

**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

**ЛАБОРАТОРНАЯ РАБОТА №5**

по дисциплине «Прикладные интеллектуальные системы и экспертные  
системы»

Нейронные сети. Обучение без учителя

Студент

Сухоруких А.О.

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. Коэффициент выбрать, уменьшая его от шага к шагу по правилу  $\eta = 50 - i/100$ , причем для каждого нейрона это будет свое значение, а подстраиваться на каждом шаге будут веса только одного (выигравшего) нейрона.

Ход работы

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

```
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)
```

✓ 0.1s

Рисунок 1 – Сгенерированная выборка

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

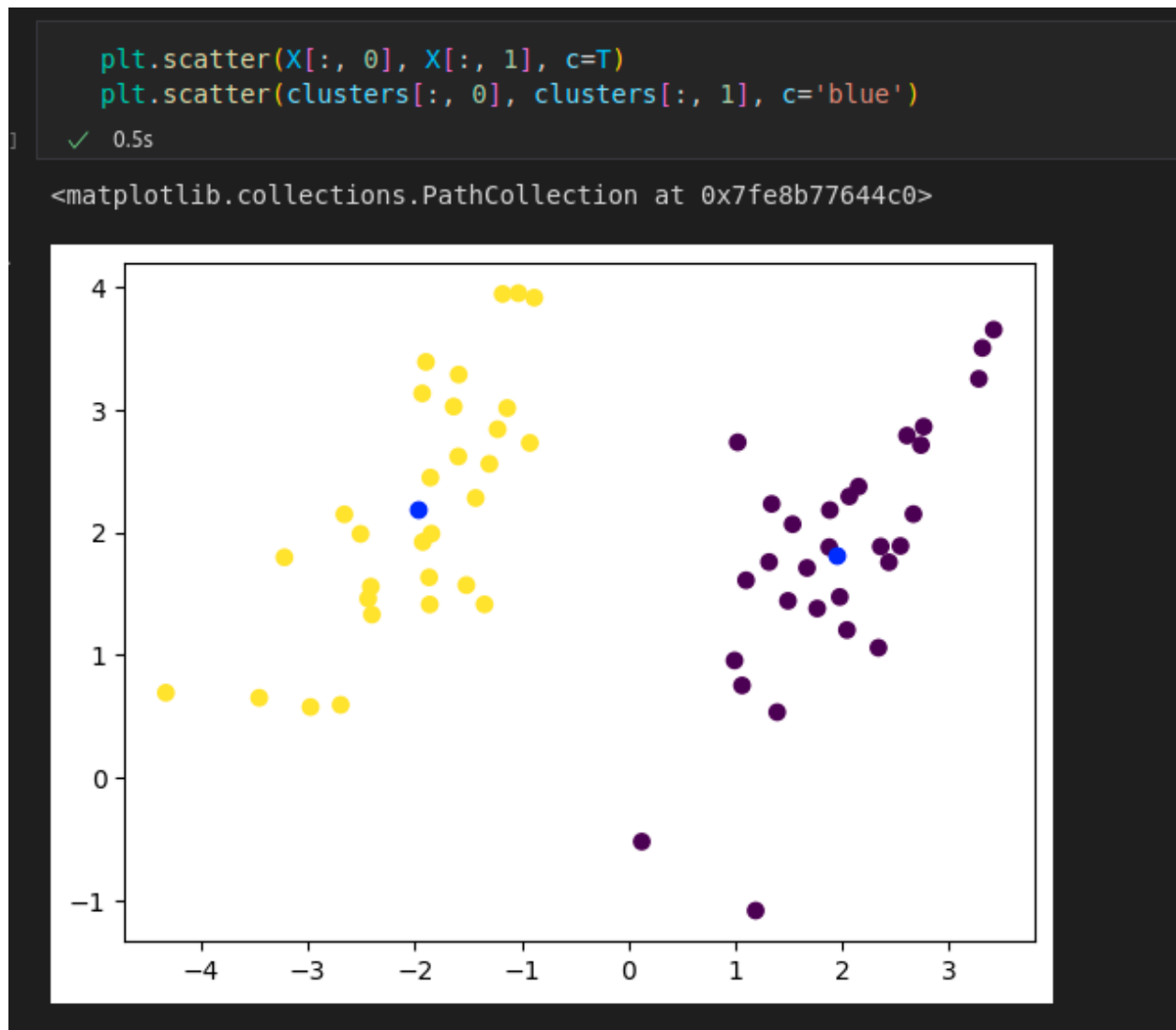


Рисунок 2 – Выделение кластеров

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

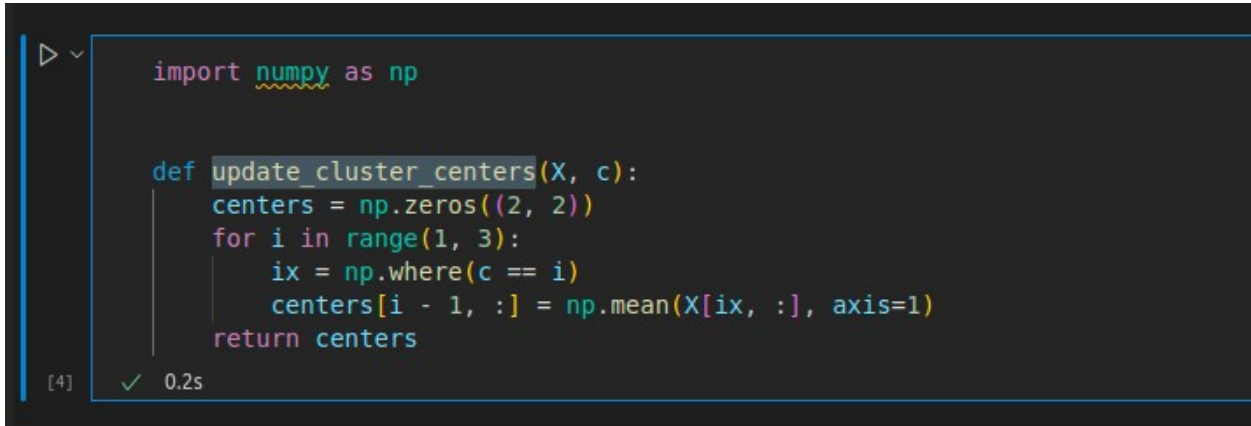
```
from scipy.cluster.hierarchy import fcluster, linkage

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

✓ 0.2s

```
array([[ 1.94772593,  1.8161229 ],
       [-1.96705206,  2.19521329]])
```

4) Последовательное обновление весов представлено на рисунке 4;



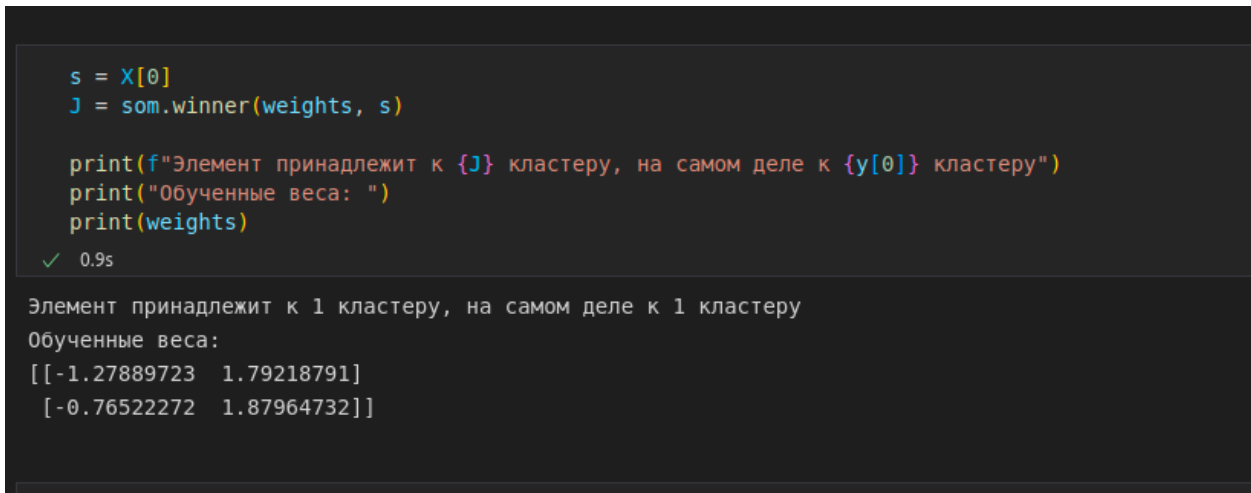
```
import numpy as np

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
```

[4] ✓ 0.2s

Рисунок 4 – Обновление весов

5) Итоговые веса представлены на рисунке 5:



```
s = X[0]
J = som.winner(weights, s)

print(f"Элемент принадлежит к {J} кластеру, на самом деле к {y[0]} кластеру")
print("Обученные веса: ")
print(weights)
```

✓ 0.9s

Элемент принадлежит к 1 кластеру, на самом деле к 1 кластеру

Обученные веса:

```
[[-1.27889723  1.79218791]
 [-0.76522272  1.87964732]]
```

Рисунок 5 – Итоговые веса

6) Итоговое качество кластеризации представлено на рисунке 6

```
y == predicted
[13] ✓ 0.8s

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

from sklearn.metrics import accuracy_score

print(f'Точность кластеризации: {accuracy_score(y, predicted) * 100}%')
[14] ✓ 0.1s

... Точность кластеризации: 51.66666666666667%
```

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

```
# %%  
  
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 - количество 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 относительно 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

# %%
# Обучающая выборка (m, n)
# m - объем выборки
# n - количество атрибутов в записи
np.random.shuffle(X)
T = X
m, n = len(T), len(T[0])

```

```

# Обучающие веса (n, C)
# n - количество атрибутов в записи
# C - количество кластеров
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("Обученные 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}%')

# %%

```