Sieci neuronowe w zastosowaniach biomedycznych - Implementacja sieci neuronowej SOM

Milena Kuna - 325033
Karol Franczuk - 325001
Zespół 16, temat 12

4 maja 2025

Temat projektu

Klasyfikacja kwiatów za pomocą sieci uczonej bez nauczyciela (SOM) – katalog: *iris*.

1. Analiza danych

1.1 Opis zbioru danych

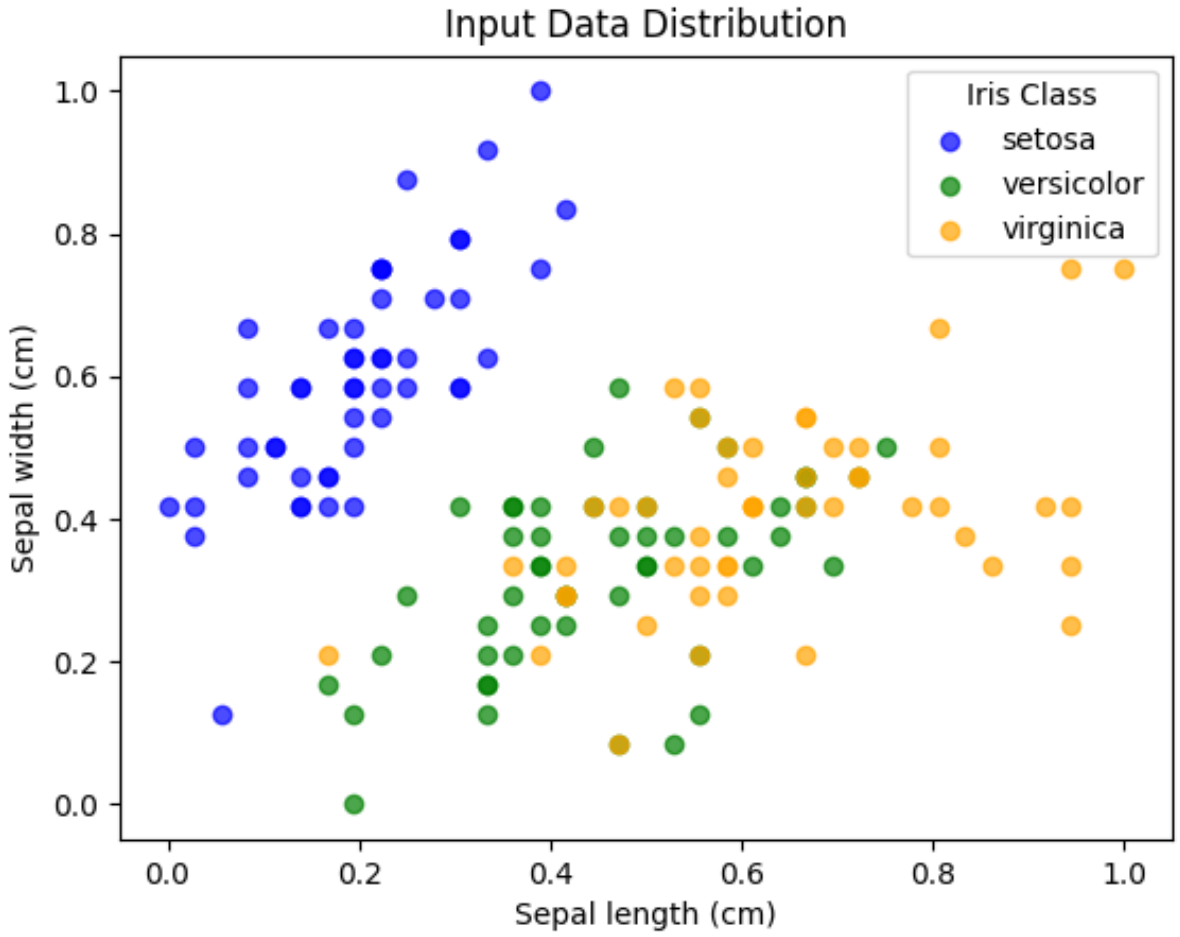
Zbiór danych *Iris (Fisher's Iris Dataset)* zawiera pomiary 150 egzemplarzy trzech gatunków irysów: *Iris-setosa*, *Iris-versicolor*, *Iris-virginica*. Każda próbka opisana jest za pomocą czterech cech numerycznych:

Nr	Nazwa cechy	Jednostka	Zakres wartości
1	Długość działki kielicha	cm	4.3 – 7.9
2	Szerokość działki kielicha	cm	2.0 – 4.4
3	Długość płatka	cm	1.0 – 6.9
4	Szerokość płatka	cm	0.1 – 2.5

Dodatkową cechą jest etykieta klasy (gatunek kwiatu), wykorzystywana jedynie w celu oceny działania klasyfikatora.

