

به نام خدا

نام و نام خانوادگی:

محمد حسین بیاتی

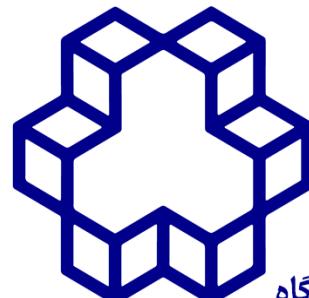
شماره دانشجویی:

۹۹۲۴۶۹۳

گزارش کار مینی پروژه شماره دو

استاد درس:

دکتر مهدی علیاری



دانشگاه  
خواجه نصیرالدین طوسی

K. N. Toosi University  
of Technology



## فهرست مطالب

۳	سوال ۱
۳	بخش ۱
۱۱	بخش ۲
۱۴	بخش ۳
۲۲	سوال ۲
۲۲	بخش ۱
۲۹	بخش ۲
۳۱	سوال ۳
۳۱	بخش ۱
۳۸	بخش ۲
۴۵	بخش ۳
۴۸	سوال ۴
۴۸	بخش ۱
۵۰	بخش ۲
۵۳	بخش ۳
۵۵	بخش ۴
۵۷	بخش ۵
۵۹	بخش ۶
۶۲	بخش ۷
۶۵	بخش ۸
۶۹	سوال ۵
۶۹	بخش ۱
۷۵	بخش ۲

## سوال ۱

مجموعه داده مربوط به این سوال را از طریق این پیوند دانلود کنید و در مراحل بعدی از آن استفاده کنید. ستون اول و دوم فایل CSV مربوط به این مجموعه داده، مربوط به ویژگی ها و ستون سوم آن مربوط به کلاس هر داده است.

### بخش ۱

داده ها را با نسبت ۸۰ به ۲۰ درصد به دو قسمت آموزش و آزمون تقسیم کنید. سپس با استفاده از قاعدة پرسپترون، یک نورون روی داده های مجموعه آموزشی، آموزش دهید (آستانه را دلخواه در نظر بگیرید).

به منظور زیاد نشدن فایل گزارش از آوردن کدهای تکراری که در مینی پروژه اول تکرار شده اند و همچنان چندین بار در این پروژه تکرار می شوند جلوگیری شده است. گزارش پله به پله نوشته شده است در نتیجه ارجاع مناسب به هر بخش کد داده شده است.

ابتدا کتابخانه های مورد نیاز را فراخوانی می کنیم.

نیاز است تا فایل CSV آپلود شده بر روی درایو را فراخوانی کنیم، بدین منظور به شکل زیر عمل می کنیم:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1HD99679NQbMNsOCFkV1uAxF13ikNM17L
#
https://drive.google.com/file/d/1HD99679NQbMNsOCFkV1uAxF13ikNM17L/view?usp=sharing
```

حال نیاز است تا همانند مطالبی که در حل تمرین گفته شد برای اعمال قاعده پرسپترون یک نورون را آموزش دهیم، برای اینکار ابتدا activation function ها را تعریف کرده:

```
def relu(x):
    return np.maximum(0, x)
def sigmoid(x):
    return 1/(1+np.exp(-x))
def tanh(x):
    return np.tanh(x)
```

سپس توابع loss مدنظر را تعریف کنیم:

```
def bce(y, y_hat):
    return np.mean(-(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)))
def bce(y, y_hat):
    return np.mean(-(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)))
```

و در انتهای برای محاسبه دقت تابع `accuracy` را به شکل زیر تعریف کنیم:

```
def accuracy(y, y_hat, t=0.5):
    y_hat = np.where(y_hat < t, 0, 1)
    acc = np.sum(y == y_hat) / len(y)
    return acc
```

حال یک نورون به کمک کلاس دقیقا مطابق آنچه که در کلاس حل تمرین گفته شد تعریف می کنیم و فرآپارتمترهای آن را مشخص می کنیم، در اینجا نیاز است تا چند تفاوت نسبت به مطالب کلاس حل تمرین ایجاد کنیم:

۱. از آنجا که قرار است آستانه را مقدار دلخواه تعریف کنیم پس دیگر مقدار آستانه را به روش گرادیان نزولی آپدیت نمی کنیم و این مقدار را به عنوان ورودی نورون دریافت می کنیم، پس نیاز است تا در کلاس آنرا رجیستر کنیم.

۲. برای رسم نمودار `decision region` نیاز است تا به داده های خام (داده هایی که `activation` بر روی آنها اعمال نشده است) داریم، پس نیاز است تا تابع `decision function` را نیز در کلاس نورون تعریف کنیم.

بنابراین کد نورون ما به شکل زیر خواهد بود:

```
class Neuron:

    def __init__(self, in_features, threshold, af=None, loss_fn=mse,
                 n_iter=100, eta=0.1, verbose=True):
        self.in_features = in_features
        # weight & bias
        self.w = np.random.randn(in_features, 1)
        self.threshold = threshold
        self.af = af
        self.loss_fn = loss_fn
        self.loss_hist = []
        self.w_grad = None
        self.n_iter = n_iter
        self.eta = eta
        self.verbose = verbose

    def predict(self, x):
        # x: [n_samples, in_features]
        y_hat = x @ self.w + self.threshold
        y_hat = y_hat if self.af is None else self.af(y_hat)
        return y_hat
```



```

def decision_function(self, x):
    # x: [n_samples, in_features]
    y_hat = x @ self.w + self.threshold
    return y_hat

def fit(self, x, y):
    for i in range(self.n_iter):
        y_hat = self.predict(x)
        loss = self.loss_fn(y, y_hat)
        self.loss_hist.append(loss)
        self.gradient(x, y, y_hat)
        self.gradient_descent()
        if self.verbose & (i % 10 == 0):
            print(f'Iter={i}, Loss={loss:.4f}')

def gradient(self, x, y, y_hat):
    self.w_grad = (x.T @ (y_hat - y)) / len(y)

def gradient_descent(self):
    self.w -= self.eta * self.threshold

def __repr__(self):
    af_name = self.af.__name__ if self.af is not None else None
    loss_fn_name = self.loss_fn.__name__ if self.loss_fn is not None
    else None
    return f'Neuron({self.in_features}, {self.threshold}, {af_name}, {loss_fn_name}, {self.n_iter}, {self.eta}, {self.verbose})'

def parameters(self):
    return {'w': self.w, 'threshold': self.threshold}

```

بنابراین فایل داده های CSV را در مجموعه دستا قرار داده، ستون آخر آنرا به عنوان هدف (target) قرار می دهیم و بقیه ستون ها را به عنوان feature ها قرار می دهیم. همچنین از آنجا که هدف های ما دو کلاسه (۱ و ۰) هستند و ما معمولاً با صفر و یک کار می کنیم، پس نیاز است تا هر جا که ۱- داریم، آنرا به صفر تبدیل کنیم، همینطور می توانیم ابعاد feature ها و هدف را مشاهده کنیم:

```

# Load the dataset
data = pd.read_csv('/content/Perceptron.csv')

# Separate features and target
X = data.iloc[:, :-1].values

```



```

y = data.iloc[:, -1].values

# Transforming y values from {-1, 1} to {0, 1}
y = np.where(y == -1, 0, 1)

# Display the dimensions of the dataset
print(f'Dimensions of the features: {X.shape}')
print(f'Dimensions of the target: {y.shape}')

```

نتایج:

```

Dimensions of the features: (400, 2)
Dimensions of the target: (400,)

```

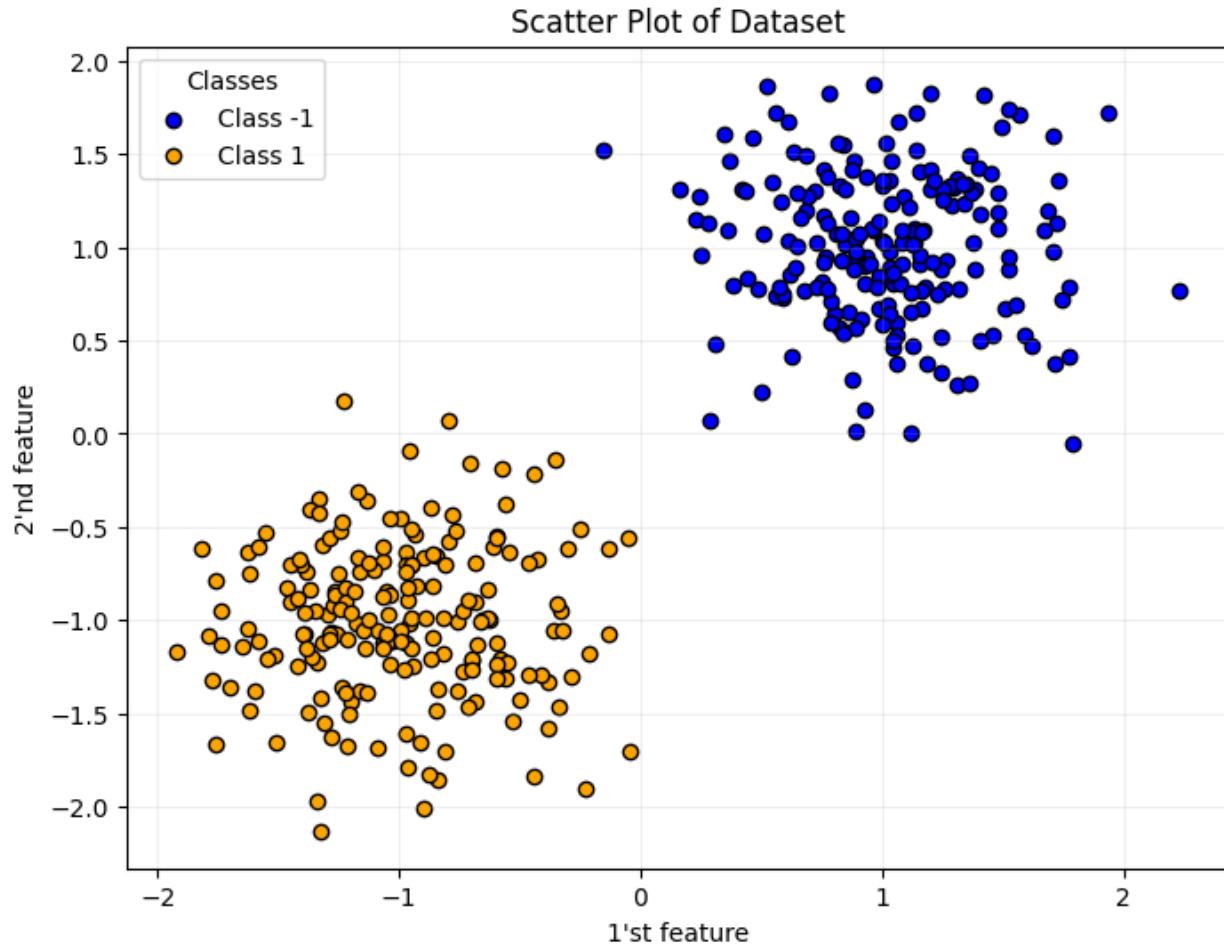
همانطور که مشخص است ۴۰۰ داده داریم که برای هر کدام ۲ ویژگی و یک هدف(در اینجا هدف ما کلاس بندی داده ها است.) داریم.

از آنجا که تنها دو ویژگی داریم، پس می توانیم به راحتی داده ها را در یک صفحه نمایش دهیم، اینکار را در مینی پروژه اول هم انجام دادیم پس از توضیحات کد آن صرف نظر می کنیم:

