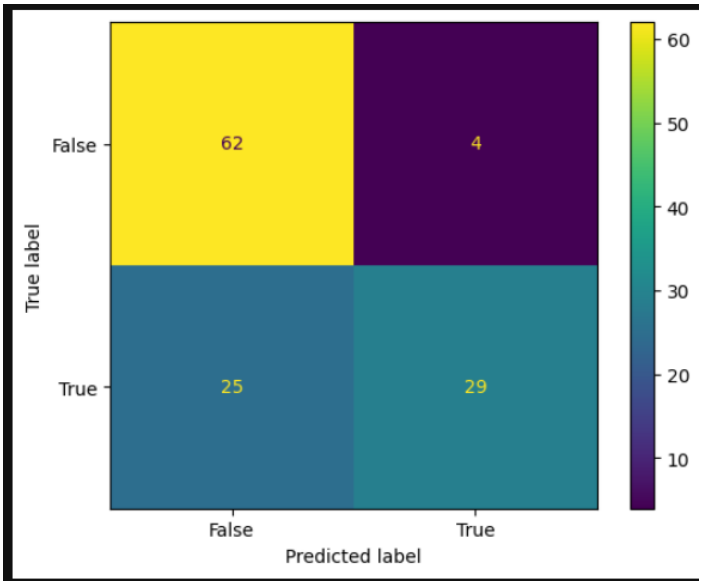

Figure 13: Matriz de confusão usando o random pruning.

Pruning Lowest Values

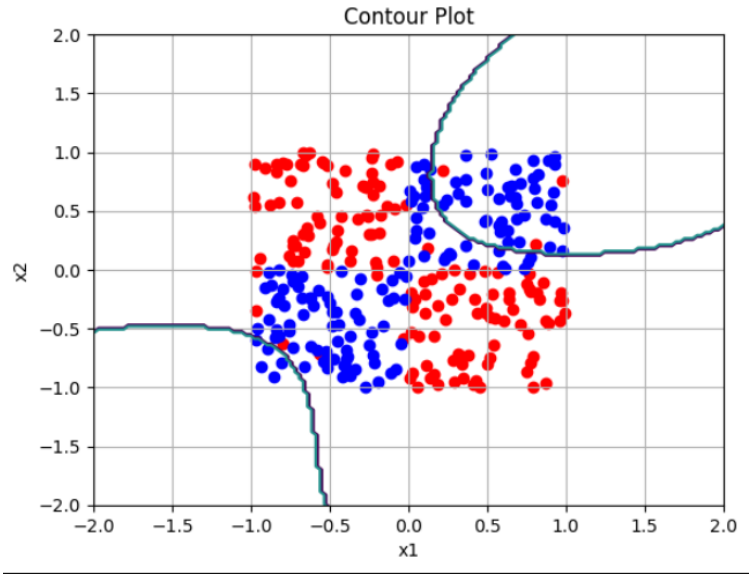

Figure 14: Gráfico de separação usando o pruning lowest value.

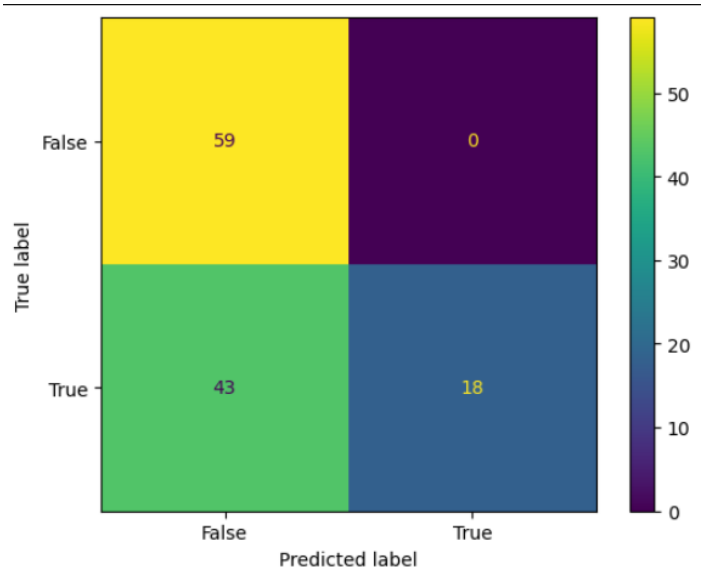

Figure 15: Matriz de confusão usando o pruning lowest value.

