AULA 09 Exercício teórico Aprendizado de máquina – Parte 2

October 24, 2023

Vitor Albuquerque de Paula

1 Seja o seguinte dataset:

Dataset:

Nome	Febre	Enjôo	Manchas	Dores	Diagnóstico
João	Sim	Sim	Pequenas	Sim	Doente
Pedro	Não	Não	Grandes	Não	Saudável
Maria	Sim	Sim	Pequenas	Não	Saudável
José	Sim	Não	Grandes	Sim	Doente
Ana	Sim	Não	Pequenas	Sim	Saudável
Leila	Não	Não	Grandes	Sim	Doente

Ensinar uma rede do tipo Perceptron a distinguir: - Pacientes potencialmente saudáveis - Pacientes potencialmente doentes

Testar a rede para novos casos: - (Luis, não, não, pequenas, sim) - (Laura, sim, sim, grandes, sim) Pode considerar =0.5, limiar = -0,5, o valor 1 para 'sim'/'grandes' e 0 para 'não'/'pequenas' no dataset .

[1]:
```
import numpy as np  # Importando a biblioteca numpy para operações com vetores
↪e matrizes

# Função de ativação
def step_function(u):
    return 1 if u >= 0.5 else 0  # Retorna 1 se u for maior ou igual a 0.5,
↪caso contrário, retorna 0

# Treinamento do Perceptron
def train_perceptron(data, labels, eta, threshold, max_epochs):
    w = np.zeros(data.shape[1])  # Inicializa os pesos com zeros
    bias = 0.5  # Inicializa o viés com 0.5

    for epoch in range(max_epochs):  # Loop para um máximo de épocas definido
        errors = 0  # Contador para erros no conjunto de treinamento
```

```
        for xi, expected in zip(data, labels):  # Itera pelos dados e seus␣
 ↪respectivos rótulos
            result = np.dot(xi, w) + bias  # Calcula o resultado como o produto␣
 ↪interno dos dados e pesos, adicionando o viés
            error = expected - step_function(result)  # Calcula o erro

            w += eta * error * xi  # Atualiza os pesos
            bias += eta * error  # Atualiza o viés

            errors += abs(error)  # Conta os erros

        if errors == 0:  # Se não houver erros, interrompe o treinamento
            break

    return w, bias  # Retorna os pesos e o viés treinados

# Codificar os dados
data = np.array([[1, 1, 0, 1],  # João
                 [0, 0, 1, 0],  # Pedro
                 [1, 1, 0, 0],  # Maria
                 [1, 0, 1, 1],  # José
                 [1, 0, 0, 1],  # Ana
                 [0, 0, 1, 1]]) # Leila

labels = np.array([1, 0, 0, 1, 0, 1])  # 1 para Doente, 0 para Saudável

# Treinamento
weights, bias = train_perceptron(data, labels, 0.5, 0.5, 1000)  # Treina o␣
 ↪perceptron com os dados, taxa de aprendizado de 0.5, limiar de 0.5 e um␣
 ↪máximo de 1000 épocas

# Função de predição
def predict(inputs, weights, bias):
    result = np.dot(inputs, weights) + bias  # Calcula o resultado para os␣
 ↪dados de entrada usando os pesos e viés
    return step_function(result)  # Retorna a classificação com a função de␣
 ↪ativação

test_data1 = np.array([0, 0, 0, 1])  # Luis
test_data2 = np.array([1, 1, 1, 1])  # Laura

print(predict(test_data1, weights, bias))  # Saída esperada: 0 ou 1
print(predict(test_data2, weights, bias))  # Saída esperada: 0 ou 1
```

```
1
1
```

1.1 Encontre a equação linear que melhor estime as vendas médias semanais (Y) dado um número de clientes (X).

X	907	926	506	741	789	889	874	510	529	420
Y	11.20	11.05	6.84	9.21	9.42	10.08	9.45	6.73	7.24	6.12

X	679	872	924	607	452	729	794	844	1010	621
Y	7.63	9.43	9.46	7.64	6.92	8.95	9.33	10.23	11.77	7.41

Dado que a equação da linha é (Y = mX + b) (onde (m) é a inclinação e (b) é a intercepção no eixo y), usaremos o método dos mínimos quadrados para encontrar os coeficientes (m) e (b).