1.2 Statystyki opisowe cech

Cecha	Średnia	Odchylenie standardowe
Długość działki kielicha	5.84	0.83
Szerokość działki kielicha	3.05	0.43
Długość płatka	3.76	1.76
Szerokość płatka	1.20	0.76



Rysunek 1: Rozkład danych wejściowych na dwóch pierwszych głównych składowych PCA. Kolory odpowiadają klasom Iris.

2. Wstępne przetwarzanie danych

- **Skalowanie cech**: cechy wejściowe zostały przeskalowane do zakresu [0, 1] (normalizacja), aby dostosować dane do wymagań zastosowanej implementacji SOM.
- **Kodowanie danych nienumerycznych**: Zbiór zawiera tylko cechy numeryczne, więc kodowanie nie było wymagane.
- **Podział danych**: W przypadku SOM, jako sieci uczącej się bez nadzoru, nie jest wymagany podział na zbiór treningowy i testowy. Etykiety klas zostały użyte jedynie do wizualizacji i oceny jakości klasteryzacji.

3. Koncepcja realizacji sieci neuronowej

3.1 Typ sieci

Zastosowana została sieć typu **Self-Organizing Map (SOM)**, znana również jako mapa Kohonena. Jest to sieć neuronowa ucząca się bez nadzoru, wykorzystywana głównie do grupowania danych i redukcji wymiarowości.

3.2 Architektura i parametry sieci

- **Liczba neuronów wejściowych**: 4 (odpowiadające czterem cechom kwiatów).
- **Struktura warstwy SOM**: siatka 10x10 neuronów (100 neuronów).
- **Liczba epok**: 50.
- **Funkcja sąsiedztwa**: gaussowska.
- **Metryka odległości**: odległość euklidesowa.
- **Algorytm uczenia**: klasyczny algorytm Kohonena z dynamicznym zmniejszaniem promienia sąsiedztwa i współczynnika uczenia.

Funkcja aktywacji: W klasycznej sieci SOM nie stosuje się funkcji aktywacji w tradycyjnym sensie. Zamiast tego, „aktywacja" neuronu to jego wybór jako zwycięzcy (BMU – *Best Matching Unit*) na podstawie najmniejszej odległości euklidesowej między wagami a wektorem wejściowym.

3.3 Algorytm uczenia

Uczenie odbywa się iteracyjnie poprzez dopasowywanie wag neuronów do danych wejściowych. W każdej iteracji:

1. Wybierany jest wektor wejściowy $\mathbf{x}$.
2. Znajdowany jest neuron BMU – neuron o najmniejszej odległości od $\mathbf{x}$.
3. Aktualizowane są wagi BMU i jego sąsiadów według wzoru:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta(t) \cdot h_{ci}(t) \cdot (\mathbf{x}(t) - \mathbf{w}_i(t))$$

gdzie:

- $\eta(t)$ – współczynnik uczenia malejący w czasie,
- $h_{ci}(t)$ – funkcja sąsiedztwa, zwykle gaussowska.

4. Wyniki - implementacja i wstępne testy

Po przeprowadzeniu treningu na zbiorze danych Iris, uzyskano następujące wyniki:

- Raport klasyfikacji przedstawia wyniki modelu w zakresie precyzji, czułości, F1-score oraz supportu dla każdej z klas.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	0.93	1.00	0.96	50
2	1.00	0.92	0.96	50
accuracy			0.97	150
macro avg	0.98	0.97	0.97	150
weighted avg	0.98	0.97	0.97	150

Rysunek 2: Raport klasyfikacji.

