Implementação e Análise de Classificadores Lineares

Introdução: O Panorama dos Classificadores Lineares

A classificação linear representa um conjunto fundamental de técnicas em aprendizado de máquina, amplamente utilizadas para resolver problemas onde o objetivo é atribuir instâncias de dados a uma de duas ou mais categorias distintas. Estes métodos são caracterizados pela construção de um limite de decisão linear no espaço de características, separando as diferentes classes. Dentro deste domínio, o algoritmo Perceptron se destaca como uma das arquiteturas mais simples e pioneiras de redes neurais artificiais, servindo como um ponto de partida essencial para aqueles que não estão familiarizados com o aprendizado de máquina [1]. Sua simplicidade não diminui sua importância histórica; o Perceptron lançou as bases para avanços significativos na inteligência artificial e no aprendizado de máquina [2]. Compreender o Perceptron não apenas fornece uma base sólida nos princípios básicos do aprendizado profundo, mas também ajuda a contextualizar modelos mais avançados [1]. De fato, o Perceptron é considerado um precursor das redes neurais profundas, destacando sua linhagem direta e conexão fundamental com as metodologias modernas de aprendizado profundo [3]. Para aqueles que buscam expandir sua expertise neste campo, o estudo do Perceptron, especialmente em sua forma de camada única, oferece um caminho claro para entender como combinar e ponderar entradas para determinar uma saída [1].

Explorando os Fundamentos do Algoritmo Perceptron

Em aprendizado de máquina, o Perceptron é formalmente definido como um algoritmo de aprendizado supervisionado para classificadores binários [4]. Sua função principal é atuar como um tomador de decisão de "sim" ou "não", dividindo as entradas em uma de duas categorias com base em se os pesos dos valores de entrada atingem um determinado valor de ativação [1]. Essencialmente, ele opera como um tipo de classificador linear, realizando suas previsões com base em uma função preditora linear que combina um conjunto de pesos com o vetor de características [4]. O funcionamento interno do Perceptron gira em torno de alguns princípios fundamentais. Primeiro, cada característica de entrada recebe um peso associado que indica sua importância na determinação da saída [2]. Esses pesos estabelecem o quanto cada entrada contribui para a decisão final do Perceptron. Em segundo lugar, o algoritmo calcula uma soma ponderada das características de entrada, multiplicando cada entrada pelo seu peso correspondente e somando-os [2]. Essa soma ponderada é então comparada a um limiar, que muitas vezes é incorporado por um termo de bias [1]. O termo de bias permite que o limite de decisão seja deslocado, proporcionando maior flexibilidade ao modelo [2]. A função de ativação, tipicamente uma função degrau ou função limiar, determina a saída do Perceptron com base nessa comparação [2]. Se a soma ponderada das entradas exceder o limiar, o Perceptron produz uma saída (por exemplo, 1); caso contrário, produz outra saída (por exemplo, 0 ou -1) [1].

O processo de aprendizagem do Perceptron é iterativo e se baseia na regra de aprendizagem

do Perceptron. Inicialmente, os pesos são definidos com valores aleatórios pequenos ou zero [4]. Para cada exemplo no conjunto de treinamento, o Perceptron faz uma previsão e, se essa previsão estiver incorreta, os pesos são ajustados [2]. O objetivo é minimizar a diferença entre a saída prevista e a saída real [5]. A regra de atualização de peso normalmente envolve ajustar os pesos na direção que reduz o erro para o exemplo atual [2]. A magnitude desses ajustes é controlada por uma taxa de aprendizado (denotada por η, eta, ou α, alfa), que é um hiperparâmetro crucial que afeta a velocidade e a estabilidade do aprendizado [4]. Geometricamente, o Perceptron pode ser interpretado como a aprendizagem de um limite de decisão – uma linha em duas dimensões ou um hiperplano em dimensões superiores – que separa os pontos de dados de diferentes classes [4]. Este limite de decisão é definido pela equação (w^T x + b = 0), onde (w) representa o vetor de pesos, (x) o vetor de características de entrada e (b) o termo de bias [6]. O termo de bias tem o efeito de deslocar a posição desse limite [4].