Circle

Ridge Regression

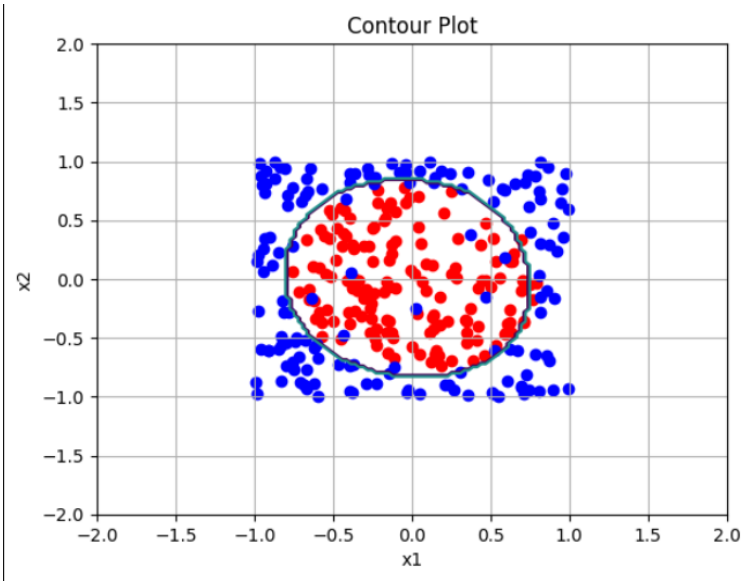

Figure 16: Gráfico de separação usando a regularização L2.

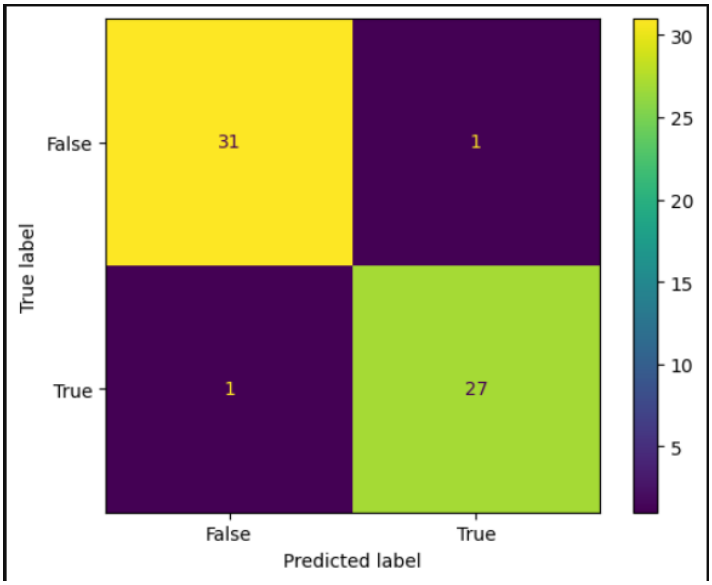

Figure 17: Matriz de confusão usando a regularização L2.

Random Pruning

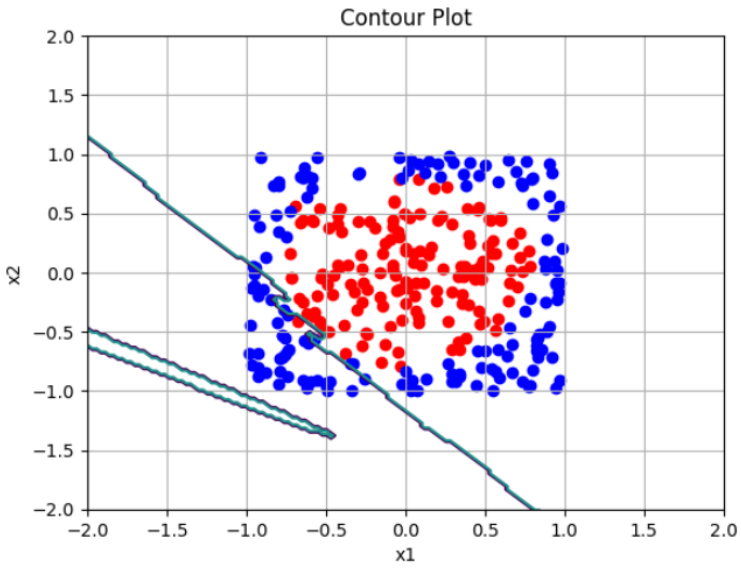

Figure 18: Gráfico de separação usando o random pruning.

