

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

Крутских А.Ю.

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

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

Генерация выборки

```
Ввод [2]: X, y = make_blobs(n_samples=50, centers=2, random_state=41, cluster_std=0.1)
```

Сгенерированные точки ¶

```
Ввод [3]: plt.scatter(X[:, 0], X[:, 1])
```

```
Out[3]: <matplotlib.collections.PathCollection at 0x7efbdc348e20>
```

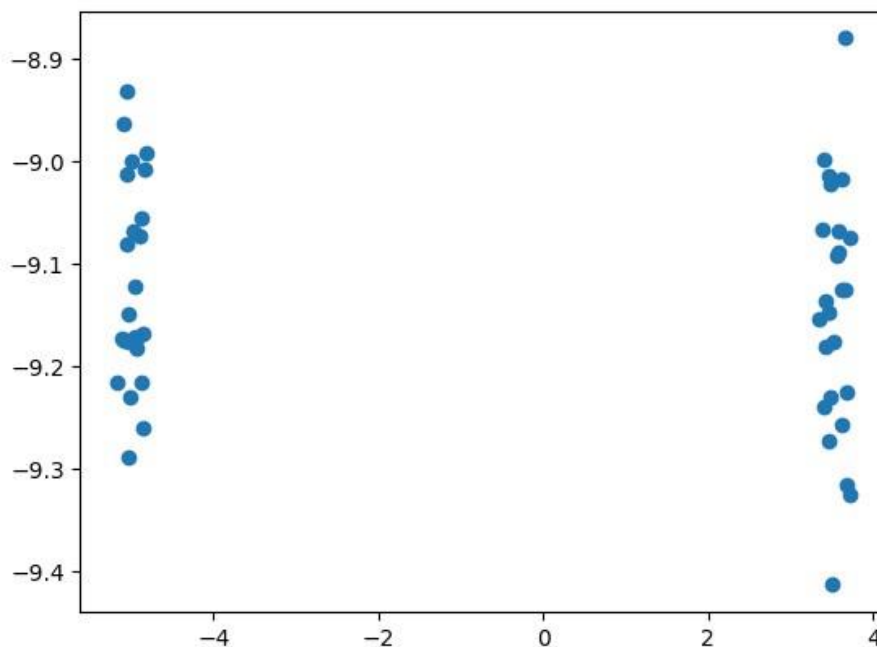


Рисунок 1 – Сгенерированная выборка

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

```
Ввод [6]: plt.scatter(X[:, 0], X[:, 1], c=T)
plt.scatter(clusters[:, 0], clusters[:, 1], c='blue')

Out[6]: <matplotlib.collections.PathCollection at 0x7efbda270640>
```