O Perceptron como um Classificador Linear

A essência do Perceptron como um classificador linear reside na sua capacidade de aprender uma função linear que separa instâncias de diferentes classes [9]. Matematicamente, o limite de decisão aprendido pelo Perceptron é representado por um hiperplano no espaço n-dimensional, onde n é o número de características de entrada [4]. No caso de dados de entrada bidimensionais, este hiperplano se manifesta como uma linha reta [4]. Este hiperplano divide o espaço de características em duas regiões distintas, cada uma correspondendo a uma das duas classes para as quais o Perceptron é projetado para classificar [6].

A capacidade do Perceptron de separar dados depende crucialmente do conceito de separabilidade linear. Diz-se que um conjunto de dados é linearmente separável se os pontos de dados pertencentes a diferentes classes puderem ser perfeitamente divididos por um único hiperplano linear [1]. Uma limitação fundamental do Perceptron de camada única é que ele só é capaz de aprender padrões que são linearmente separáveis [1]. Por exemplo, conjuntos de dados que representam operações lógicas como AND e OR são linearmente separáveis, e um Perceptron pode aprender a classificá-los com precisão [11]. No entanto, operações lógicas como XOR não são linearmente separáveis em seu espaço de características original, e um único Perceptron não pode aprender um limite de decisão linear para separá-las [11]. Essa limitação destaca a necessidade de arquiteturas mais complexas, como os Perceptrons Multicamadas (MLPs), para lidar com dados que não podem ser separados por um único hiperplano linear [1]. Os MLPs incorporam camadas ocultas e funções de ativação não lineares, permitindo-lhes aprender e resolver problemas de classificação mais sofisticados que envolvem classificações não lineares [1].

Implementando o Perceptron do Zero com NumPy

A implementação do algoritmo Perceptron do zero usando a biblioteca NumPy em Python

envolve várias etapas principais. Primeiro, é necessário definir uma classe Perceptron para encapsular a funcionalidade do algoritmo [11]. Essa classe normalmente inclui um método construtor (__init__) para inicializar os pesos e a taxa de aprendizado [11]. Os pesos, incluindo o bias, são frequentemente inicializados com pequenos valores aleatórios para quebrar a simetria e facilitar o aprendizado [11]. A taxa de aprendizado (alpha ou eta) é um hiperparâmetro que controla o tamanho do passo durante as atualizações de peso [11].

Em seguida, um método para a função de ativação, geralmente uma função degrau, precisa ser implementado [11]. Essa função recebe uma entrada e retorna 1 se a entrada for positiva e 0 caso contrário, efetuando a decisão binária [11]. A lógica para calcular a soma ponderada das entradas pode ser implementada usando a função np.dot() do NumPy para realizar a multiplicação de matrizes entre as características de entrada e os pesos [12].

O coração da implementação é o método fit, responsável por treinar o Perceptron nos dados fornecidos [11]. Este método itera sobre um número predefinido de épocas. Uma época representa uma passagem completa por todo o conjunto de dados de treinamento [11]. Dentro de cada época, o método itera sobre cada ponto de dados de entrada e seu rótulo de classe alvo correspondente. Para cada ponto de dados, ele calcula a saída prevista usando a soma ponderada e a função de ativação [11]. Se a previsão não corresponder ao rótulo alvo, os pesos são atualizados usando a regra delta: (W = W - \alpha \times error \times x), onde (error) é a diferença entre a previsão e o alvo [11]. O termo de bias também é atualizado de forma semelhante [12].

Finalmente, um método predict é implementado para fazer previsões em novos dados [11]. Este método recebe uma matriz de características de entrada, calcula a soma ponderada usando os pesos aprendidos e aplica a função de ativação para retornar os rótulos de classe previstos [11].

