SVM

20250415_Suport_vector_machine_실습 강의자료, p.15

SVM Gradient Decent Method (GD)

Gradient decent algorithm

① 현재 지점에서 미분을 이용해 gradient 계산

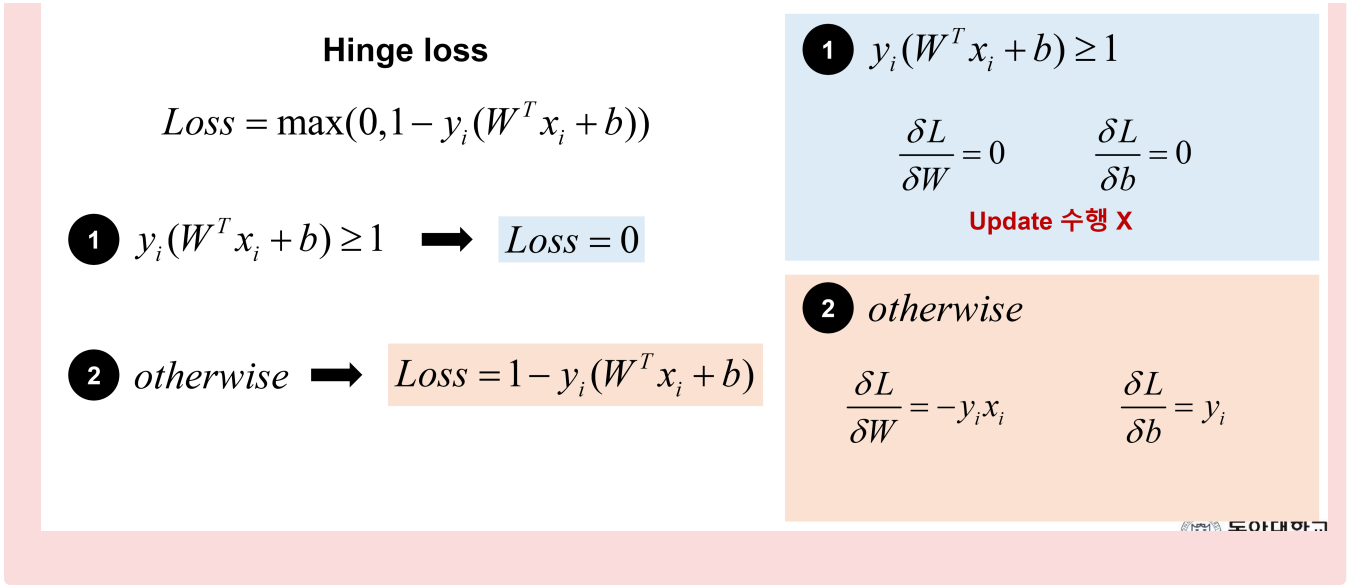
② Gradient에 learning rate를 곱하고
반대 방향으로 weight update

Hinge loss

$$Loss = \max(0, 1 - y_i(W^T x_i + b))$$

1 $y_i(W^T x_i + b) \geq 1$ ➡ $Loss = 0$

2 $otherwise$ ➡ $Loss = 1 - y_i(W^T x_i + b)$



SVM(Support Vector Machine) 개요

SVM은 분류를 위한 지도학습 알고리즘으로, 데이터를 가장 잘 분리하는 최적의 경계(결정 경계)를 찾는 것을 목표로 합니다. 경사 하강법(Gradient Descent)을 통해 최적의 초평면을 찾습니다.

1. 필요한 라이브러리 임포트

데이터 처리와 시각화에 필요한 기본 라이브러리를 가져옵니다.

