**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.

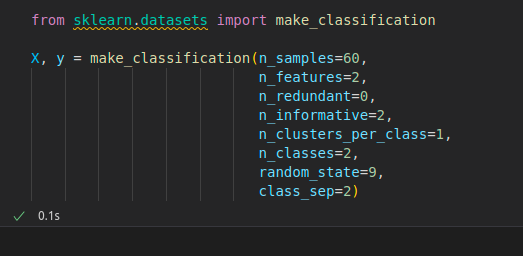
Липецк 2022 г.

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 𝛼 выбрать, уменьшая его от шага к шагу по правилу 𝛼 = 50−𝑖100, причем для каждого нейрона это будет свое значение 𝛼, а подстраиваться на каждом шаге будут веса только одного (выигравшего) нейрона.

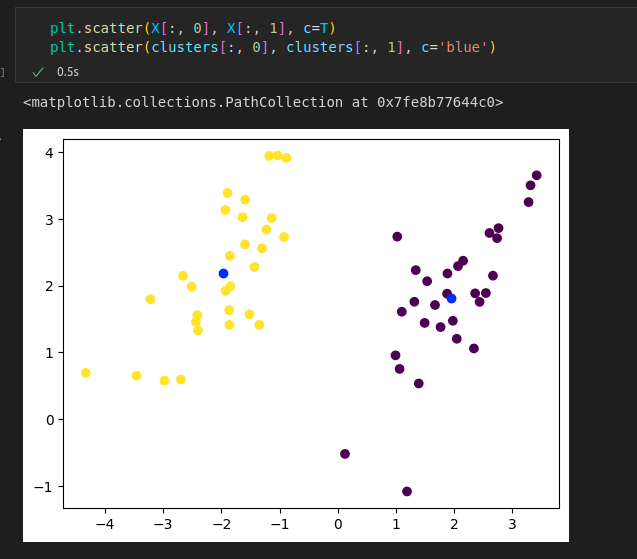
Ход работы

1) Сгенерируем выборку с помощью функции make\_blobs. Данная операция представлена на рисунке 1.

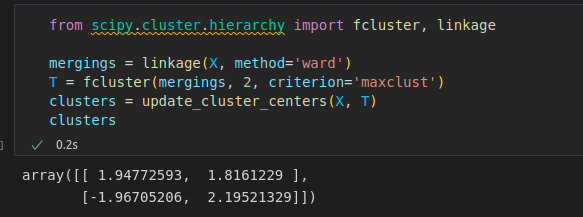
 Рисунок 1 – Сгенерированная выборка

2) Выделим два кластера и обозначим их центры, полученный график

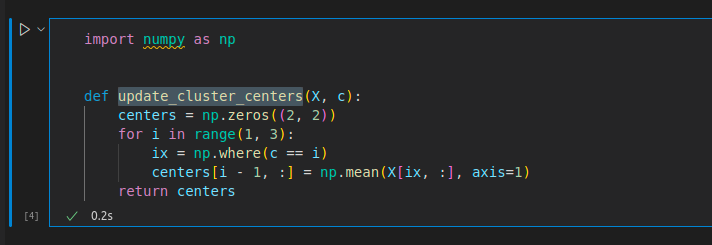
представлен на рисунке 2.

 Рисунок 2 – Выделение кластеров

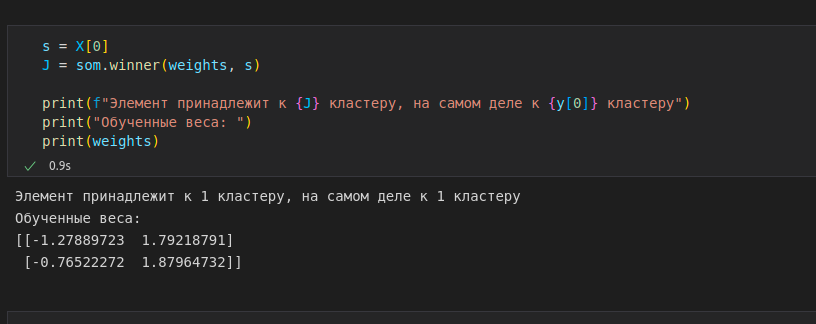
3) Для работы нейросети Кохонена необходимо сгегерировать веса, которые представлены на рисунке 3.

 Рисунок 3 – Веса нейросети

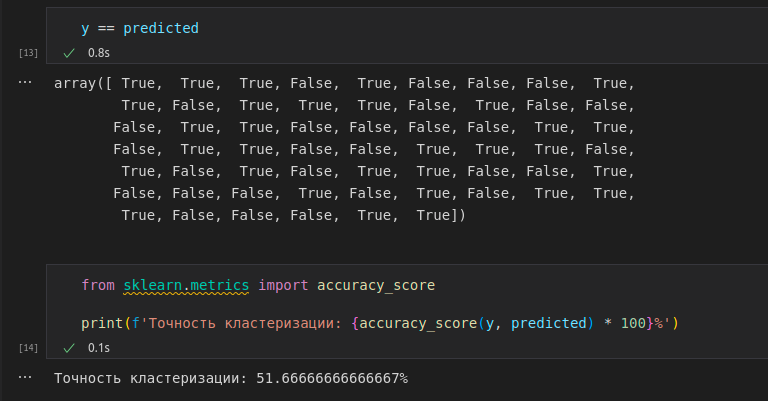
4) Последовательное обновление весов представлено на рисунке 4;

Рисунок 4 – Обновление весов

5) Итоговые веса представлены на рисунке 5:

 Рисунок 5 – Итоговые веса

6) Итоговое качество кластеризации представлено на рисунке 6

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 - 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

# %%

# 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}%')

# %%