```

# Create a scatter plot
plt.figure(figsize=(8, 6)) # Set the figure size
scatter_class_0 = plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue',
label='Class -1', edgecolors='k', marker='o')
scatter_class_1 = plt.scatter(X[y == 1, 0], X[y == 1, 1], color='orange',
label='Class 1', edgecolors='k', marker='o')
plt.xlabel("1'st feature")
plt.ylabel("2'nd feature")
plt.title('Scatter Plot of Dataset') # Title for the plot
plt.legend(handles=[scatter_class_0, scatter_class_1],
title='Classes', loc="upper left")
plt.grid(alpha=0.2) # Display grid lines

```



همانطور که مشخص است داده ها را به راحتی می توان با یک خط از هم جدا کرد.

حال داده ها را به نسبت ۸۰ به ۲۰ برای مجموعه های train و test جدا می کنیم، برای اینکه خاصیت تکرار پذیری وجود داشته باشد از random\_state استفاده می کنیم، همچنین برای اینکه نظم داده ها از بین برود از shuffle کردن داده ها استفاده می کنیم و همچنین تعداد داده یکسانی از هر کلاس برای داده های تست انتخاب می کنیم، پس به صورت زیر عمل می کنیم:

```
# Splitting the dataset into the Training set and Test set (80/20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=93, stratify=y, shuffle=True)

# Display the dimensions of the training and testing sets
print(f'Dimensions of the training features: {X_train.shape}')
print(f'Dimensions of the training target: {y_train.shape}')
print(f'Dimensions of the testing features: {X_test.shape}')
print(f'Dimensions of the testing target: {y_test.shape}'')
```

نتایج:

```
Dimensions of the training features: (320, 2)
Dimensions of the training target: (320, 1)
Dimensions of the testing features: (80, 2)
Dimensions of the testing target: (80, 1)
```

حال برای آموزش داده ها نیاز است تا نورون مدل نظرمان را تعریف کنیم و بر داده های مجموعه آموزشی قیت کنیم و سپس عمل predict را برای مجموعه داده های ارزیابی انجام دهیم (مقدار آستانه را برابر ۰.۲ قرار می دهیم):

از آنجا که داده های ما دو کلاسه هستند می توانیم از تابع فعال ساز سیگموید و تابع خطا bce استفاده کنیم.

```
neuron = Neuron(in_features=2, threshold=0.1, af=sigmoid, loss_fn=bce,
n_iter=500, eta=0.1, verbose=True)
neuron.fit(X_train, y_train[:, None])
print(f'Neuron specification: {neuron}')
print(f'Neuron parameters: {neuron.parameters()}' )
```

نتایج:

```
Iter=0, Loss=1.672
Iter=10, Loss=1.51
Iter=20, Loss=1.354
Iter=30, Loss=1.206
Iter=40, Loss=1.067
Iter=50, Loss=0.9382
Iter=60, Loss=0.8192
Iter=70, Loss=0.7109
Iter=80, Loss=0.6136
Iter=90, Loss=0.527
Iter=100, Loss=0.4508
Iter=110, Loss=0.3845
Iter=120, Loss=0.3271
Iter=130, Loss=0.2778
Iter=140, Loss=0.2358
Iter=150, Loss=0.2
Iter=160, Loss=0.1697
Iter=170, Loss=0.1441
Iter=180, Loss=0.1224
Iter=190, Loss=0.1042
Iter=200, Loss=0.08879
Iter=210, Loss=0.0758
Iter=220, Loss=0.06484
Iter=230, Loss=0.05557
Iter=240, Loss=0.04774
Iter=250, Loss=0.0411
Iter=260, Loss=0.03547
Iter=270, Loss=0.03068
Iter=280, Loss=0.02661
```