Um exemplo de código conciso que demonstra uma implementação do Perceptron usando NumPy para Stochastic Gradient Descent (SGD) é o seguinte [13]:

```
import numpy as np
X = np.array([-2,4,-1],
   [4,1,-1],
   [1, 6, -1],
   [2, 4, -1],
   [6, 2, -1])
y = np.array([-1,-1,1,1,1])
def perceptron_sgd(X, Y):
   w = np.zeros(len(X))
   eta = 1
   epochs = 20
```

```
    for t in range(epochs):
        for i, x in enumerate(X):
            if (np.dot(X[i], w)*Y[i]) <= 0:
                w = w + eta*X[i]*Y[i]
    return w
w = perceptron_sgd(X,y)
print(w)
```

Este código inicializa um vetor de pesos com zeros, define uma taxa de aprendizado e itera sobre os dados de treinamento por um número especificado de épocas. Para cada amostra, se a classificação estiver incorreta, os pesos são atualizados. A função retorna os pesos aprendidos.

Utilizando a Classe Perceptron do Scikit-learn

A biblioteca Scikit-learn oferece uma implementação eficiente do algoritmo Perceptron através da classe Perceptron no módulo sklearn.linear_model [7]. Essa classe fornece uma interface de alto nível para aplicar o Perceptron a tarefas de classificação linear, simplificando o processo para os usuários [8]. É importante notar que a classe Perceptron do Scikit-learn é essencialmente um wrapper para a classe SGDClassifier com parâmetros predefinidos para loss (definido como "perceptron"), learning_rate (definido como "constant") e eta0 (taxa de aprendizado inicial, definida como 1.0) e penalty (definido como None por padrão) [19].

Para usar a classe Perceptron para tarefas de classificação binária, as seguintes etapas são normalmente seguidas [8]: Primeiro, a classe Perceptron é importada do módulo sklearn.linear_model. Em seguida, uma instância da classe Perceptron é criada. Durante a criação da instância, vários hiperparâmetros podem ser especificados, como max_iter (o número máximo de épocas de treinamento), eta0 (a taxa de aprendizado inicial) e penalty (o tipo de regularização a ser aplicada, se houver) [8]. Depois que o modelo Perceptron é instanciado, ele é treinado usando o método .fit() passando os dados de treinamento (características e rótulos) como argumentos [8]. Uma vez treinado, o modelo pode ser usado para fazer previsões sobre novos dados usando o método .predict() [8].

Além da classificação binária, a classe Perceptron do Scikit-learn também pode ser usada para tarefas de classificação multiclasse [8]. Isso é conseguido usando estratégias como a abordagem um-contra-todos (OvA), onde um classificador binário é treinado para cada classe versus todas as outras classes [8]. O Scikit-learn lida com essa adaptação internamente quando o número de classes no conjunto de dados de destino é maior que dois.

Aqui está um exemplo de como usar a classe Perceptron para classificação binária usando o conjunto de dados de câncer de mama do Scikit-learn [8]:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score

# Carregar o conjunto de dados de câncer de mama
data = load_breast_cancer()
X = data.data
y = data.target

# Dividir o conjunto de dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Criar um modelo Perceptron
clf = Perceptron(max_iter=1000, eta0=0.1, random_state=42)

# Treinar o modelo
clf.fit(X_train, y_train)

# Fazer previsões no conjunto de teste
y_pred = clf.predict(X_test)

# Avaliar a precisão do modelo
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Este exemplo demonstra as etapas essenciais de carregamento de dados, divisão em conjuntos de treinamento e teste, criação e treinamento de um modelo Perceptron e, finalmente, avaliação de seu desempenho usando a métrica de precisão.

Analisar e Avaliar o Desempenho de Classificadores Lineares

A análise e avaliação do desempenho de classificadores lineares são etapas cruciais no processo de aprendizado de máquina. Várias métricas são comumente usadas para quantificar o quão bem um classificador está executando.

A **matriz de confusão** é uma tabela que resume o desempenho de um classificador, mostrando o número de verdadeiros positivos (TP), verdadeiros negativos (TN), falsos positivos (FP) e falsos negativos (FN) [8]. Ela fornece uma visão detalhada dos tipos de erros que o modelo está cometendo.