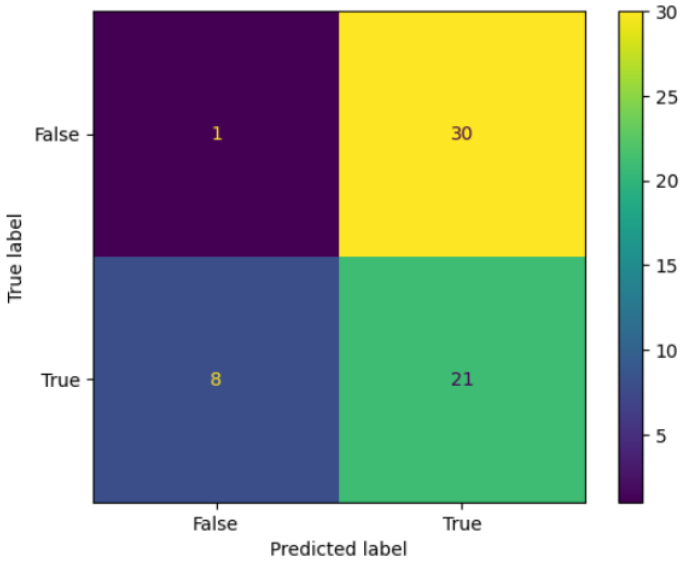

Figure 19: Matriz de confusão usando o random pruning.

Pruning Lowest Values

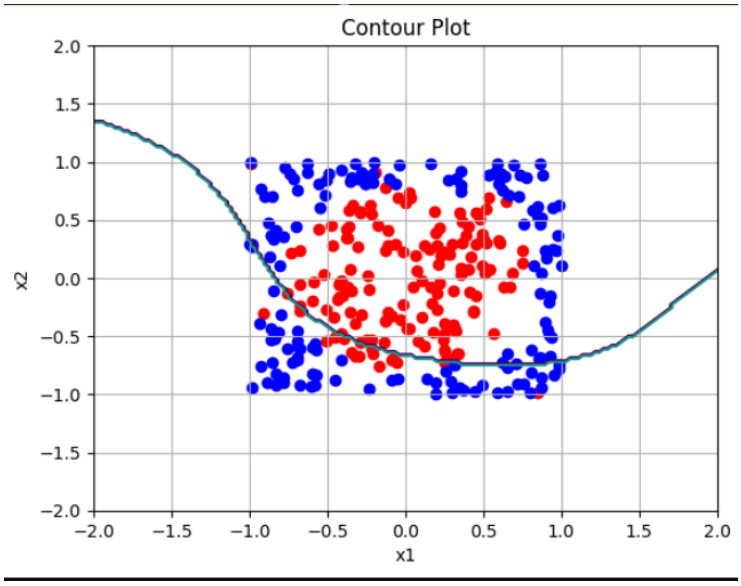

Figure 20: Gráfico de separação usando o pruning lowest value.

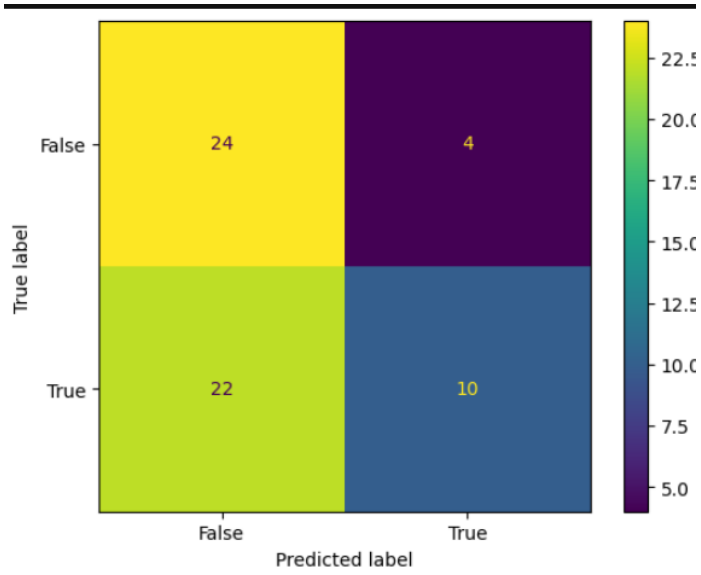

Figure 21: Matriz de confusão usando o pruning lowest value.