```

Iter=290, Loss=0.02313
Iter=300, Loss=0.02016
Iter=310, Loss=0.01762
Iter=320, Loss=0.01543
Iter=330, Loss=0.01355
Iter=340, Loss=0.01193
Iter=350, Loss=0.01053
Iter=360, Loss=0.00932
Iter=370, Loss=0.008267
Iter=380, Loss=0.007351
Iter=390, Loss=0.006551
Iter=400, Loss=0.005852
Iter=410, Loss=0.00524
Iter=420, Loss=0.004702
Iter=430, Loss=0.004228
Iter=440, Loss=0.00381
Iter=450, Loss=0.00344
Iter=460, Loss=0.003112
Iter=470, Loss=0.00282
Iter=480, Loss=0.002561
Iter=490, Loss=0.002329
Neuron specification: Neuron(2, 0.1, sigmoid, bce, 500, 0.1, True)
Neuron parameters: {'w': array([-4.59977464], [-3.98132232]), 'threshold': 0.1}

```

همانطور که مشخص خطا های مد نظر چاپ شده (برای اینکه verbose را قرار دادیم) همینطور مشخصات نورون و پارامترهای آن نیز به ما نشان داده شده اند (وزن ها در اینجا آپدیت شده و مقدار به روز شده آنها نشان داده شده است اما در مدل تعريف شده مقدار آستانه به روز نمی شود).

اگر بخواهیم خطا از این مقدار هم کمتر شود می توانیم نرخ یادگیری و تعداد ایپاک ها را تغییر دهیم.

همینطور نمودار خطا را به راحتی می توان رسم کرد:

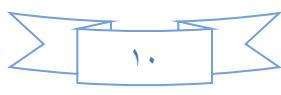
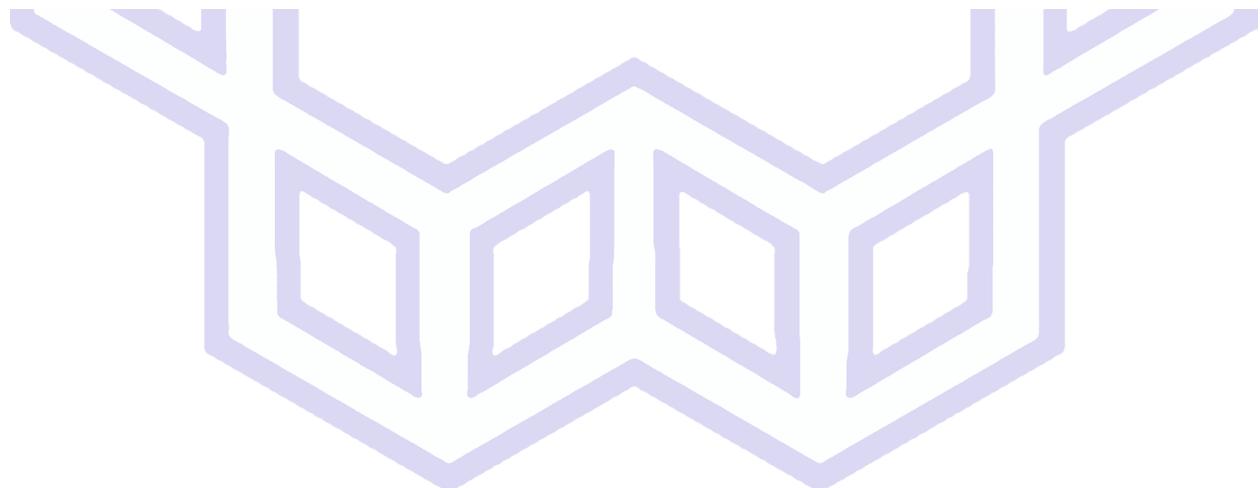
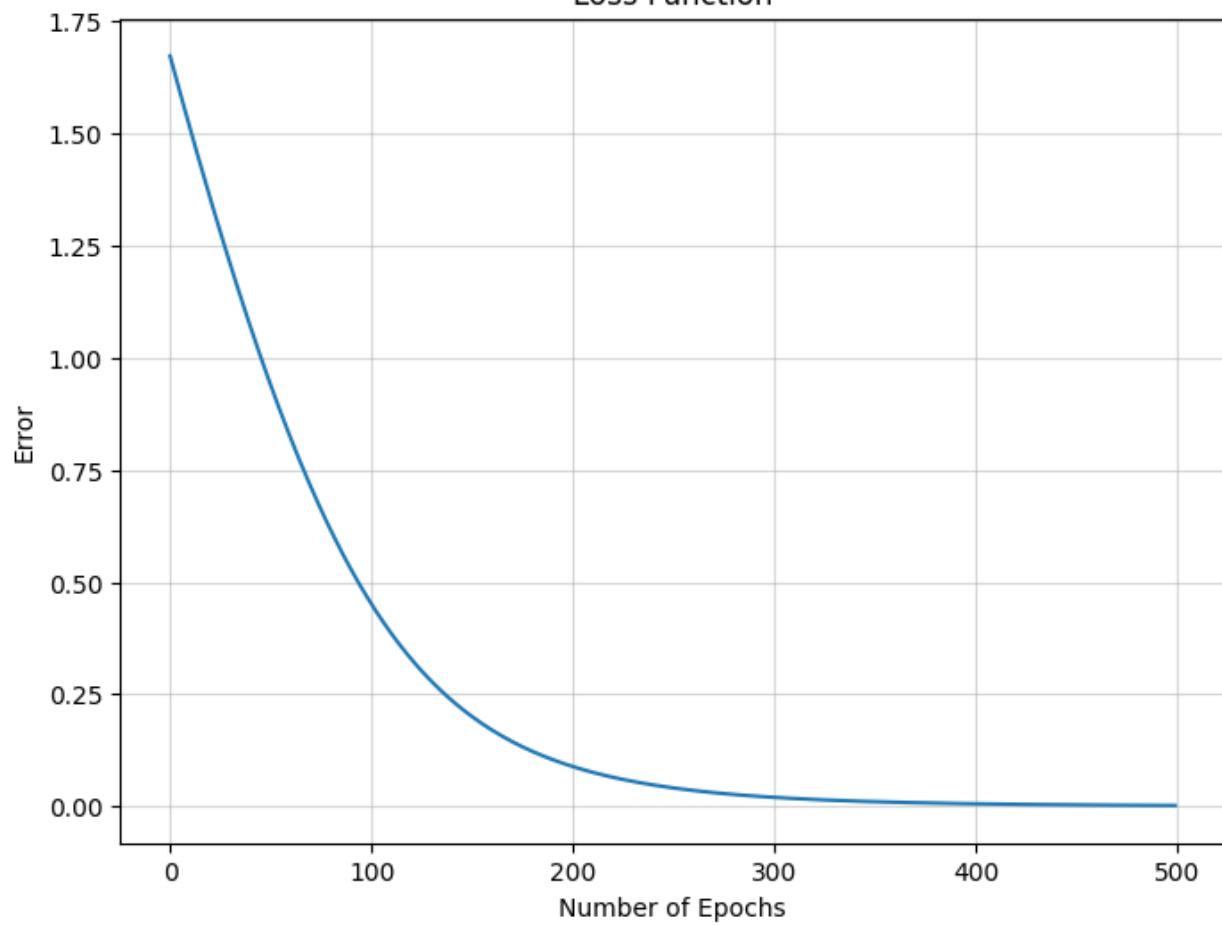
```

