ソフトウェア工学 レポート課題7-8

2025年5月22日, 6月5日授業分
学籍番号：35714121
名前：福富隆大

課題7-8-1：ソフトウェア設計の原則についての調査

調査対象のコード

以下は、以前に作成したパーセプトロンの実装コードの一部です：

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def error_function(f, y):
    return -y * np.log(f) - (1 - y) * np.log(1 - f)

# パラメータの初期値（ランダム）
w = np.random.normal(0, 0.3, d+1)

# 確率的勾配降下法によるパラメータ推定
error = []
error_test = []

num_epoch = 10
eta = 0.01

for epoch in range(0, num_epoch):
    index = np.random.permutation(n)
    e_train = np.full(n,np.nan)

    for i in index:
        xi = np.append(1, x_train[i, :])
        yi = y_train[i]
        fi = sigmoid(np.dot(w, xi))
        e_train[i] = error_function(fi, yi)

        if epoch == 0:
            continue

        eta_t = eta / (epoch + 1)
        gradient = (fi - yi) * xi
        w = w - eta_t * gradient
```

生成AIによる設計原則の評価

生成AIに上記のコードを評価してもらったところ、以下の点が指摘されました：

1. **単一責任の原則（SRP）違反**：
    - コードが学習アルゴリズム、活性化関数、誤差計算など複数の責任を持っている
    - 各機能を別々のクラスやモジュールに分離すべき
2. **DRY原則（Don't Repeat Yourself）の遵守**：
    - 訓練データとテストデータの処理で同じロジックが繰り返されている可能性がある
    - 共通処理を関数化すべき
3. **グローバル変数の過剰使用**：
    - `w`, `error`, `error_test`などのグローバル変数に依存している
    - これらをクラスの属性として管理すべき

自分の意見と改善案

生成AIの指摘は最もだなと思うのと同時に、小さい規模のコードでは機能を分けすぎるのも見にくくなるのではないかと思った。 以下に改善案を示す。

```
class Perceptron:
    def __init__(self, input_dim, learning_rate=0.01, epochs=10):
        self.w = np.random.normal(0, 0.3, input_dim+1)
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.train_errors = []
        self.test_errors = []

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def error_function(self, f, y):
        return -y * np.log(f) - (1 - y) * np.log(1 - f)

    def predict(self, x):
        # 入力xに対する予測
        x_with_bias = np.append(1, x)
        return self.sigmoid(np.dot(self.w, x_with_bias))

    def compute_error(self, X, y):
        # データセット全体の誤差を計算
        n = len(y)
        errors = np.zeros(n)
        for i in range(n):
            xi = np.append(1, X[i])
            fi = self.sigmoid(np.dot(self.w, xi))
            errors[i] = self.error_function(fi, y[i])
        return np.mean(errors)

    def fit(self, X_train, y_train, X_test=None, y_test=None):
```

```
        n, d = X_train.shape

        for epoch in range(self.epochs):
            # データをシャッフル
            indices = np.random.permutation(n)

            for i in indices:
                xi = np.append(1, X_train[i])
                yi = y_train[i]

                # 予測と誤差計算
                fi = self.sigmoid(np.dot(self.w, xi))

                # パラメータ更新（epoch=0でも更新するように修正）
                eta_t = self.learning_rate / (epoch + 1)
                gradient = (fi - yi) * xi
                self.w = self.w - eta_t * gradient

            # エポックごとの誤差を記録
            self.train_errors.append(self.compute_error(X_train, y_train))
            if X_test is not None and y_test is not None:
                self.test_errors.append(self.compute_error(X_test,
y_test))
```

この改善案では：

1. クラスを導入して責任を明確に分離
2. 予測と誤差計算を別メソッドに分離
3. グローバル変数をクラス属性に変更
4. 訓練とテストの共通処理を関数化

課題7-8-2：コードレビュー

レビュー対象のコード

以下は、以前に作成した数列計算のコードです：

```
import numpy as np

# 行列Aとベクトルbの作成
A = np.arange(1, 21, dtype=float).reshape(4, 5)
print("行列A:")
print(A)

# ベクトルbの作成
b = np.array([1, 0, 1, 0, 1], dtype=float)
print("\nベクトルb:")
print(b)

# 行列Aとベクトルbの積Abを計算
Ab = A @ b
```

```
print("\n行列Aとベクトルbの積Ab:")
print(Ab)

# 行列Aの列和と行和を計算
column_sum = A.sum(axis=0)
print("\n行列Aの列和:")
print(column_sum)

row_sum = A.sum(axis=1)
print("\n行列Aの行和:")
print(row_sum)