Breast Cancer

Ridge Regression

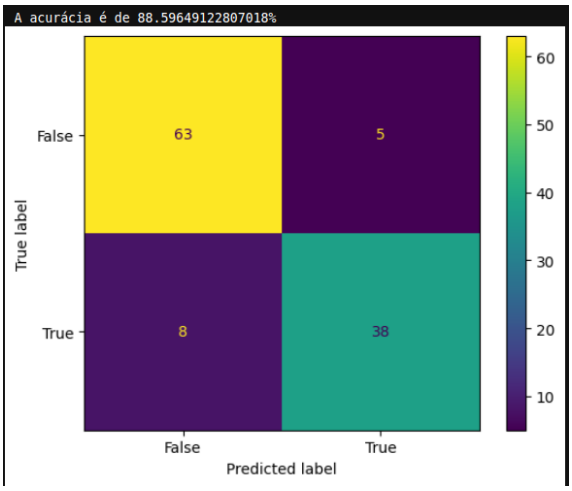

Figure 22: Matriz de confusão na base de dados Breast Cancer usando o Ridge Regression.

Random Pruning

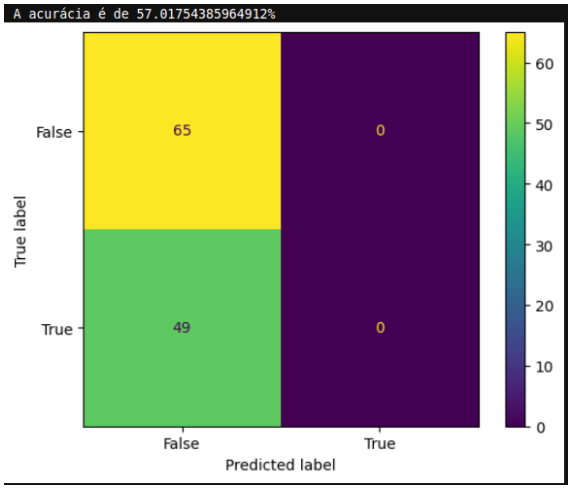

Figure 23: Matriz de confusão na base de dados Breast Cancer usando o random pruning.

Pruning Lowest Values

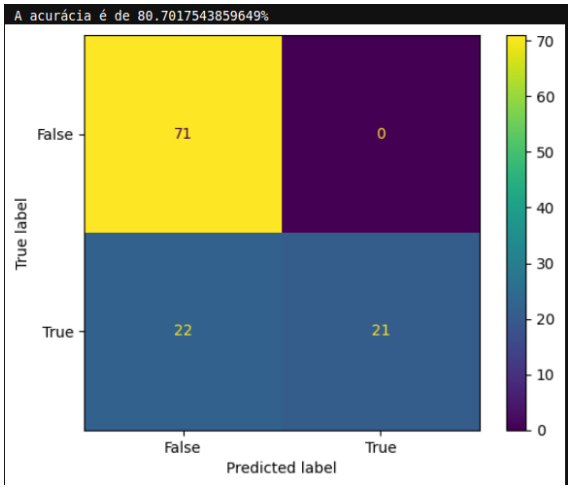

Figure 24: Matriz de confusão na base de dados Breast Cancer usando o pruning lowest value.

Ao analisar as melhores situações para os 3 diferentes modelos com 100 neurônios na camada intermediária é possível concluir que o modelo com a regularização L2 contém as melhores acurácias e superfícies de separação. O modelo com a poda aleatória contém as piores superfícies de separação. O melhor valor de λ foi variável em todas as bases de dados, com valores variando entre 0.2 e 0.7. O valor do Keep rate também foi variável, mas para um intervalo com valores maiores, indo de 0.7 ate 0.99. Isso indica que um número maior de pesos não nulos influencia consideravelmente a boa performance do modelo.

Ao reduzir o número de neurônios, pela metade e por um quarto, a acurácia e as superfícies de todos os 3 modelos também caem consideravelmente. O tempo de treinamento e teste também cai, pois o modelo fica menos complexo, necessitando realizar menos calculos computacionais.

Ao aumentar o número de neurônios, o treinamento e o teste demoram mais, e a acurácia aumenta um pouco.

Agora, iremos avaliar o desempenho dos 3 modelos para as bases de dados de regressão. Inicialmente uma função senoidal será aproximada, depois os modelos serão aplicados para as bases de dados Boston House e Wine Quality.