plt.figure(figsize=(8, 6))
plt.plot(neuron.loss_hist)
plt.xlabel("Number of Epochs")
plt.ylabel("Error")
plt.title('Loss Function')
plt.grid(alpha=0.5)

```

نتایج:

### Loss Function



## بخش ۲

نتیجه را روی داده های مجموعه آزمون نشان دهید و دقت را به دست آورید. برای داده های تست دوخط موازی جداکننده به دست آمده از قاعده پرسپترون را نمایش دهید و داده های تفکیک شده دو کلاس را با رنگ مجزا در Plot Scatter مشخص کنید.

برای اینکه نتیجه بر روی داده های آزمون اعمال شود از تابع `predict` که در نورون تعریف کردیم استفاده می کنیم:

```
y_hat = neuron.predict(X_test)  
accuracy(y_test[:, None], y_hat, t=0.5)
```

نتایج:

1.0

همانطور که مشخص است به دقت ۱۰۰ درصد رسیدیم.

همینطور می توانیم داده های پیش بینی شده و داده های واقعی را هم زمان ببینیم:

```
y_hat[:, 0], y_test
```

نتایج:

```
(array([[1.71667542e-06, 8.29455706e-04, 1.52893246e-04, 9.99981956e-01,  
       4.28565497e-04, 1.79597219e-01, 9.99999630e-01, 6.73894608e-05,  
       4.33115012e-02, 9.98628619e-01, 3.50330641e-04, 2.31050833e-05,  
       9.99957774e-01, 3.62770295e-04, 3.64913877e-04, 9.99557284e-01,  
       9.99908556e-01, 6.05963242e-03, 9.99361106e-01, 1.01713544e-05,  
       9.99692206e-01, 9.99918788e-01, 9.91120251e-01, 2.43531033e-05,  
       4.83174187e-05, 9.99300289e-01, 4.31731000e-04, 9.99995693e-01,  
       9.93333404e-01, 9.99979703e-01, 9.99538581e-01, 6.53441732e-05,  
       2.00401405e-04, 2.14420339e-03, 9.99635582e-01, 1.41213696e-03,  
       9.80808228e-01, 9.99966995e-01, 9.99735064e-01, 9.99509836e-01,  
       1.44332310e-03, 9.99980456e-01, 9.99323206e-01, 9.99859387e-01,  
       9.99043687e-01, 9.99864033e-01, 1.12313258e-05, 9.92294966e-01,  
       3.06636651e-06, 2.41177530e-05, 9.93609062e-01, 6.20655831e-04,  
       1.06765911e-03, 9.99989027e-01, 9.28124590e-03, 2.27569924e-03,  
       9.99992648e-01, 4.02371613e-06, 9.99682544e-01, 5.94193609e-06,  
       9.99375934e-01, 4.54036267e-04, 9.99931635e-01, 9.99911578e-01,  
       9.99427788e-01, 2.93958222e-04, 9.99898903e-01, 9.99961847e-01,  
       4.09281680e-04, 4.018444875e-03, 9.99992670e-01, 9.99905333e-01,  
       5.13740111e-04, 9.99661568e-01, 3.34065423e-04, 6.41703796e-03,  
       3.22796994e-04, 6.94205427e-05, 9.99961020e-01, 2.05335663e-05]),  
array([0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,  
      1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,  
      1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,  
      1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0])])
```

همینطور برای اینکه نمودار خط داده های تفکیک شده را به همراه داده های تست ببینیم، می توانیم از روش اولی که برای مینی پروژه اول گفته شد استفاده کنیم، همینطور از تابع `decision_function` تعریف شده در نورون برای نمایش خط استفاده می کنیم.

```
# Define the range of values for x1 and x2
x1_min, x2_min = X_test.min(0) - 0.5
x1_max, x2_max = X_test.max(0) + 0.5

# Generate a meshgrid of points
n = 500
x1r = np.linspace(x1_min, x1_max, n)
x2r = np.linspace(x2_min, x2_max, n)
x1m, x2m = np.meshgrid(x1r, x2r)

# Flatten the meshgrid points and predict the class labels
xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
ym = neuron.decision_function(xm) # Use decision_function instead of predict

