پیاده سازی پرسپترون

این سند یک نمای کلی و جزئیات پیاده سازی را برای یک مدل پرسپترون ساده با استفاده از پایتون ارائه می دهد. Perceptron یک بلوک ساختمانی اساسی در یادگیری ماشین است و به عنوان پایه ای برای معماری شبکه های عصبی پیچیده تر عمل می کند.

1. پیاده سازی:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

class F:
    @staticmethod
    def sign(x):
        return np.where(x >= 0, 1, 0)

class Perceptron:
    def __init__(self, input_size):
        self.weights = np.zeros(input_size)
        self.bias = 0

    def forward(self, input):
        return F.sign(np.dot(input, self.weights) +
self.bias)

class Optimizer:
    def __init__(self, model):
        self.model = model

    def update(self, x, y, y_hat):
        self.model.weights += (y - y_hat) * x
        self.model.bias += y - y_hat

def fit(iterations):
```

```
    n_iter = iterations
    for _ in range(n_iter):
        for x, y_true in zip(X, y):
            y_pred = perceptron.forward(x)
            if y_true != y_pred:
                optimizer.update(x, y_true, y_pred)

# Create a linearly separable dataset
X, y = make_blobs(n_samples=100, n_features=2, centers=2,
random_state=41)

# Initialize Perceptron and Optimizer
perceptron = Perceptron(input_size=2)
optimizer = Optimizer(model=perceptron)

# Train the perceptron
fit(iterations=1000)

# Plot the dataset
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis',
marker='o', edgecolors='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# Plot the dividing line
x_values = np.linspace(min(X[:,0]),max(X[:,0]))
y_values = -(perceptron.bias+(perceptron.weights[0]*
x_values))/perceptron.weights[1]

plt.plot(x_values, y_values, color='red', linestyle='-',
label='The dividing line')

# Show the plot
plt.title('Perceptron')
plt.legend()
plt.show()
```

2. توضیحات:

ابتدا کتابخانه های مورد نیاز import شده اند.

1. کلاس `**F**`:

- این کلاس یک کلاس کمکی است که یک متد استاتیک به نام `sign` دارد.

- متد `sign` از `numpy` استفاده می‌کند تا علامت ورودی‌های عددی را تعیین کند. برای اعداد مثبت یا صفر ۱ و برای اعداد منفی ۰ را باز می‌گرداند.

2. کلاس `**Perceptron**`:

- این کلاس مدل پرسپترون را تعریف می‌کند.

- در متد `__init__` وزن‌ها و bias پرسپترون را مقداردهی اولیه می‌کند.

- متد `forward` ورودی‌ها را به پرسپترون می‌دهد و خروجی را محاسبه می‌کند.

3. کلاس `**Optimizer**`:

- این کلاس به عنوان یک بهینه‌ساز برای به‌روزرسانی وزن‌ها و bias پرسپترون در هر مرحله آموزش عمل می‌کند.

- متد `update` وزن‌ها و bias پرسپترون را با توجه به خطاهای پیش‌بینی به‌روزرسانی می‌کند.

4. تابع `**fit**`:

- این تابع به پرسپترون داده‌ها را می‌دهد و مدل را برای آموزش به‌روزرسانی می‌کند.

- با تکرار مشخص تعداد مراحل آموزش، برای هر داده در داده‌های آموزش خطاهای پیش‌بینی را محاسبه کرده و وزن‌ها و bias پرسپترون را به‌روز می‌کند.

5. ایجاد مجموعه داده:

- این بخش از کتابخانه `scikit-learn` استفاده می‌کند تا یک مجموعه داده خطی جداپذیر را ایجاد کند.

- تابع `make_blobs` یک مجموعه داده با تعداد نمونه‌ها، ویژگی‌ها، مراکز و تنظیمات تصادفی مشخص شده را تولید می‌کند.

6. ایجاد و آموزش مدل پرسپترون:

- یک نمونه از کلاس پرسپترون و بهینه‌ساز ایجاد می‌شود.

- سپس مدل پرسپترون با استفاده از داده‌ها آموزش داده می‌شود.

7. ترسیم نمودار:

- این بخش از کد به کمک `matplotlib` نمودار مجموعه داده‌ها و خط جداکننده پرسپترون را ترسیم می‌کند.

- ابتدا نقاط داده‌ها را با استفاده از `plt.scatter` ترسیم می‌کند.

- سپس خط جداکننده با استفاده از وزن‌ها و bias پرسپترون محاسبه و با استفاده از `plt.plot` ترسیم می‌شود.

- در نهایت با استفاده از `plt.show` نمودار نمایش داده می‌شود.