Precyzja to miara dokładności klasyfikatora. Dla każdej z klas, wyniki przedstawiają się następująco:

- Klasa 0: 1.00
- Klasa 1: 0.93
- Klasa 2: 1.00

Czułość mierzy zdolność modelu do wykrywania prawdziwych przypadków danej klasy:

- Klasa 0: 1.00
- Klasa 1: 1.00
- Klasa 2: 0.92

F1-score to średnia harmoniczna precyzji i czułości:

- Klasa 0: 1.00
- Klasa 1: 0.96
- Klasa 2: 0.96

Support oznacza liczbę próbek w każdej klasie:

- Klasa 0: 50
- Klasa 1: 50
- Klasa 2: 50

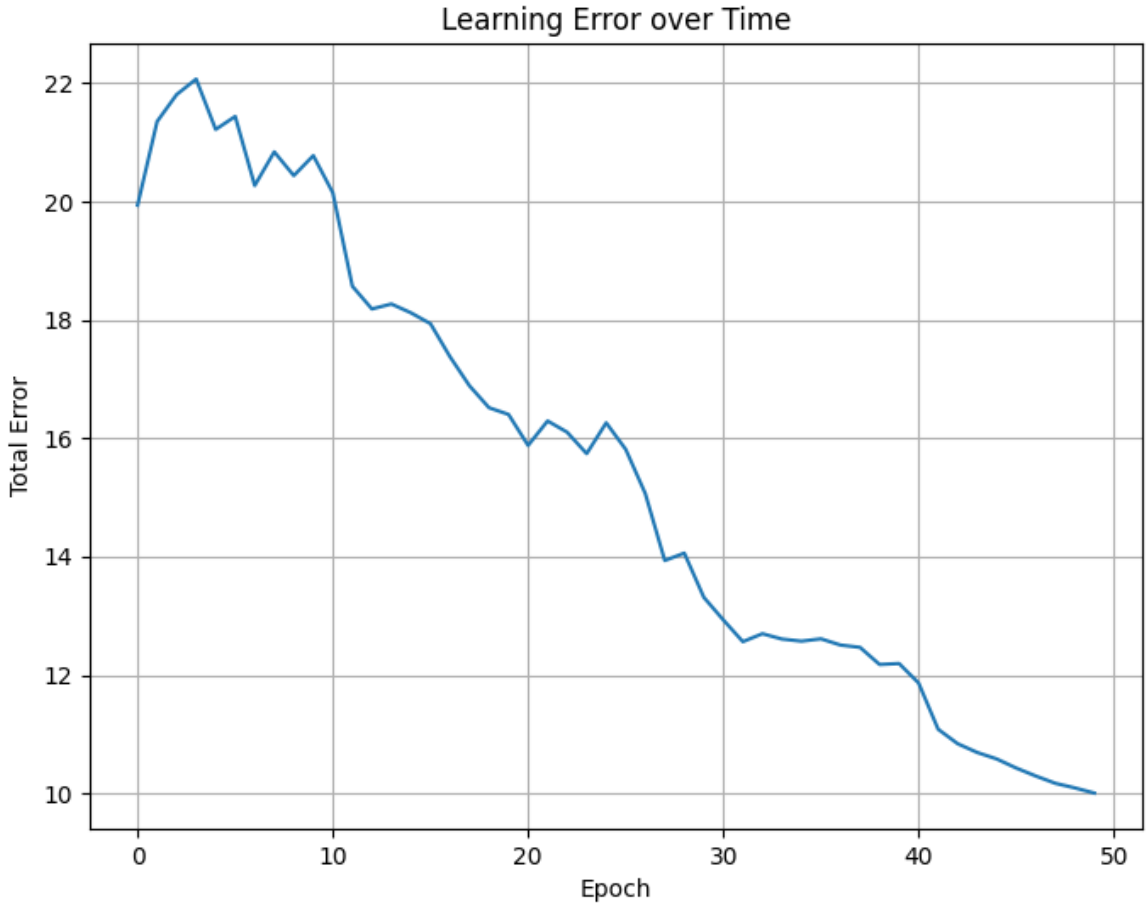
- Dokładność (Accuracy): 0.97
- Średnia makro (Macro average):

