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 Коровайцев А.А.

Группа М-ИАП-23-1

Руководитель Кургасов В.В.

Доцент

Липецк 2023 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 α выбрать, уменьшая его от шага к шагу по правилу $\alpha = (50-i)/100$, причем для каждого нейрона это будет свое значение α, а подстраиваться на каждом шаге будут веса только одного (выигравшего) нейрона.

Ход работы

Для генерации данных воспользуемся функцией make_classification из пакета sklearn.datasets. После генерации данных визуализируем их с использованием библиотеки matplotlib.pyplot. Визуализация полученных данных представлена на рисунке 1.

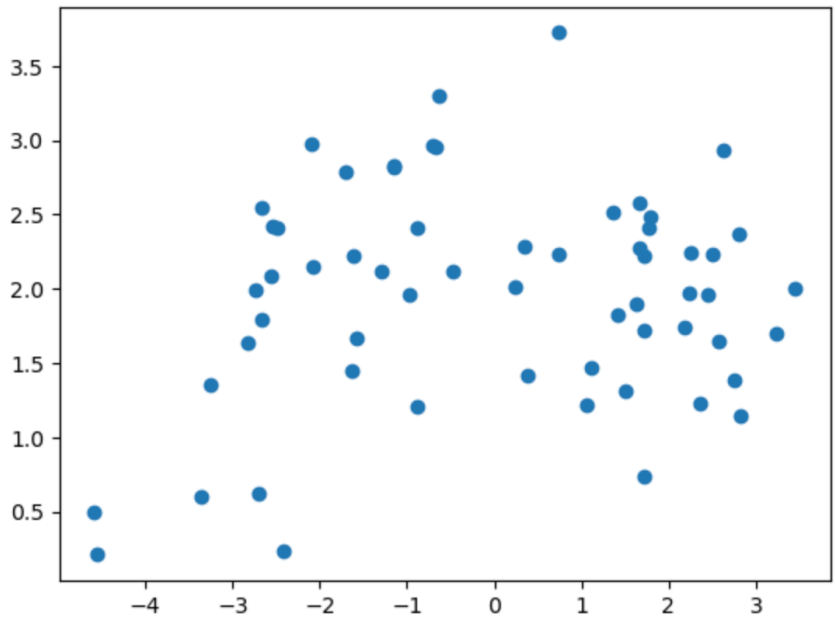

Рисунок 1 – График полученных данных

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

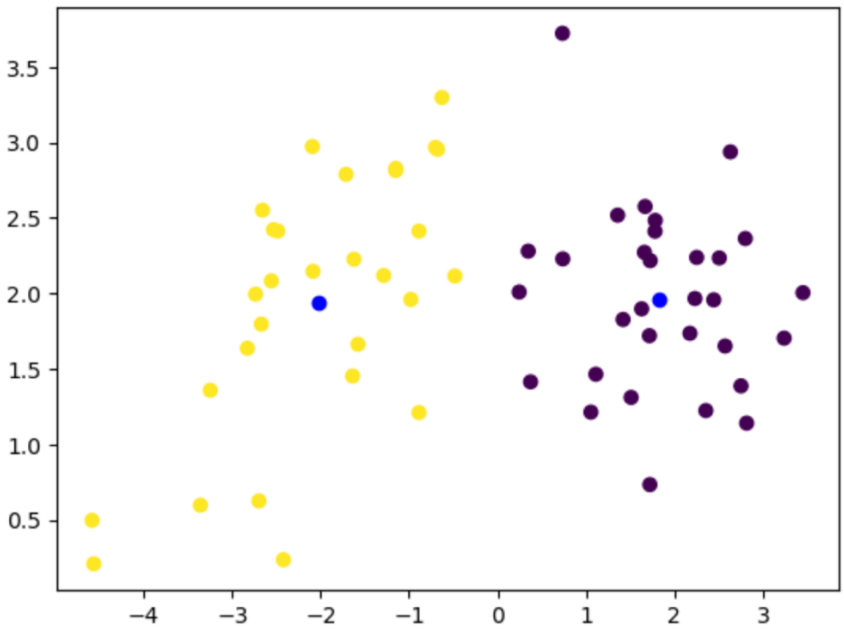

Рисунок 2 – График кластеров данных и их центры

Перемешаем данные и последовательно будем подавать их на вход в нейронную сеть Кохонена. Для её работы необходимы веса, которые представлены на рисунке 3.

```
([[1.09909639, 0.86606284],
  [0.92866499, 0.98073679]])
```

Рисунок 3 – Веса для работы нейронной сети

Далее будем обновлять эти веса после при выборе выигрышного нейрона. Веса после выбора нейрона представлены на рисунке 4. Итоговые веса после обучения представлены на рисунке 5.

```
Шаг для 0 кластера = 0.5
Веса после обновления:
[[0.10539675 1.63957672]
 [0.92866499 0.98073679]]

Шаг для 1 кластера = 0.5
Веса после обновления:
[[ 0.10539675  1.63957672]
 [-0.74608651  0.60786591]]

Шаг для 1 кластера = 0.49
Веса после обновления:
[[0.10539675 1.63957672]
 [0.46070016 1.1527871 ]]

Шаг для 1 кластера = 0.48
Веса после обновления:
[[ 0.10539675  1.63957672]
 [-0.18657318  1.1805079 ]]
```

Рисунок 4 – Веса при каждой итерации нейронной сети

```
[[-0.03004055  1.94265381]
 [ 0.23121714  2.0696989 ]]
```

Рисунок 5 – Итоговые обученные веса

Посмотрим на итоговую точность классификации предсказанных значений принадлежности точек к кластерам. Полученный результат представлен на рисунке 6.

```
Точность кластеризации: 63.333333333333%
```

Рисунок 6 – Итоговая точность кластеризации

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, которая составила 63,3%.

Приложение А

Исходный код

```
#!/usr/bin/env python
# coding: utf-8

# # Лабораторная работа №0
# ## Задание
# ### 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 α выбрать, уменьшая его от шага к шагу по правилу α = (50-i)/100, причем для каждого нейрона это будет свое значение α, а подстраиваться на каждом шаге будут веса только одного (выигравшего) нейрона.

# In[703]:


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=56,
                           class_sep=2)


# In[704]:


import matplotlib.pyplot as plt

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


# In[705]:


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


# In[706]:
```

```
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


# In[707]:


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


# In[708]:


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


# In[709]:


# 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[710]:


som = SOM(n, C)
som


# In[711]:


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


# In[712]:


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[713]:


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


# In[714]:


y == predicted


# In[715]:
```

```
from sklearn.metrics import accuracy_score

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