A **acurácia** é uma métrica simples que representa a proporção de instâncias corretamente classificadas em relação ao número total de instâncias [8]. É calculada como ((TP + TN) / (TP +

TN + FP + FN)). Embora seja uma métrica amplamente utilizada, a acurácia pode ser enganosa em conjuntos de dados desbalanceados, onde o número de instâncias em diferentes classes é significativamente diferente.

A **precisão** mede a proporção de instâncias classificadas como positivas que são realmente positivas [8]. É calculada como (TP / (TP + FP)). A precisão é importante quando o objetivo é minimizar falsos positivos. Por exemplo, em diagnósticos médicos, um falso positivo pode levar a um tratamento desnecessário.

O **recall** (também conhecido como sensibilidade ou taxa de verdadeiros positivos) mede a proporção de instâncias positivas reais que foram corretamente identificadas pelo classificador [8]. É calculado como (TP / (TP + FN)). O recall é importante quando o objetivo é minimizar falsos negativos. Por exemplo, na detecção de fraudes, um falso negativo significa que uma transação fraudulenta passa despercebida.

O **F1-score** é a média harmônica da precisão e do recall [8]. É calculado como (2 \times (Precision \times Recall) / (Precision + Recall)). O F1-score fornece uma medida equilibrada do desempenho de um classificador quando há uma necessidade de equilibrar precisão e recall.

Em Python, a biblioteca Scikit-learn fornece funções convenientes no módulo sklearn.metrics para calcular essas métricas de avaliação [8]. Isso inclui funções como confusion_matrix, accuracy_score, precision_score, recall_score e f1_score. Essas funções podem ser aplicadas às saídas previstas de um modelo Perceptron (implementado com NumPy ou Scikit-learn) e aos rótulos verdadeiros para obter uma avaliação quantitativa de seu desempenho.

Aplicando Métricas de Avaliação a Modelos Perceptron

Para demonstrar como aplicar as métricas de avaliação mencionadas acima a modelos Perceptron implementados com NumPy e Scikit-learn, podemos usar o exemplo de classificação binária com o conjunto de dados de câncer de mama.

Primeiro, vamos treinar um modelo Perceptron usando a classe Perceptron do Scikit-learn e avaliar seu desempenho. O código abaixo continua do exemplo anterior:

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

# Fazer previsões no conjunto de teste
y_pred = clf.predict(X_test)

# Calcular a matriz de confusão
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', cm)
```

```
# Calcular acurácia
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Calcular precisão
precision = precision_score(y_test, y_pred)
print(f'Precision: {precision:.2f}')

# Calcular recall
recall = recall_score(y_test, y_pred)
print(f'Recall: {recall:.2f}')

# Calcular F1-score
f1 = f1_score(y_test, y_pred)
print(f'F1-score: {f1:.2f}')
```

A execução desse código produzirá a matriz de confusão e os valores de acurácia, precisão, recall e F1-score para o modelo Perceptron treinado no conjunto de dados de câncer de mama.

Para um modelo Perceptron implementado com NumPy, o processo de avaliação seria semelhante. Primeiro, o modelo seria treinado usando a implementação NumPy. Em seguida, previsões seriam feitas no conjunto de teste. Finalmente, as mesmas funções de métricas de avaliação do sklearn.metrics seriam usadas para calcular o desempenho do modelo comparando as previsões com os rótulos verdadeiros.

Por exemplo, assumindo que temos uma implementação NumPy do Perceptron e o treinamos em X_train e y_train, e o método predict do nosso modelo NumPy retorna previsões em X_test como y_pred_numpy, podemos avaliar seu desempenho da mesma forma:

```
# Assumindo que 'perceptron_numpy' é nossa implementação NumPy e já está treinada
y_pred_numpy = perceptron_numpy.predict(X_test)

cm_numpy = confusion_matrix(y_test, y_pred_numpy)
print('Confusion Matrix (NumPy Implementation):\n', cm_numpy)
accuracy_numpy = accuracy_score(y_test, y_pred_numpy)
print(f'Accuracy (NumPy Implementation): {accuracy_numpy:.2f}')
precision_numpy = precision_score(y_test, y_pred_numpy)
print(f'Precision (NumPy Implementation): {precision_numpy:.2f}')
recall_numpy = recall_score(y_test, y_pred_numpy)
print(f'Recall (NumPy Implementation): {recall_numpy:.2f}')
f1_numpy = f1_score(y_test, y_pred_numpy)
```

```
print(f'F1-score (NumPy Implementation): {f1_numpy:.2f}')
```

Essa abordagem permite uma comparação direta do desempenho entre as implementações NumPy e Scikit-learn do algoritmo Perceptron no mesmo conjunto de dados e com as mesmas métricas de avaliação.

Vantagens e Desvantagens do Algoritmo Perceptron

O algoritmo Perceptron oferece várias vantagens notáveis. Uma de suas principais forças reside na sua **simplicidade** e **facilidade de implementação** [1]. Sua estrutura e o processo de aprendizagem são conceitualmente diretos, tornando-o um excelente ponto de partida para entender os fundamentos das redes neurais e da classificação linear [1]. Para problemas de **classificação binária** onde os dados são **linearmente separáveis**, o Perceptron pode ser bastante **eficaz** [1]. Além disso, o Perceptron é **computacionalmente eficiente**, especialmente para conjuntos de dados grandes, pois processa cada instância de treinamento individualmente para atualizar os pesos [5]. Sua eficiência o torna adequado para tarefas de classificação em tempo real, onde a velocidade é crucial, desde que o problema seja linearmente separável.

No entanto, o Perceptron também possui desvantagens significativas. Sua principal limitação é a **incapacidade de lidar com dados que não são linearmente separáveis** em sua forma básica de camada única [1]. Em tais casos, o algoritmo pode não convergir para uma solução ou pode produzir classificações ruins. Outra desvantagem é sua **sensibilidade à ordem de apresentação dos dados de treinamento**, embora o uso do Stochastic Gradient Descent (SGD) ajude a mitigar esse problema, introduzindo aleatoriedade no processo de treinamento. Além disso, o Perceptron básico não fornece **estimativas de probabilidade** para suas previsões, o que é uma característica desejável em muitas aplicações onde a confiança na previsão é importante. Em comparação com outros classificadores lineares, como a Regressão Logística e as Máquinas de Vetores de Suporte (SVMs), o Perceptron pode ter um desempenho inferior em conjuntos de dados que não são perfeitamente linearmente separáveis [14]. A Regressão Logística, por exemplo, fornece probabilidades e geralmente funciona melhor em dados não perfeitamente linearmente separáveis. As SVMs, por outro lado, visam encontrar o hiperplano de separação ótimo com a maior margem, muitas vezes levando a uma melhor generalização.

Cenários de Eficácia e Limitações do Perceptron

O algoritmo Perceptron se destaca particularmente em cenários onde o problema de classificação binária envolve dados que são conhecidos ou suspeitos de serem linearmente separáveis [1]. Em tais situações, sua simplicidade e eficiência computacional o tornam uma escolha prática. Por exemplo, em tarefas simples de classificação de padrões onde as características podem ser divididas por um limite linear, o Perceptron pode fornecer resultados

rápidos e razoáveis. Além disso, o Perceptron serve como um **componente fundamental** para entender arquiteturas de redes neurais mais complexas e conceitos de aprendizado profundo [1]. Aprender sobre o Perceptron fornece uma base intuitiva para compreender como as redes neurais processam informações em camadas e tomam decisões. Em aplicações onde a velocidade e a simplicidade são primordiais, como em certas tarefas de classificação em tempo real, o Perceptron pode ser eficaz, desde que a separabilidade linear seja atendida.

