機械学習 課題：学習定数を定める

235738B 越後 玲輝

2025 年 5 月 1 日

1. 課題概要

text の Adaline のコードを実行して、学習定数の値をさまざまに試し、最適と思われる値を特定してください。また、その値を選んだ理由も添えてください。

2. Python コード

Listing 1 Adaline 学習率比較コード

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

#クラスを定義AdalineSGD
class AdalineSGD(object):
    def __init__(self, eta=0.01, n_iter=10, shuffle=True, random_state=None):
        self.eta = eta
        self.n_iter = n_iter
        self.w_initialized = False
        self.shuffle = shuffle
        if random_state:
            np.random.seed(random_state)

    def fit(self, X, y):
        self._initialize_weights(X.shape[1])
        self.cost_ = []
        for _ in range(self.n_iter):
            if self.shuffle:
                X, y = self._shuffle(X, y)
            cost = []
            for xi, target in zip(X, y):
                cost.append(self._update_weights(xi, target))
            avg_cost = sum(cost) / len(y)
            self.cost_.append(avg_cost)
        return self

    def partial_fit(self, X, y):
        if not self.w_initialized:
            self._initialize_weights(X.shape[1])
        if y.ravel().shape[0] > 1:
            for xi, target in zip(X, y):
                self._update_weights(xi, target)
        else:
            self._update_weights(X, y)
        return self

    def _shuffle(self, X, y):
        r = np.random.permutation(len(y))
```

```
        return X[r], y[r]

    def _initialize_weights(self, m):
        self.w_ = np.zeros(1 + m)
        self.w_initialized = True

    def _update_weights(self, xi, target):
        output = self.net_input(xi)
        error = (target - output)
        self.w_[1:] += self.eta * xi.dot(error)
        self.w_[0] += self.eta * error
        cost = 0.5 * error**2
        return cost

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        return self.net_input(X)

    def predict(self, X):
        return np.where(self.activation(X) >= 0.0, 1, -1)

iris = load_iris()
X = iris.data[:100, [0, 2]]
y = iris.target[:100]
y = np.where(y == 0, -1, 1)

sc = StandardScaler()
X_std = sc.fit_transform(X)

#学習率ごとの比較
etas = [0.0001, 0.001, 0.01, 0.1, 0.5]
colors = ['r', 'g', 'b', 'c', 'm']

plt.figure(figsize=(10, 6))

for eta, color in zip(etas, colors):
    ada = AdalineSGD(n_iter=15, eta=eta, random_state=1)
    ada.fit(X_std, y)
    plt.plot(range(1, len(ada.cost_) + 1),
             ada.cost_, marker='o', color=color, label=f'eta={eta}')

plt.xlabel('Epochs')
plt.ylabel('Average Cost')
plt.title('AdalineSGD - Learning Rate Comparison')
plt.legend()
plt.grid(True)
plt.show()
```

3. 実験結果

以下の図は、各学習率ごとの平均コスト（SSE: Sum of Squared Errors）の変化を示している。

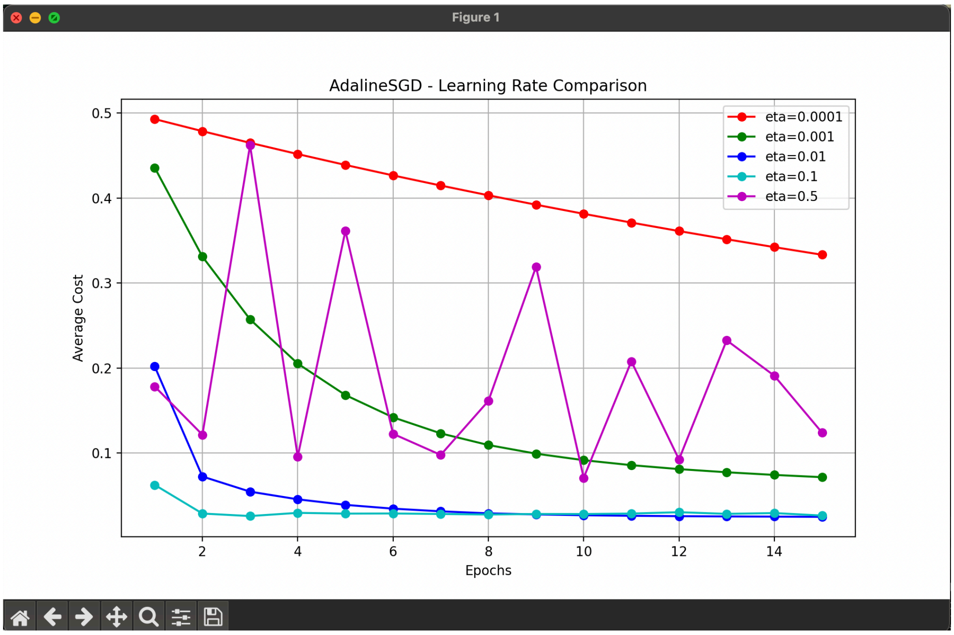

図 1　学習率ごとの平均コスト推移（AdalineSGD）

グラフの描画には、Python ライブラリである `matplotlib.pyplot` を用いた。

各学習率（η）ごとに 15 エポックの学習を実行し、各エポックの平均コスト（SSE）をリストとして記録。

エポック数を横軸、平均コストを縦軸として描画。

4. 各学習率について考察

それぞれの学習率に対して、以下のような学習挙動が見られた。

- $\eta = 0.0001$： 学習率が極めて小さいため、1 回の更新による重みの変化が微小となり、学習が進みにくく、結果として収束に非常に時間がかかる。
- $\eta = 0.001$： 多少改善されたものの、更新幅が小さく、依然として誤差の減少速度は遅い。精度は上がるが効率は悪い。
- $\eta = 0.01$： 重みが過不足なく更新されるため、誤差が安定して減少し、効率よく収束するパターン。
- $\eta = 0.1$： 学習率がやや大きいため、収束は速く見えるものの、最適値の周囲でコストにブレを確認。

- $\eta = 0.5$： 学習率が大きすぎて、重みが極端に更新されることで適切な方向に学習が進まず、むしろ誤差が増加する傾向となった。

これらの結果から、学習率は小さすぎても大きすぎても学習が非効率・不安定となるため、$\eta = 0.01$ のような中間的な値が最も望ましいといえる。

5. 結論

η を 0.0001 から 0.5 まで 5 段階で変化させた。それぞれの学習率について平均コストの推移を観察した結果、$\eta = 0.01$ のときに最も安定して学習が進み、15 エポック以内に誤差が十分に収束した。これに対して、$\eta = 0.0001$ や 0.001 では収束が遅く、$\eta = 0.1$ 以上では誤差が発散または振動する傾向が見られた。したがって、$\eta = 0.01$ が最もバランスの取れた学習定数であり、最適であると判断した。