Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №6

по дисциплине «Прикладные интеллектуальные системы и экспертные системы»

Нейронные сети. Обучение без учителя

Студент Мастылина А.А.

Группа М-ИАП-22

Руководитель Кургасов В.В.

Липецк 2022 г.

Задание кафедры

Применить нейронную сеть Кохонена с самообучение для задачи кластеризации. На первом этапе сгенерировать случайные точки на плоскости вокруг 2 центров кластеризации (примерно по 20-30 точек). Далее считать, что сеть имеет два входа (координаты точек) и два выхода – один из них равен 1, другой 0 (по тому, к какому кластеру принадлежит точка). Подавая последовательно на вход (вразнобой) точки, настроить сеть путем применения описанной процедуры обучения так, чтобы она приобрела способность определять, к какому кластеру принадлежит точка

Ход работы

Необходимые библиотеки

```
from sklearn.datasets import make_classification
```

```
import matplotlib.pyplot as plt
```

Рисунок 1 – Импортируем необходимые библиотеки

1) Сгенерируем выборку с помощью функции make_blobs. Данная операция представлена на рисунке 2.

```
import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1])
```

```
<matplotlib.collections.PathCollection at 0x1955a85be80>
```

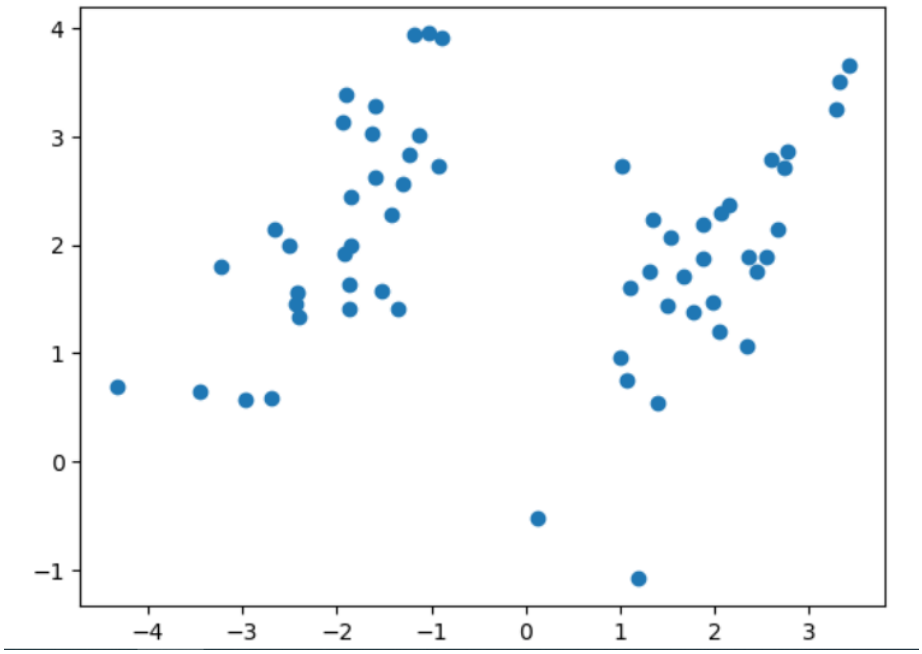

Рисунок 2 – Сгенерированная выборка

2) Выделим два кластера и обозначим их центры, полученный график представлен на рисунке 3.

```
<matplotlib.collections.PathCollection at 0x1955a919ee0>
```

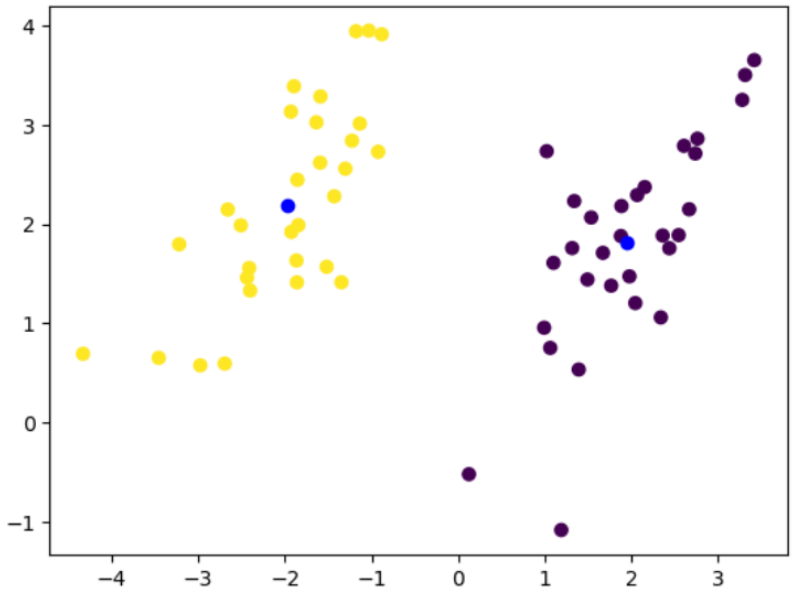

Рисунок 3 – Выделение кластеров

3) Для работы нейросети Кохонена необходимо сгегерировать веса, которые представлены на рисунке 4.

```
np.random.shuffle(X)
T = X
m, n = len(T), len(T[0])

C = 2

weights = np.random.normal(100, 10, size=(n, C)) / 100
weights
```

```
array([[0.89779211, 0.82398696],
       [0.98279444, 0.78587285]])
```