Passo 1: Calcular as médias de X e Y.

$$\bar{X} = \frac{\sum X}{n}$$

$$\bar{Y} = \frac{\sum Y}{n}$$

Passo 2: Determinar o coeficiente de inclinação (m) usando a fórmula:

$$m = \frac{n(\sum XY) - (\sum X)(\sum Y)}{n(\sum X^2) - (\sum X)^2}$$

Passo 3: Usar as médias de X e Y do Passo 1 para determinar a intercepção no eixo y, (b):

$$b = \bar{Y} - m\bar{X}$$

Calculando os valores:

Primeiro, vamos listar todos os valores de X e Y:

$$X = [907, 926, 506, 741, 789, 889, 874, 510, 529, 420, 679, 872, 924, 607, 452, 729, 794, 844, 1010, 621]$$

$$Y = [11.20, 11.05, 6.84, 9.21, 9.42, 10.08, 9.45, 6.73, 7.24, 6.12, 7.63, 9.43, 9.46, 7.64, 6.92, 8.95, 9.33, 10.23, 11.77, 7.41]$$

Passo 1: Calcular as médias

$$\bar{X} = \frac{15080}{20} = 754$$

$$\bar{Y} = \frac{177.77}{20} = 8.8885$$

Passo 2: Determinar o coeficiente de inclinação (m) Primeiro, vamos calcular os somatórios:

$$\sum X = 15080$$

$$\sum Y = 177.77$$
$$\sum XY = 1422629.8$$
$$\sum X^2 = 11398294$$

Agora, plugando na fórmula:

$$m = \frac{20(1422629.8) - (15080)(177.77)}{20(11398294) - (15080)^2} = 0.0098$$

Passo 3: Calcular (b)

$$b = 8.8885 - 0.0098(754) = 1.5677$$

Então, a equação linear que melhor estima as vendas médias semanais (Y) dado um número de clientes (X) é:

$$Y = 0.0098X + 1.5677$$

2 Pesquise quais dos algoritmos supervisionados vistos em aula para classificação de dados (KNN, Naive Bayes, Árvores de Decisão, SVM, MLP) podem ser adaptados para regressão e como funcionariam nesse caso.

2.1 K-Nearest Neighbors (KNN)

O KNN pode ser facilmente adaptado para regressão, sendo neste caso comumente referido como KNN de regressão. Em vez de retornar a classe mais comum entre os K vizinhos mais próximos, ele retorna a média (ou mediana) dos valores de destino dos K vizinhos mais próximos.

2.2 Naive Bayes

O Naive Bayes não é tipicamente usado para regressão e pode não ser o método mais adequado para tarefas de regressão, pois é fundamentalmente um classificador.

2.3 Árvores de Decisão

As Árvores de Decisão podem ser adaptadas para regressão e, neste caso, são chamadas de Árvores de Regressão. Ao invés de retornar uma classe, uma Árvore de Regressão retorna um valor contínuo, que pode ser a média dos valores de destino das observações em uma folha.

2.4 Support Vector Machines (SVM)

O SVM pode ser adaptado para regressão, sendo comumente referido como Support Vector Regression (SVR). O SVR tenta encontrar uma função que tenha no máximo desvio dos valores de destino reais para todos os pontos de treinamento, enquanto mantém a margem máxima.

2.5 Multilayer Perceptron (MLP)

O MLP é uma rede neural, e as redes neurais podem ser usadas para regressão. Para adaptar o MLP para regressão, é possivel usar uma função de ativação linear na camada de saída e treinar a rede para minimizar o erro quadrático médio, por exemplo.

Essas adaptações são relativamente comuns e são suportadas por muitas bibliotecas de aprendizado de máquina, como scikit-learn.

2.6 Referencias

- KNN para Regressão
- Árvores de Regressão
- Support Vector Regression (SVR)
- MLP para Regressão

Os links fornecem mais informações sobre como esses algoritmos podem ser adaptados para regressão, incluindo exemplos de código usando a biblioteca scikit-learn.