    - Precyzja: 0.98
    - Czułość: 0.97
    - F1-score: 0.97
  - Średnia ważona (Weighted average):
    - Precyzja: 0.98
    - Czułość: 0.97
    - F1-score: 0.97

Model osiągnął bardzo dobre wyniki, zwłaszcza w precyzji i czułości dla klasy 0 i 2, gdzie obie metryki wynoszą 1.00. Dla klasy 1, precyzja wynosi 0.93, a czułość 1.00, co również świadczy o wysokiej dokładności klasyfikacji. F1-score dla klas 1 i 2 wynosi 0.96, co wskazuje na dobrą równowagę między precyzją a czułością. Ogólna dokładność modelu wynosi 0.97, co potwierdza jego wysoką efektywność.

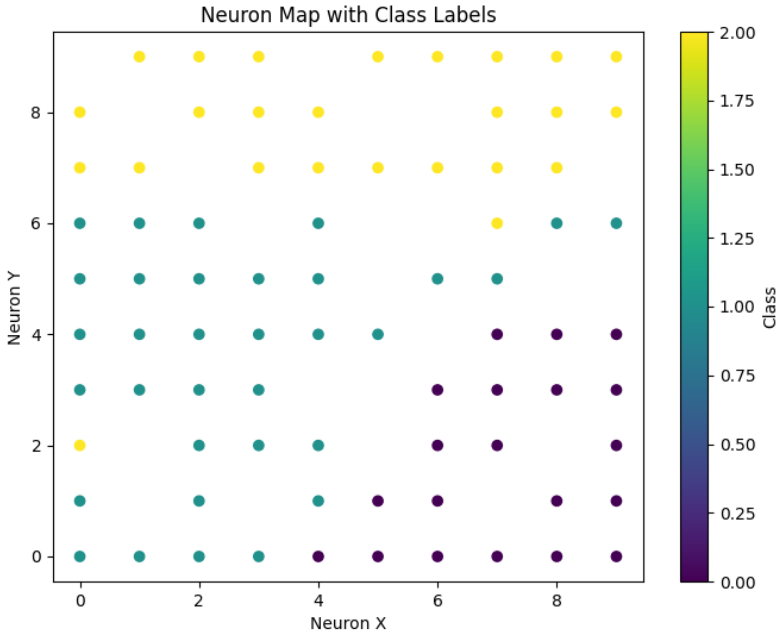
Wykresy:

- **Błąd uczenia** – pokazuje, jak zmienia się błąd w czasie treningu.



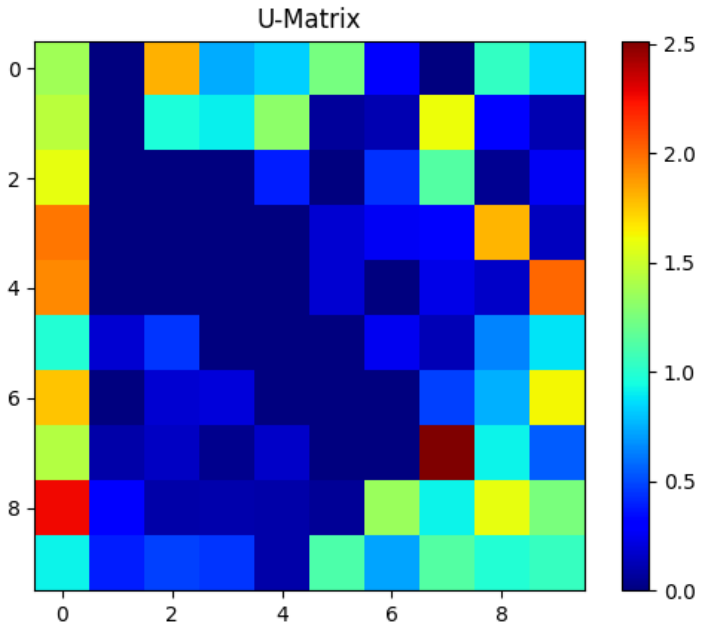
Rysunek 3: Przebieg błędu w zależności od liczby iteracji (epok).

- **Mapa neuronów z klasami** – ilustruje, jak różne klasy zostały przypisane do poszczególnych neuronów.



Rysunek 4: Mapa neuronów z klasami.