Рисунок 4 – Веса нейросети

4) Последовательное обновление весов представлено на рисунке 4;

```
Шаг для 1 кластера = 0.5
Веса после обновления:
[[ 0.89779211  0.82398696]
 [-0.02363193  2.36745942]]

Шаг для 1 кластера = 0.49
Веса после обновления:
[[0.89779211 0.82398696]
 [1.18146645 2.0681199 ]]

Шаг для 0 кластера = 0.5
Веса после обновления:
[[0.95964319 1.77910692]
 [1.18146645 2.0681199 ]]

Шаг для 0 кластера = 0.49
Веса после обновления:
[[1.45766739 1.62940097]
 [1.18146645 2.0681199 ]]
```

Рисунок 4 – Обновление весов

5) Итоговые веса представлены на рисунке 5:

```
[[-0.72747648  1.65354141]
 [-0.24903032  2.29820975]]
```

Рисунок 5 – Итоговые веса

6) Итоговое качество кластеризации представлено на рисунке 6

```
y == predicted
```

```
array([False, False,  True,  True,  True, False, False, False,  True,
        True, False, False, False, False, False,  True,  True, False,
        True,  True, False, False,  True,  True, False, False,  True,
       False, False,  True,  True, False, False,  True,  True, False,
        True, False,  True, False,  True, False,  True,  True,  True,
        True, False, False,  True, False,  True,  True, False,  True,
        True,  True,  True,  True,  True, False])
```

```
from sklearn.metrics import accuracy_score

print(f'Точность кластеризации: {accuracy_score(y, predicted) * 100}%')
```

```
Точность кластеризации: 53.333333333333336%
```

Рисунок 6 – Точность классификации

Вывод

В ходе выполнения данной лабораторной работы мною были получены навыки построения нейронной сети Кохонена с самообучения для решения задачи кластеризации. После успешного построения и обучения модели была рассчитана характеристика точности классификации точек к их кластерам.

Код программы

```
#!/usr/bin/env python
# coding: utf-8
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=60,
                           n_features=2,
                           n_redundant=0,
                           n_informative=2,
                           n_clusters_per_class=1,
                           n_classes=2,
                           random_state=9,
                           class_sep=2)
import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1])
import numpy as np

def update_cluster_centers(X, c):
    centers = np.zeros((2, 2))
    for i in range(1, 3):
        ix = np.where(c == i)
        centers[i - 1, :] = np.mean(X[ix, :], axis=1)
    return centers
from scipy.cluster.hierarchy import fcluster, linkage

mergings = linkage(X, method='ward')
T = fcluster(mergings, 2, criterion='maxclust')
clusters = update_cluster_centers(X, T)
clusters
```

```
plt.scatter(X[:, 0], X[:, 1], c=T)
plt.scatter(clusters[:, 0], clusters[:, 1], c='blue')
import math
class SOM:
    def __init__(self, n, c):
        """
        n - количестов атрибутов
        C - количество кластеров
        """
        self.n = n
        self.c = c
        self.a = [0 for _ in range(n)]

    def calculate_a(self, i):
        """
        Вычисление значение шага относительного текущего выбора
        """
        return (50 - i) / 100

    def winner(self, weights, sample):
        """
        Вычисляем выигравший нейрон (вектор) по Евклидову расстоянию
        """
        d0 = 0
        d1 = 0
        for i in range(len(sample)):
            d0 += math.pow((sample[i] - weights[0][i]), 2)
            d1 += math.pow((sample[i] - weights[1][i]), 2)

        if d0 > d1:
```

```
            return 0
        else:
            return 1

    def update(self, weights, sample, j):
        """
        Обновляем значение для выигравшего нейрона
        """
        for i in range(len(weights)):
            weights[j][i] = weights[j][i] + self.calculate_a(self.a[j]) * (sample[i] -
weights[j][i])

        print(f'\nШаг для {j} кластера = {self.calculate_a(self.a[j])}')
        self.a[j] += 1
        print(f'Веса после обновления:')
        print(weights)

        return weights

np.random.shuffle(X)
T = X
m, n = len(T), len(T[0])

C = 2

weights = np.random.normal(100, 10, size=(n, C)) / 100
weights
som = SOM(n, C)
som
```

```
    for i in range(m):
        sample = T[i]
        J = som.winner(weights, sample)
        weights = som.update(weights, sample, J)
    s = X[0]
    J = som.winner(weights, s)

    print(f"Элемент принадлежит к {J} кластеру, на самом деле к {y[0]} кластеру")
    print("Веса: ")
    print(weights)
    predicted = np.array([som.winner(weights, s) for s in X])
    predicted
    y == predicted
    from sklearn.metrics import accuracy_score

    print(f'Точность кластеризации: {accuracy_score(y, predicted) * 100}%')
```