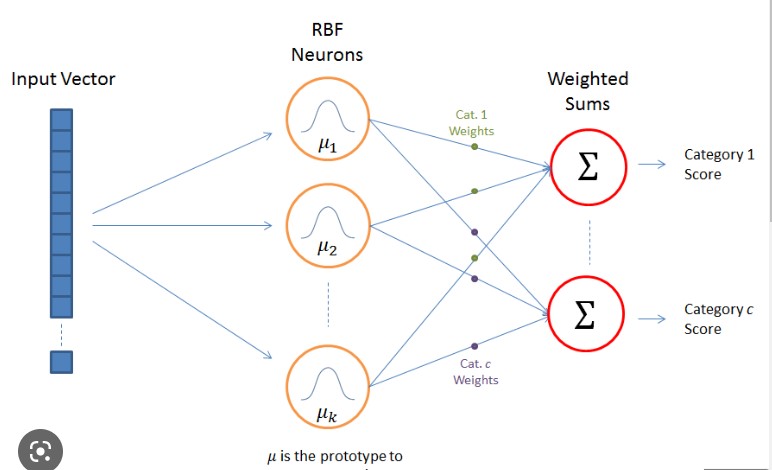
**Rbfتمرین**

پویا شریفی

پارسا پیله ور

ساختار rbf :



اول تابع گوسین را تعریف کردیم:

def rbf(x, C, S):

    #c mu

    #s sigma

    return (1 / np.sqrt(2 \* np.pi \* (S \*\* 2))) \* np.exp((-(x - C) \*\* 2) / (2 \* (S

\*\* 2)))

سپس انحراف معیار را بدست آوردیم:

def euclidean\_distance(x1, x2):

    return np.sqrt(np.sum((x1 - x2) \*\* 2))

سپس تابع kmeans را تعریف کردیم(ما دو تا kmeans هم زدیم ,که یکیش کامت هست و به صورت کلاس بود و دیگری به تابع تعریف شده است)

def k\_means(X: np.array, k: int, max\_iters = 10000):

    centers = X[np.random.choice(range(len(X)), k, replace=False)]

    converged = False

    current\_iter = 0

    while (not converged) and (current\_iter < max\_iters):

        cluster\_list = [[] for i in range(len(centers))]

        for x in X:

            distances\_list = []

            for c in centers:

                distances\_list.append(get\_distance(c, x))

            cluster\_list[int(np.argmin(distances\_list))].append(x)

        cluster\_list = list((filter(None, cluster\_list)))

        prev\_centers = centers.copy()

        centers = []

        for j in range(len(cluster\_list)):

            centers.append(np.mean(cluster\_list[j], axis=0))

        pattern = np.abs(np.sum(prev\_centers) - np.sum(centers))

        # print('K-MEANS: ', int(pattern))

        converged = (pattern == 0)

        current\_iter += 1

    return np.array(centers), [np.std(x) for x in cluster\_list]

سپس داده های ورودی را همانطور که در دستور کار گفته شده بود ,تعریف میکنیم:

NUM\_SAMPLES = 100

X = np.random.uniform(0., 1., NUM\_SAMPLES)

X = np.sort(np.array(X), axis=0)

noise = np.random.uniform(-0.1, 0.1, NUM\_SAMPLES)

y = np.sin(2 \* np.pi \* X)  + noise

print(y.shape)

plt.plot(X,y)

print("x",X)

print("y",y)

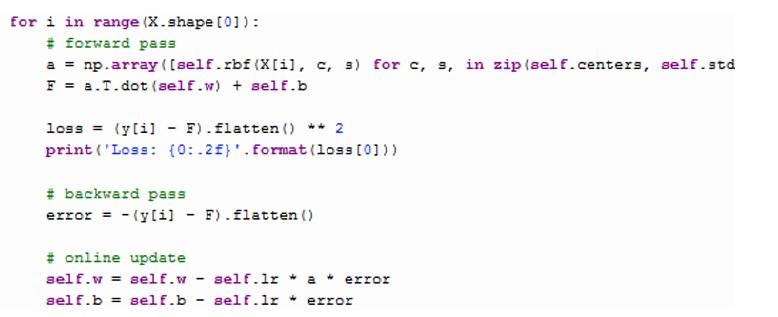
ما k means را تعریف کرده بودیم تا بتوانیم سری ای از داده ها را به آن بدهیم و centroids و cluster های آن و distance ها را در بیاوریم,پس این دیتا ی generate شده را به kmeans میدهیم و موارد مورد نظر را بدست می آوریم:

x = np.expand\_dims(X, 1)