# 数列の計算
print("\n数列 a0 = 6, an+1 = an/2 (an が偶数) または 3an+1 (an が奇数) の最初の
10項:")

# 初期値 a0 = 6
a = 6
print(f"a0 = {a}")

# a1からa10までを計算
for n in range(1, 11):
    # 偶数の場合（2で割り切れる場合）
    if a % 2 == 0:
        a = a / 2
    # 奇数の場合
    else:
        a = 3 * a + 1

    print(f"a{n} = {a}")
```

自分で行ったレビュー内容

1. **コメントの不足**：
    - 関数や複雑なロジックに対する説明コメントが少ない
    - 特に数列計算の部分で、アルゴリズムの説明がない
2. **変数名の改善**：
    - 変数`a`は意味が不明確。`sequence_value`などより説明的な名前にすべき
    - `Ab`も`matrix_vector_product`などより説明的な名前が望ましい
3. **コードの構造化**：
    - 行列計算と数列計算が混在している
    - 機能ごとに関数に分割すべき
4. **エラーハンドリングの欠如**：
    - 入力値の検証や例外処理がない

生成AIによるレビュー内容

生成AIは以下の点を指摘しました：

1. **モジュール化の不足**：
    - 行列計算と数列計算を別々の関数に分割すべき
    - メイン処理を`if __name__ == "__main__":`ブロック内に配置すべき
2. **ドキュメンテーションの不足**：
    - 関数のドキストリングがない
    - コードの目的や使用方法の説明がない
3. **テストの欠如**：
    - コードの正確性を検証するテストがない
4. **定数の使用**：
    - マジックナンバー（6, 10など）を定数として定義すべき
5. **型ヒントの欠如**：
    - Python 3.5以降では型ヒントを使用して可読性を向上させるべき

自分の意見と考察

生成AIの指摘は的確だと思います。特にモジュール化とドキュメンテーションは、コードの保守性と再利用性を高めるために重要です。

私が見落としていた点として、「メイン処理を`if __name__ == "__main__":`ブロック内に配置する」という指摘は重要です。これにより、スクリプトをモジュールとしてインポートした場合に自動実行されるのを防ぐことができるためです。

また、型ヒントの使用はコードの理解しやすさを向上させる重要な要素だと考えます。

改善案として、以下のようなコードを提案します：

```
import numpy as np
from typing import List, Tuple

def create_matrix_and_vector() -> Tuple[np.ndarray, np.ndarray]:
    """行列Aとベクトルbを作成する関数"""
    A = np.arange(1, 21, dtype=float).reshape(4, 5)
    b = np.array([1, 0, 1, 0, 1], dtype=float)
    return A, b

def calculate_matrix_operations(A: np.ndarray, b: np.ndarray) ->
Tuple[np.ndarray, np.ndarray, np.ndarray]:
    """行列演算を行う関数"""
    Ab = A @ b
    column_sum = A.sum(axis=0)
    row_sum = A.sum(axis=1)
    return Ab, column_sum, row_sum
```

```python
def calculate_sequence(initial_value: float, n_terms: int) -> List[float]:
    """
    数列 a0 = initial_value, an+1 = an/2 (an が偶数) または 3an+1 (an が奇数)
の
    最初のn_terms項を計算する関数
    """
    sequence = [initial_value]
    current_value = initial_value

    for n in range(1, n_terms + 1):
        if current_value % 2 == 0:
            current_value = current_value / 2
        else:
            current_value = 3 * current_value + 1
        sequence.append(current_value)

    return sequence

if __name__ == "__main__":
    # 行列計算
    A, b = create_matrix_and_vector()
    Ab, column_sum, row_sum = calculate_matrix_operations(A, b)

    print("行列A:")
    print(A)
    print("\nベクトルb:")
    print(b)
    print("\n行列Aとベクトルbの積Ab:")
    print(Ab)
    print("\n行列Aの列和:")
    print(column_sum)
    print("\n行列Aの行和:")
    print(row_sum)

    # 数列計算
    INITIAL_VALUE = 6
    N_TERMS = 10

    sequence = calculate_sequence(INITIAL_VALUE, N_TERMS)

    print(f"\n数列 a0 = {INITIAL_VALUE}, an+1 = an/2 (an が偶数) または 3an+1
(an が奇数) の最初の{N_TERMS}項:")
    for i, value in enumerate(sequence):
        print(f"a{i} = {value}")
```