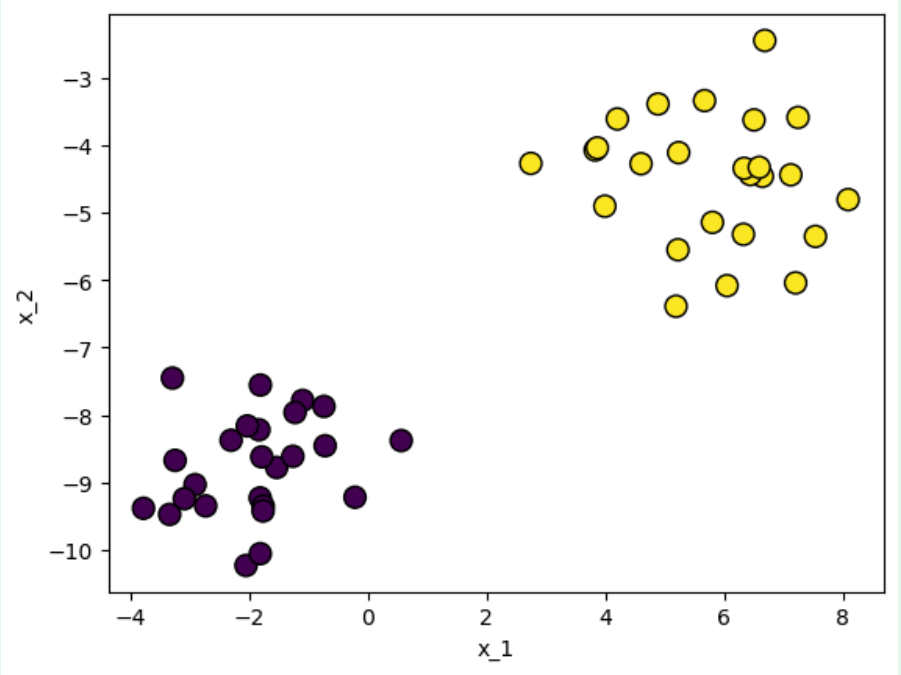
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
```

2. 테스트 데이터 생성 및 시각화

선형 분리 가능한 이진 분류 데이터를 생성하고 시각화합니다.

```
X, y = make_blobs(n_samples=50, n_features=2, centers=2, cluster_std=1.05, random_state=40)
plt.scatter(X[:, 0], X[:, 1], marker='o', c=y, s=100, edgecolor="k", linewidth=1)
plt.xlabel("x_1")
plt.ylabel("x_2")
plt.show()
```

✓ Success



3. SVM 클래스 구현

SVM 알고리즘을 직접 구현한 클래스입니다.

```
class SVM:
    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.weights = None  # 가중치 벡터
        self.bias = None  # 절편

    def fit(self, X, y):
        """
        SVM 모델 학습
        - X: 입력 데이터 (data 개수 x feature 개수)
        - y: 타겟 레이블 (data 개수만큼 -1 또는 1로 이루어진 배열)
        """
        n_samples, n_features = X.shape # n_samples: 데이터 개수

        # 레이블을 -1 또는 1로 변환
        y_modified = np.where(y <= 0, -1, 1)

        # Weight 및 bias 초기화
        self.weights = np.zeros(n_features)  # 가중치 벡터
        self.bias = 0  # 절편

        # 경사 하강법(Gradient Descent) 구현
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                # 조건: y_i(W^T * x_i + b) >= 1 (마진 조건 확인)
                condition = y_modified[idx] * (np.dot(x_i, self.weights) + self.bias) >= 1

                if not condition:  # 마진 조건을 만족하지 않는 경우
                    # 힌지 손실 함수의 그래디언트를 계산하여 가중치 업데이트
                    # ∂L/∂W = -y_i * x_i 에 따라 가중치 업데이트
                    self.weights += self.learning_rate * (y_modified[idx] * x_i)
                    # ∂L/∂b = -y_i 에 따라 바이어스 업데이트
                    self.bias += self.learning_rate * (y_modified[idx])
```

```
    def predict(self, X):
        """
        새로운 데이터에 대한 클래스 예측
        - X: 입력 데이터
        - 반환값: 예측된 클래스 레이블 (-1 또는 1)
        """
        # 결정 함수: w^T * x + b
        linear_output = np.dot(X, self.weights) + self.bias
        # sign 함수로 -1 또는 1로 변환
        return np.sign(linear_output)
```

SVM 핵심 원리

- **목표**: 마진(margin)을 최대화하는 초평면(hyperplane) 찾기
- **초평면 방정식**: w^T * x + b = 0
- **마진 조건**: y_i(W^T * x_i + b) ≥ 1
- **최적화**: 마진 조건을 만족하지 않는 경우에만 가중치와 편향 업데이트

주요 매개변수:

- learning_rate: 학습률 (가중치 업데이트 크기 조절)
- n_iters: 반복 횟수 (전체 데이터셋 반복 학습 횟수)

주의사항

1. SVM에서는 레이블이 반드시 -1과 1이어야 합니다 (0, 1이 아님)
2. 경사 하강법은 마진 조건을 만족하지 않는 샘플(서포트 벡터 후보)에 대해서만 수행합니다
3. 선형 커널만 구현되어 있어 비선형 분류 문제에는 적합하지 않습니다

4. 모델 학습

SVM 모델을 생성하고 데이터로 학습시킵니다.