0.1 Aproximação senoidal

Ridge Regression

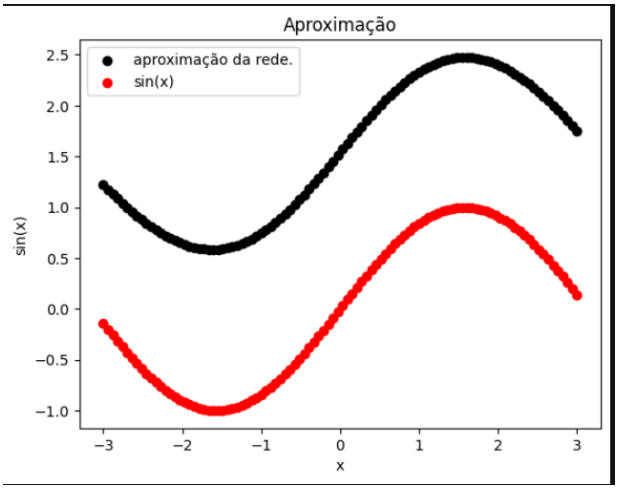

Figure 25: Aproximação senoidal modelo com regularização L2.

Random Pruning

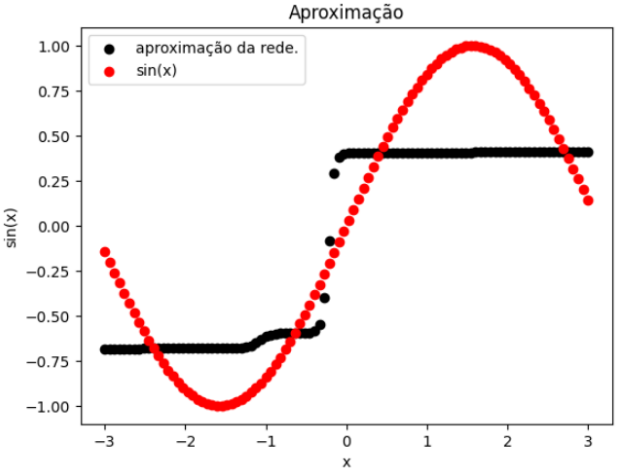

Figure 26: Aproximação senoidal modelo com o random pruning.

Pruning Lowest Values

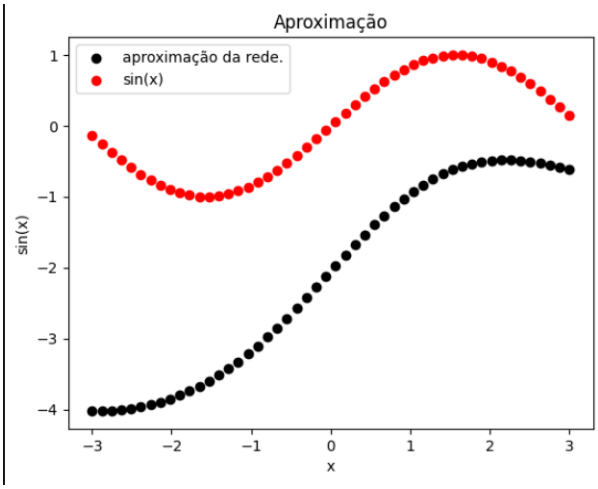

Figure 27: Aproximação senoidal modelo com o pruning lowest value.

0.2 Boston House

Ridge Regression

```
[68]: p_n = 100
      lam_val = np.arange(start = 0.1, stop = 0.99, step = 0.01)
      best_lam, lowest_MSE, arr_ret = techniques.grid_searchCV_L2(xin = np.array(x), yin = np.array(y), p = p_n, lam = lam_val, CV_groups = 20, classification

      The model with lowest MSE is : 11.030652552715175
      The model parameters with lowest MSE is using lambda : 0.29999999999999993
```

Figure 28: MSE do modelo com regularização L2.

Random Pruning

```
[11]: p_n = 100
      keep_val = np.arange(start = 0.1, stop = 0.99, step = 0.01)
      best_keep, lowest_MSE, arr_ret = techniques.grid_searchCV_pruning(xin = np.array(x), yin = np.array(y), p = p_n, keep_rate = keep_val, CV_groups = 2

      The model with lowest MSE is : 101.84814132072297
      The model parameters with lowest MSE is using keep_rate : 0.9799999999999995
```

Figure 29: MSE do modelo com o random pruning.