No entanto, existem vários cenários onde o Perceptron pode não ser a melhor escolha [1]. Problemas que envolvem **dados não linearmente separáveis** são um desafio significativo para o Perceptron de camada única. Para tais problemas, modelos mais complexos capazes de aprender limites de decisão não lineares, como Perceptrons Multicamadas (MLPs), máquinas de vetores de suporte com kernels não lineares ou árvores de decisão, são mais adequados. Além disso, em aplicações onde é necessário ter uma medida de **confiança** nas previsões sob a forma de estimativas de probabilidade, o Perceptron não é ideal, pois sua saída é uma decisão binária direta sem probabilidades associadas. Conjuntos de dados com **alta dimensionalidade** também podem apresentar desafios para o Perceptron básico. Embora o Perceptron possa ser aplicado a dados de alta dimensão, ele não incorpora mecanismos inerentes para lidar com problemas comuns em tais cenários, como a necessidade de escalonamento de características e regularização para evitar overfitting.

Explorando Possíveis Extensões e Variações do Algoritmo Perceptron

Embora o Perceptron de camada única tenha limitações, várias extensões e variações foram desenvolvidas para superar algumas dessas restrições. Uma extensão significativa é o **Perceptron Multicamadas (MLP)**, que incorpora uma ou mais camadas ocultas de neurônios entre a camada de entrada e a camada de saída [1]. Ao usar funções de ativação não lineares nessas camadas ocultas, os MLPs podem aprender limites de decisão não lineares e, portanto, lidar com dados que não são linearmente separáveis [1]. Essa capacidade torna os MLPs adequados para uma gama muito mais ampla de problemas de classificação e regressão.

Outra variação é o **Perceptron com Votação** [7]. Em vez de manter um único vetor de pesos, o Perceptron com Votação mantém vários vetores de pesos que foram aprendidos durante o processo de treinamento. A previsão final é feita combinando as previsões desses múltiplos classificadores usando um esquema de votação ponderada. Essa abordagem pode melhorar a robustez e a precisão do Perceptron, especialmente em cenários com dados complexos.

O **Perceptron Kernelizado** é outra extensão que permite que o Perceptron lide com dados não linearmente separáveis [7]. Ele emprega funções kernel para mapear os dados de entrada em um espaço de características de dimensão superior onde eles podem se tornar linearmente separáveis. Ao usar kernels como funções polinomiais ou funções de base radial (RBF), o Perceptron Kernelizado pode aprender limites de decisão complexos no espaço de entrada original. Essa técnica conecta o Perceptron aos métodos baseados em kernel, como as

máquinas de vetores de suporte.

Essas extensões e variações expandem a aplicabilidade do algoritmo Perceptron, permitindo que ele resolva problemas mais complexos do que o Perceptron de camada única original. O MLP, em particular, forma a base para muitas arquiteturas de aprendizado profundo que têm alcançado sucesso notável em várias tarefas, incluindo visão computacional, processamento de linguagem natural e reconhecimento de fala.

Conclusão: Refletindo sobre o Papel do Perceptron no Aprendizado de Máquina

Em resumo, o algoritmo Perceptron é um classificador linear fundamental que opera com base no princípio de aprender um hiperplano para separar diferentes classes de dados. Sua simplicidade o torna uma ferramenta valiosa para entender os conceitos básicos de classificação linear e redes neurais. Através da implementação prática usando bibliotecas Python como NumPy e Scikit-learn, torna-se evidente sua capacidade de resolver problemas de classificação binária linearmente separáveis. A avaliação do desempenho do Perceptron usando métricas como matriz de confusão, acurácia, precisão, recall e F1-score fornece insights sobre sua eficácia em diferentes cenários.