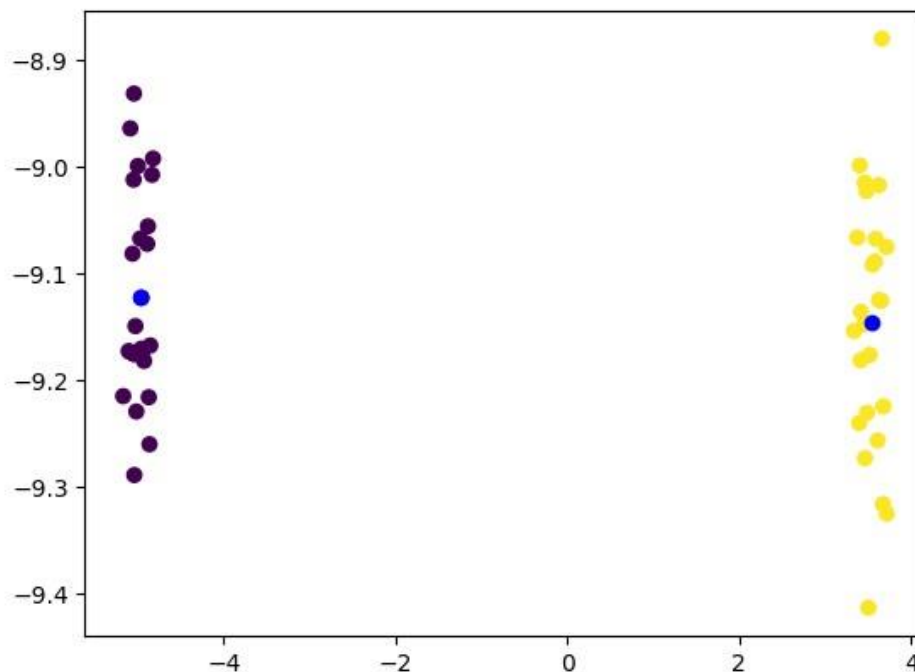


Рисунок 2 – Выделение кластеров

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

```
Ввод [55]: # 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

Out[55]: array([[0.98676425, 1.14036345],
                 [1.06541381, 0.89179194]])
```

Рисунок 3 – Веса нейросети

4) Последовательное обновление весов представлено на рисунке 4;

```
Ввод [50]: for j in range(100):
            for i in range(m):
                sample = T[i]
                J = som.winner(weights, sample)
                weights = som.update(weights, sample, J)
```



```
Шаг для 0 кластера = 0.5
Веса после обновления:
[[ 2.3261831 -4.16405659]
 [ 1.04562194  0.92402661]]

Шаг для 1 кластера = 0.5
Веса после обновления:
[[ 2.3261831 -4.16405659]
 [ 2.22927584 -4.12867032]]

Шаг для 0 кластера = 0.49
Веса после обновления:
[[-1.25303827 -6.58571382]
 [ 2.22927584 -4.12867032]]

Шаг для 1 кластера = 0.49
Веса после обновления:
[[-1.25303827 -6.58571382]
```

Рисунок 4 – Обновление весов 5)

Итоговые веса представлены на рисунке 5:

```
Ввод [56]: 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)
```



```
Элемент принадлежит к 0 кластеру, на самом деле к 1 кластеру
Обученные веса:
[[0.98676425  1.14036345]
 [1.06541381  0.89179194]]
```

Рисунок 5 – Итоговые веса

6) Итоговое качество кластеризации представлено на рисунке 6

```
Ввод [68]: y == predicted
```



```
Out[68]: array([ True,  True, False, False, False,  True, False,  True,  True,
                True, False, False, False, False,  True,  True,  True, False,
                False, True, False, True,  True,  True,  True, False,  True,
                False, True,  True,  True,  True,  True,  True,  True,  True,
                False, False,  True, False, False,  True,  True,  True,  True,
                True,  True,  True,  True, False])
```



```
Ввод [69]: print(f'Точность кластеризации: {accuracy_score(y, predicted) * 100}%')
```



```
Точность кластеризации: 64.0%
```

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
```

```
# In[1]:
```

```
from sklearn.datasets import make_blobs import  
matplotlib.pyplot as plt import numpy as np from  
scipy.cluster.hierarchy import fcluster, linkage  
import math from sklearn.metrics import  
accuracy_score
```

```
# ## Генерация выборки
```

```
# In[57]:
```

```
X, y = make_blobs(n_samples=50, centers=2, random_state=42,  
cluster_std=0.1)
```

```
# ## Сгенерированные точки
```

```
# In[58]:
```

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

```
# In[59]:
```

```
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
```

```
# In[60]:
```

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

```
# In[61]:
```

```
plt.scatter(X[:, 0], X[:, 1], c=T) plt.scatter(clusters[:,  
0], clusters[:, 1], c='blue') # In[62]:
```

```
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 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] + 0.5 * (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

```

In[63]:

```

# 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

```

```
# In[64]:
```

```
som = SOM(n, C) som
```

```
# In[65]:
```

```
for j in range(100):
```

```
    for i in range(m):
```

```
        sample = T[i]
```

```
            J = som.winner(weights, sample)
```

```
            weights = som.update(weights, sample, J)
```

```
# In[66]:
```

```
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)
```

```
# In[67]:
```

```
predicted = np.array([som.winner(weights, s) for s in X]) predicted
```

```
# In[68]:
```

```
y == predicted
```

```
# In[69]:
```

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

```
# In[ ]:
```

```
# In[ ]:
```

```
# In[ ]:
```