Pruning Lowest Values

```
[5]: p_n = 100
     keep_val = np.arange(start = 0.1, stop = 0.99, step = 0.01)
     best_keep, lowest_MSE, arr_ret = techniques.grid_searchCV_pruning(xin = np.array(x), yin = np.array(y), p = p_n, keep_rate = keep_val, CV_groups = 20, c

     The model with lowest MSE is : 97.15464771610694
     The model parameters with lowest MSE is using keep_rate : 0.9499999999999995
```

Figure 30: MSE do modelo com o pruning lowest value.

0.3 Wine Quality

Ridge Regression

```
p_n = 100
lam_val = np.arange(start = 0.1, stop = 0.99, step = 0.01)
best_lam, lowest_MSE, arr_ret = techniques.grid_searchCV_L2(xin = np.array(x), yin = np.array(y), p = p_n, lam = lam_val, CV_groups = 10, classification

The model with lowest MSE is : 0.39222934770041873
The model parameters with lowest MSE is using lambda : 0.9399999999999996
```

Figure 31: MSE do modelo com regularização L2.

Random Pruning

```
6]: p_n = 100
    keep_val = np.arange(start = 0.1, stop = 0.99, step = 0.01)
    best_keep, lowest_MSE, arr_ret = techniques.grid_searchCV_pruning(xin = np.array(x), yin = np.array(y), p = p_n, keep_rate = keep_val, CV_groups = 10,

    The model with lowest MSE is : 0.774483785595655
    The model parameters with lowest MSE is using keep_rate : 0.9799999999999995
```

Figure 32: MSE do modelo com o random pruning.

Pruning Lowest Values

```
p_n = 100
keep_val = np.arange(start = 0.5, stop = 0.99, step = 0.01)
best_keep, lowest_MSE, arr_ret = techniques.grid_searchCV_pruning(xin = np.array(x), yin = np.array(y), p = p_n, keep_rate = keep_val, CV_groups = 10, c

The model with lowest MSE is : 0.6859845559579296
The model parameters with lowest MSE is using keep_rate : 0.8300000000000005
```

Figure 33: MSE do modelo com o pruning lowest value.

Portanto, fazendo uma análise sobre as aproximações, é podemos afirmar também que o modelo com regularização L2 também teve um melhor desempenho e, para todos os casos, é o tipo de regularização que leva ao menor MSE. A poda aleatória é, visualmente, a pior aproximação para a função senoidal, enquanto a poda dos menores valores aproxima bem a curva senoidal.

Conclusão

Após a realização de todos os treinamentos e testes para os diferentes modelos, é possível extrair conclusões significativas. Primeiramente, observa-se que o método de poda aleatória não se revela como o mais eficiente para realizar o pruning. Isso se deve ao fato de que pesos com maior influência na resposta podem ser inadvertidamente alterados para zero, impactando consideravelmente o resultado final do modelo. Por outro lado, a análise indica que o método de poda dos parâmetros menos influentes é uma abordagem viável. Ao adotar esse método, o modelo não tende a desenvolver viés e evita o overfitting, além de contribuir para a redução do tempo e do custo de teste. Além disso, observa-se que, em todas as instâncias, a aplicação da regularização Ridge Regression resultou em um desempenho superior para o modelo. Essa técnica demonstrou ser eficaz na mitigação de problemas de overfitting, contribuindo assim para a generalização e robustez do modelo.

Em suma, os resultados destacam a importância de escolher cuidadosamente o método de pruning e a regularização adequada para otimizar o desempenho e a eficiência dos modelos de aprendizado de máquina.

Referências

[1] HAYKIN, Simon. Neural Networks: a comprehensive foundation. 2. ed. India: Pearson Education, Inc, 2001.

[2] G. B. Huang, Q. Y. Zhu, and C. K. Siew, "Extreme learning machine: Theory and applications," Neurocomputing, vol. 70, Dec. 2006.

[3] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse, "OP-ELM: Optimally pruned extreme learning machine," IEEE Trans. Neural Netw., vol. 21, no. 1, pp. 158–162, Jan. 2010.

[4] Braga, A P, Carvalho, A P L e Ludermir, T B (2007). Redes neurais artificiais: teoria e aplicações. LTC, Livros Técnicos e Científicos.

[5] CUN, Yann Le; DENKER, John S.; SOLLA, Sara A.. Optimal Brain Damage. Holmdel: Bell Laboratories, 1989.

[6] Canais confiáveis e renomados em plataformas como youtube.

[7] Sites como IBM, Microsoft, Google.