Embora o Perceptron ofereça vantagens como simplicidade, facilidade de implementação e eficiência para dados linearmente separáveis, ele também tem limitações, principalmente sua incapacidade de lidar com dados não linearmente separáveis em sua forma básica. Essa limitação levou ao desenvolvimento de extensões como o Perceptron Multicamadas, o Perceptron com Votação e o Perceptron Kernelizado, que aumentam sua capacidade de resolver problemas mais complexos.

O estudo do Perceptron continua relevante no campo do aprendizado de máquina. Ele serve como um bloco de construção essencial para compreender arquiteturas de redes neurais mais avançadas e fornece uma base conceitual para técnicas de classificação linear. Sua importância histórica e seu papel como precursor do aprendizado profundo garantem que o Perceptron permaneça um tópico fundamental para estudantes e profissionais de aprendizado de máquina.

Referências citadas

1. What Is Perceptron? | Coursera, acessado em março 24, 2025, https://www.coursera.org/articles/what-is-perceptron
2. What is Perceptron | The Simplest Artificial neural network - GeeksforGeeks, acessado em março 24, 2025, https://www.geeksforgeeks.org/what-is-perceptron-the-simplest-artificial-neural-network/
3. Perceptron: Basic Principles of Deep Neural Networks - Cardiovascular Prevention and Pharmacotherapy, acessado em março 24, 2025, https://www.e-

jcpp.org/journal/view.php?doi=10.36011/cpp.2021.3.e9
4. Perceptron - Wikipedia, acessado em março 24, 2025, https://en.wikipedia.org/wiki/Perceptron
5. Understanding the Perceptron: A Foundation for Machine Learning Concepts, acessado em março 24, 2025, https://www.lucentinnovation.com/blogs/technology-posts/understanding-the-perceptron
6. stats.stackexchange.com, acessado em março 24, 2025, https://stats.stackexchange.com/questions/410952/why-perceptron-is-linear-classifier#:~:text=It%20is%20called%20a%20linear,wtx%3Eb%7D.
7. Perceptron class in Sklearn - GeeksforGeeks, acessado em março 24, 2025, https://www.geeksforgeeks.org/sklearn-perceptron/
8. Perceptron Algorithm for Classification using Sklearn - GeeksforGeeks, acessado em março 24, 2025, https://www.geeksforgeeks.org/sklearn-classification-using-perceptron/
9. Linear Classification and Perceptron - University of Colorado Boulder, acessado em março 24, 2025, https://cmci.colorado.edu/classes/INFO-4604/files/slides-3_perceptron.pdf
10. Perceptron in scikit-learn - FutureLearn, acessado em março 24, 2025, https://www.futurelearn.com/info/courses/machine-learning-for-image-data/0/steps/365511
11. Implementing the Perceptron Neural Network with Python ..., acessado em março 24, 2025, https://pyimagesearch.com/2021/05/06/implementing-the-perceptron-neural-network-with-python/
12. Perceptron: Building it from scratch in python | by Becaye Baldé ..., acessado em março 24, 2025, https://medium.com/@becaye-balde/perceptron-building-it-from-scratch-in-python-15716806ef64
13. A Perceptron in just a few Lines of Python Code, acessado em março 24, 2025, https://maviccprp.github.io/a-perceptron-in-just-a-few-lines-of-python-code/
14. Perceptron Algorithm for Classification in Python - MachineLearningMastery.com, acessado em março 24, 2025, https://machinelearningmastery.com/perceptron-algorithm-for-classification-in-python/
15. Training the Perceptron with Scikit-Learn and TensorFlow | QuantStart, acessado em março 24, 2025, https://www.quantstart.com/articles/training-the-perceptron-with-scikit-learn-and-tensorflow/
16. Calculate the Decision Boundary of a Single Perceptron; Visualizing Linear Separability, acessado em março 24, 2025, https://thomascountz.com/2018/04/13/calculate-decision-boundary-of-perceptron
17. Building a Multi-Layer Perceptron from Scratch with NumPy | by Kassem - Medium, acessado em março 24, 2025, https://elcaiseri.medium.com/building-a-multi-layer-perceptron-from-scratch-with-numpy-e4cee82ab06d
18. Implementing a perceptron using numpy [closed] - Stack Overflow, acessado em março 24, 2025, https://stackoverflow.com/questions/79442381/implementing-a-perceptron-using-numpy
19. Perceptron — scikit-learn 1.6.1 documentation, acessado em março 24, 2025, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html