```
model = SVM()
margin_log = model.fit(X, y)

print(model.weights, model.bias)
```

5. 결정 경계 시각화 함수

SVM의 결정 경계와 마진을 시각화하는 함수를 정의합니다.

```
def get_hyperplane_value(x, w, b, offset):
    return (-w[0] * x - b + offset) / w[1]

def visualize_svm(w, b):
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    plt.scatter(X[:, 0], X[:, 1], marker="o", c=y)

    x0_1 = np.amin(X[:, 0])
    x0_2 = np.amax(X[:, 0])
```

```
    x1_1 = get_hyperplane_value(x0_1, w, b, 0)
    x1_2 = get_hyperplane_value(x0_2, w, b, 0)

    x1_1_m = get_hyperplane_value(x0_1, w, b, -1)
    x1_2_m = get_hyperplane_value(x0_2, w, b, -1)

    x1_1_p = get_hyperplane_value(x0_1, w, b, 1)
    x1_2_p = get_hyperplane_value(x0_2, w, b, 1)

    ax.plot([x0_1, x0_2], [x1_1, x1_2], "y--")
    ax.plot([x0_1, x0_2], [x1_1_m, x1_2_m], "k")
    ax.plot([x0_1, x0_2], [x1_1_p, x1_2_p], "k")

    x1_min = np.amin(X[:, 1])
    x1_max = np.amax(X[:, 1])
    ax.set_ylim([x1_min - 3, x1_max + 3])
    plt.xlabel("x_1")
    plt.ylabel("x_2")
    plt.show()
```

시각화 요소

- 점선(노란색): 결정 경계 (w^T * x + b = 0)
- 실선(검은색): 마진 경계 (w^T * x + b = ±1)
- 마진 경계 사이의 거리: 2/||w|| (여기서 ||w||는 가중치 벡터의 크기)

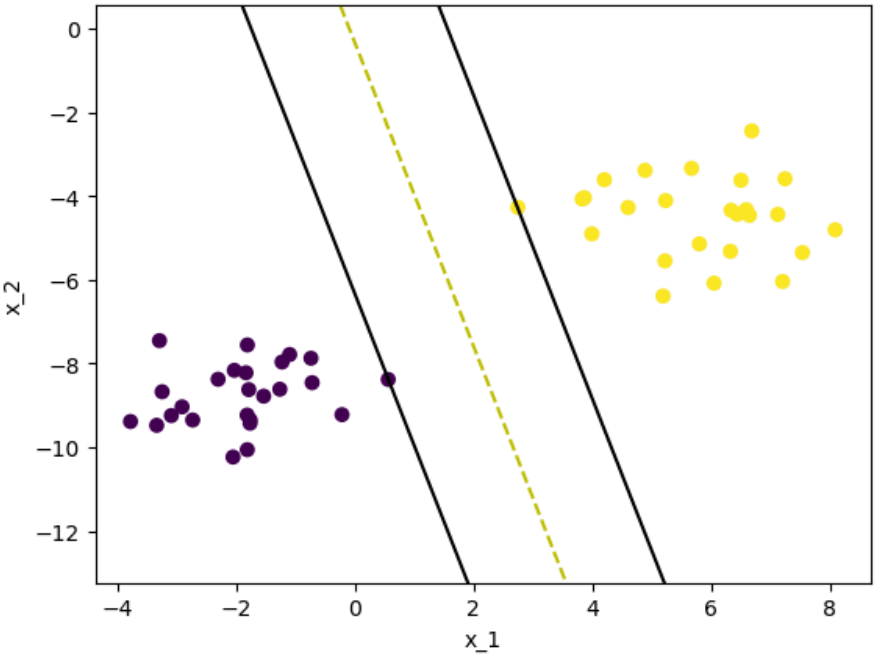
6. 결과 시각화

학습된 모델의 결정 경계와 마진을 시각화합니다.

```
visualize_svm(model.weights, model.bias)
```

✓ SVM 해석

- 결정 경계: 두 클래스를 구분하는 직선(초평면)
- 마진: 결정 경계와 가장 가까운 데이터 점들(서포트 벡터) 사이의 거리
- 목표 달성: 최대 마진 분류기가 성공적으로 학습되었음을 시각적으로 확인



참고사항

- 파란색/주황색 점: 두 클래스의 데이터 포인트
- 노란색 점선: 결정 경계
- 검은색 실선: 마진 경계
- 마진 경계에 가까운 점들이 서포트 벡터(Support Vector)입니다.