# k = k\_means(x,2)

centers, stds = k\_means(x, 2)

سپس همان طور که در دستور کار گفته شده بود دیتا را train میکنیم:



حالا تابع predict هم میسازیم که در آن وزن ها را ضرب در ورودی ها میکنیم:

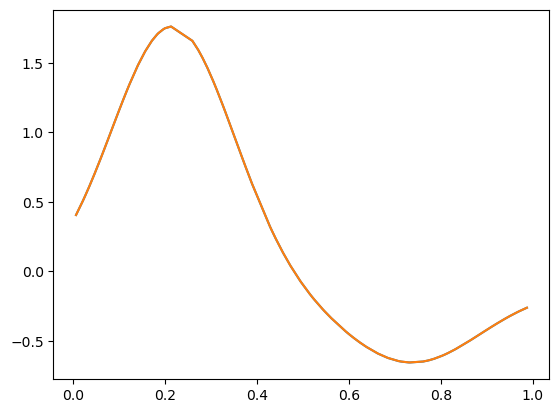
def predict(x):

        return sum([w \* rbf(x, c, s) + b for w, b, c, s in zip(w, b, centers, stds)])

حالا ار این تابع استفاده میکنیم:

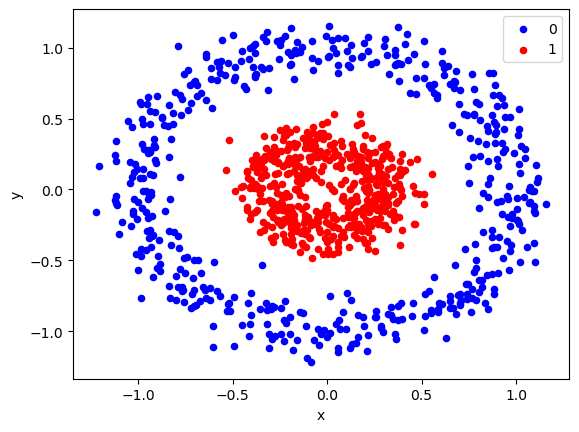
result = [predict(i) for i in x]

و سپس نمودار جدید را plot میکنیم:



تمرین:

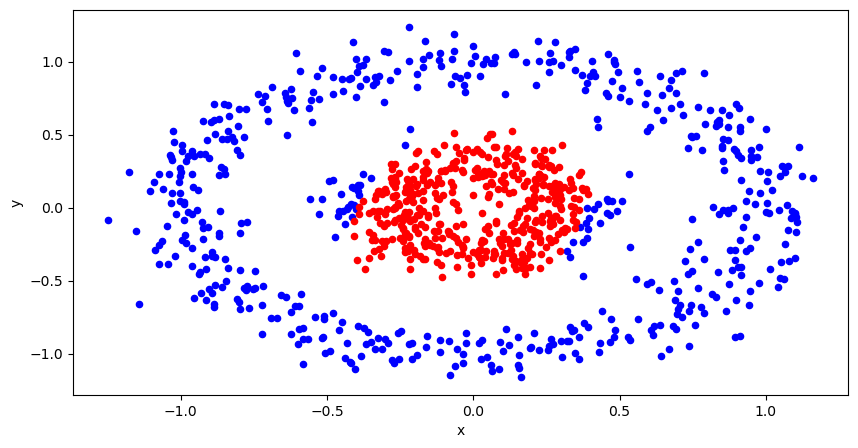
1-سپس دیتا ست جدید که در سوال از ما خواسته شده را درست میکنیم:



با استفاده از لایبری ه

from sklearn.datasets import make\_blobs, make\_circles

و پاسخی که میگیرریم:



و دیتای بعدی