- **U-Matrix** – (Unified Distance Matrix) to wizualizacja, która pokazuje odległości między wagami sąsiednich neuronów w sieci samoorganizującej się (SOM). Ciemniejsze obszary U-Matrix wskazują na większe odległości (więcej różnic między danymi), a jaśniejsze obszary oznaczają mniejsze odległości (mniejsze różnice, większe podobieństwo).



Rysunek 5: U-Matrix.

Na podstawie uzyskanych wyników, możemy zauważyć, że sieć SOM skutecznie grupuje próbki danych w odpowiednich klasach. Wskaźniki oceny klasyfikacji, takie jak czułość i specyficzność, zostały

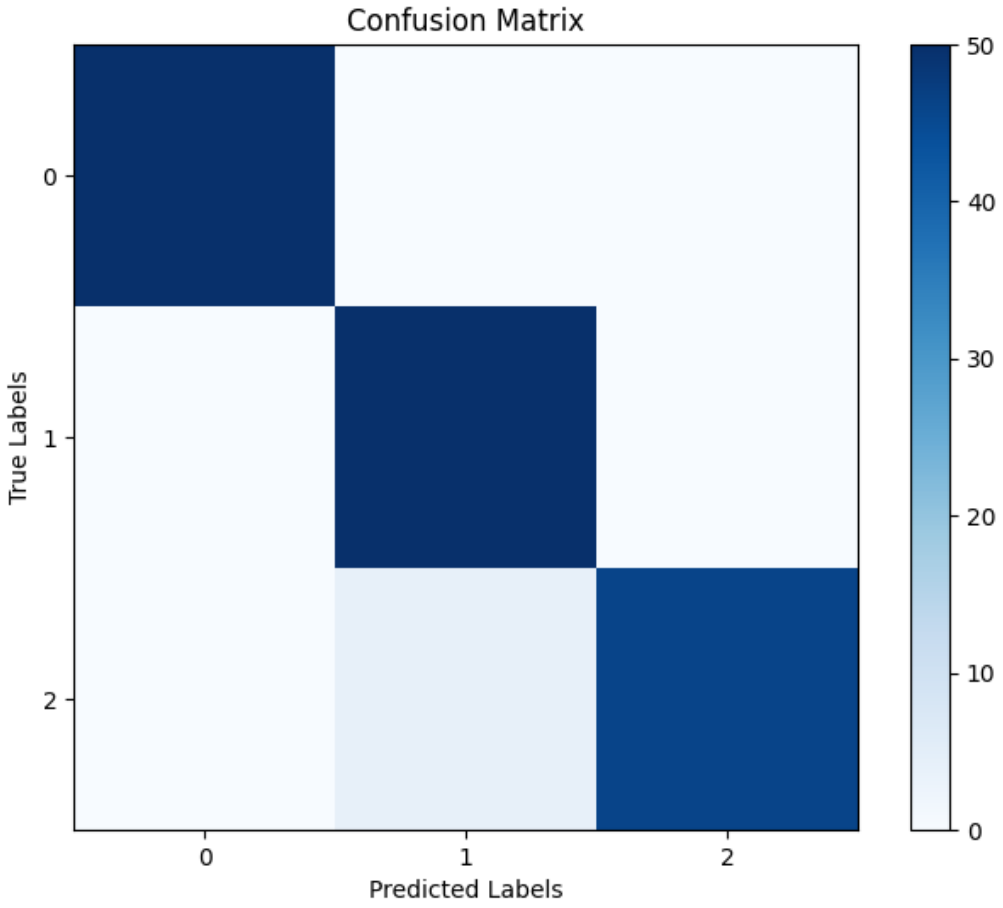
obliczone i są obiecujące, jednak dalsze eksperymenty z różnymi parametrami mogą poprawić dokładność klasyfikacji.

Ewaluacja (czułość/specyficzność)

Na podstawie wyników klasyfikacji modelu SOM uzyskaliśmy następującą macierz pomyłek:

Confusion Matrix:

$$\begin{bmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 7 & 43 \end{bmatrix}$$



Rysunek 6: Macierz pomyłek wygenerowana za pomocą pyplot.

Gdzie:

- 50 to liczba prawidłowo sklasyfikowanych próbek klasy 0 (True Positives - TP),
- 47 to liczba prawidłowo sklasyfikowanych próbek klasy 1 (TP),
- 43 to liczba prawidłowo sklasyfikowanych próbek klasy 2 (TP),
- 0 to liczba błędnych klasyfikacji klasy 0 jako 1 lub 2 (False Negatives - FN),

- 3 to liczba błędnych klasyfikacji klasy 1 jako 2 (FN),
- 7 to liczba błędnych klasyfikacji klasy 2 jako 1 (FN).

Czułość (Sensitivity)

Czułość obliczamy według wzoru:

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

Obliczenia dla każdej klasy:

Dla klasy 0:

$$TP_0 = 50, \quad FN_0 = 0$$
$$\text{Sensitivity}_0 = \frac{50}{50 + 0} = 1.00$$

Dla klasy 1:

$$TP_1 = 47, \quad FN_1 = 3$$
$$\text{Sensitivity}_1 = \frac{47}{47 + 3} = \frac{47}{50} = 0.94$$

Dla klasy 2:

$$TP_2 = 43, \quad FN_2 = 7$$
$$\text{Sensitivity}_2 = \frac{43}{43 + 7} = \frac{43}{50} = 0.86$$

Specyficzność (Specificity)

Specyficzność obliczamy według wzoru:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Gdzie TN to liczba prawidłowo zaklasyfikowanych negatywnych przypadków, a FP to liczba błędnych klasyfikacji próbek, które nie należą do danej klasy, ale zostały zaklasyfikowane jako ta klasa.

Dla klasy 0:

$$TN_0 = 47 + 43 = 90, \quad FP_0 = 0 + 7 = 7$$
$$\text{Specificity}_0 = \frac{90}{90 + 7} = \frac{90}{97} \approx 0.9286$$

Dla klasy 1:

$$TN_1 = 50 + 43 = 93, \quad FP_1 = 0 + 7 = 7$$
$$\text{Specificity}_1 = \frac{93}{93 + 7} = \frac{93}{100} = 0.93$$

Dla klasy 2:

$$TN_2 = 50 + 47 = 97, \quad FP_2 = 0 + 3 = 3$$
$$\text{Specificity}_2 = \frac{97}{97 + 3} = \frac{97}{100} = 0.97$$

Podsumowanie wyników

- Czułość (Sensitivity) dla klasy 0: 1.00 (idealna),
- Czułość (Sensitivity) dla klasy 1: 0.94,
- Czułość (Sensitivity) dla klasy 2: 0.86,
- Specyficzność (Specificity) dla klasy 0: 0.9286,
- Specyficzność (Specificity) dla klasy 1: 0.93,
- Specyficzność (Specificity) dla klasy 2: 0.97.

5. Kod źródłowy

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from collections import defaultdict
from sklearn.metrics import confusion_matrix, classification_report

# === KLASA SOM ===
class SOM:
    def __init__(self, x, y, input_len, learning_rate=0.5, radius=None,
   num_epochs=100):
        self.x = x
        self.y = y
        self.input_len = input_len
        self.learning_rate = learning_rate
        self.num_epochs = num_epochs
        self.radius = radius if radius else max(x, y) / 2
        self.time_constant = self.num_epochs / np.log(self.radius)
        self.weights = np.random.rand(x, y, input_len)
        self.neuron_locations = np.array(list(self._neuron_locations()))
        self.errors = []

    def _neuron_locations(self):
        for i in range(self.x):
            for j in range(self.y):
                yield np.array([i, j])

    def _find_bmu(self, sample):
        bmu_idx = None
        min_dist = np.inf
        for loc in self.neuron_locations:
            w = self.weights[loc[0], loc[1], :]
            dist = np.linalg.norm(sample - w)
            if dist < min_dist:
                min_dist = dist
                bmu_idx = loc
        return bmu_idx, min_dist

    def _decay_radius(self, epoch):
        return self.radius * np.exp(-epoch / self.time_constant)

    def _decay_learning_rate(self, epoch):
        return self.learning_rate * np.exp(-epoch / self.num_epochs)

    def train(self, data):
        for epoch in range(self.num_epochs):
```

```
            total_error = 0
            for sample in data:
                bmu, error = self._find_bmu(sample)
                total_error += error
                radius = self._decay_radius(epoch)
                lr = self._decay_learning_rate(epoch)
                for loc in self.neuron_locations:
                    dist_to_bmu = np.linalg.norm(loc - bmu)
                    if dist_to_bmu <= radius:
                        influence = np.exp(-(dist_to_bmu ** 2) / (2 * (radius ** 2)))
                        delta = lr * influence * (sample - self.weights[loc[0], loc[1], :])
                        self.weights[loc[0], loc[1], :] += delta
            self.errors.append(total_error)

    def map_input(self, data):
        return [self._find_bmu(x)[0] for x in data]

# ===   ADOWANIE   DANYCH ===
iris = load_iris()
data = iris.data
labels = iris.target
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data)

# === INICJALIZACJA I TRENING ===
som = SOM(x=10, y=10, input_len=4, learning_rate=0.5, num_epochs=50)
som.train(data_scaled)
mapped = som.map_input(data_scaled)

# === MAPOWANIE KLAS DO NEURON W ===
neuron_label_map = defaultdict(list)
for pos, label in zip(mapped, labels):
    neuron_label_map[tuple(pos)].append(label)

# === ZAMIENIAMY NA DOMINUJ CE KLASY DLA KA DEGO NEURONU ===
neuron_class_map = {}
for neuron, classes in neuron_label_map.items():
    dominant_class = np.bincount(classes).argmax()  # Dominuj ca klasa
    neuron_class_map[neuron] = dominant_class

# === GENERUJEMY PREDYKCJE NA PODSTAWIE MAPOWANIA NEURON W ===
predictions = [neuron_class_map[tuple(bmu)] for bmu in mapped]

# === PRINTUJ NEURONY Z DOMINUJ CYMI KLASAMI ===
for neuron, class_label in list(neuron_class_map.items())[:10]:
    print(f"Neuron {neuron}: Class {class_label}")

# === PREDYKCJA I METRYKI ===
conf_matrix = confusion_matrix(labels, predictions)
print("Confusion Matrix:")
print(conf_matrix)

# Oblicz metryki klasyfikacyjne
print("Classification Report:")
print(classification_report(labels, predictions))

# === WIZUALIZACJE ===

# 1. B    d treningowy
plt.figure(figsize=(8, 6))
plt.plot(som.errors)
```

```
plt.xlabel("Epoch")
plt.ylabel("Total Error")
plt.title("Learning Error over Time")
plt.grid(True)
plt.savefig("training_error.png")
plt.close()

# 2. Rozk ad neuron w z przypisanymi klasami
plt.figure(figsize=(8, 6))
plt.scatter(
    [neuron[0] for neuron in neuron_class_map.keys()],
    [neuron[1] for neuron in neuron_class_map.keys()],
    c=list(neuron_class_map.values()), cmap='viridis')
plt.colorbar(label="Class")
plt.xlabel("Sepal length (cm)")
plt.ylabel("Sepal width (cm)")

plt.title("Neuron Map with Class Labels")
plt.xlabel("Neuron X")
plt.ylabel("Neuron Y")
plt.savefig("neuron_map.png")
plt.close()

# 3. Rozk ad danych wej ciowych na mapie

# Tworzymy kolorow  map  r cznie
colors = ['blue', 'green', 'orange']
names = ['setosa', 'versicolor', 'virginica']

for i in range(3):
    plt.scatter(data_scaled[labels == i, 0], data_scaled[labels == i, 1],
                label=names[i], c=colors[i], alpha=0.7)

plt.legend(title="Iris Class")
plt.title("Input Data Distribution")
plt.xlabel("Sepal length (cm)")
plt.ylabel("Sepal width (cm)")
plt.savefig("input_data_labeled.png")
plt.close()


# 4. Macierz pomy ek
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap='Blues')
plt.title("Confusion Matrix")
plt.colorbar()
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.xticks(np.arange(3), [0, 1, 2])
plt.yticks(np.arange(3), [0, 1, 2])
plt.savefig("confusion_matrix.png")
plt.close()

# 5. Raport klasyfikacji - zapis do pliku
with open("classification_report.txt", "w") as f:
    f.write(classification_report(labels, predictions))
    print("Classification report saved as classification_report.txt")
```

Listing 1: Kod źródłowy opatrzony komentarzami.