# Plot the decision region
plt.contourf(x1m, x2m, ym.reshape(x1m.shape), levels=[-0.5, 0.5])

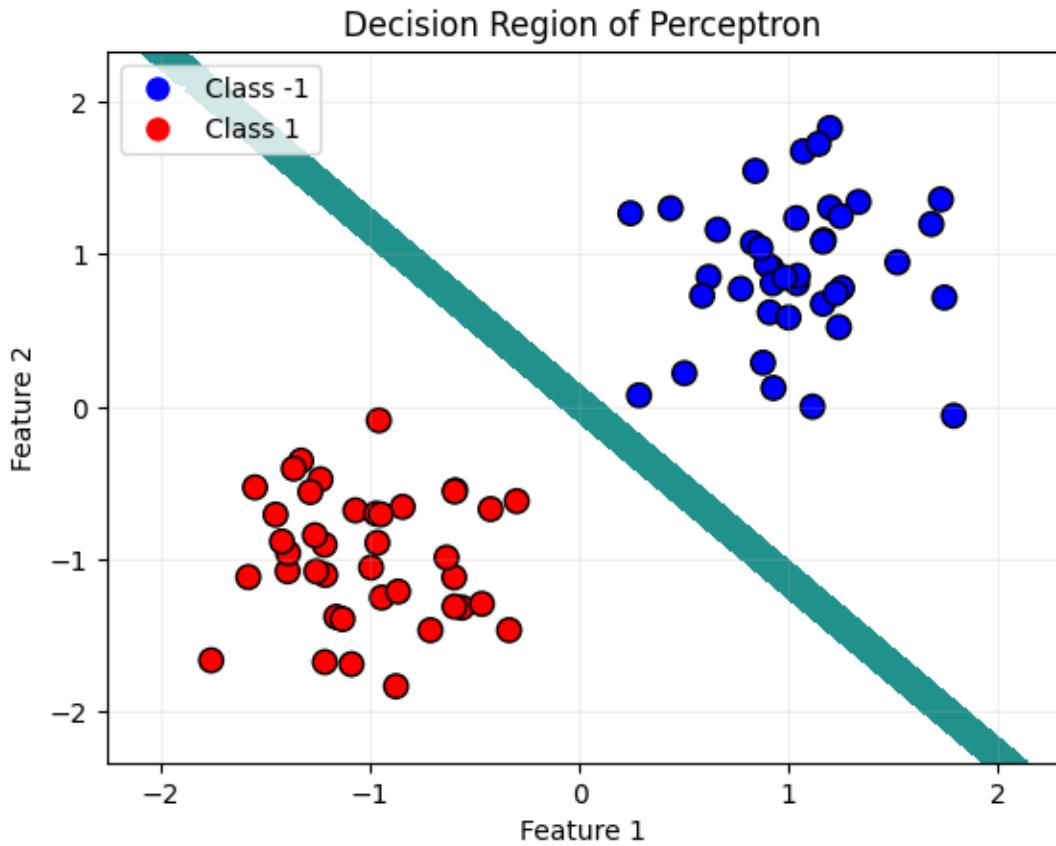
# Scatter plot for the test data with different markers
colors = np.array(['blue', 'red'])
plt.scatter(X_test[:, 0], X_test[:, 1], c=colors[y_test], edgecolors='k',
marker='o', s=80, linewidth=1, label='Test Data')

# Set labels and legend
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Region of Perceptron')
plt.legend()
plt.grid(alpha=0.2) # Display grid lines

# Add legend for class -1 and class 1
legend_elements = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='blue', markersize=10, label='Class -1'),
                   plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', markersize=10, label='Class 1')]
plt.legend(handles=legend_elements, loc='upper left')

# Show the plot
plt.show()
```

نتایج:



به کمک level تعریف شده در کد نمودار می توانیم به کمک دو خط گفته شده داده ها را از هم جدا کنیم، همینطور که مشخص است این خط می تواند داده ها را به طور کامل از هم جدا کند. خط اصلی جدا کننده دقیقاً خطی است که موازی با این دو خط است و فاصله یکسان تا هر کدام از دو خط دیگر دارد.

### بخش ۳

قسمت های «۱» و «۲» را با آستانه دیگری انجام داده و نتایج را با حالت قبل مقایسه کنید. تحلیل کنید که انتخاب آستانه در پرسپترون چه تأثیری روی نتایج طبقه بندی دارد. ضمن پیاده سازی تحلیل کنید که حذف بایاس چه تأثیری بر نتایج خواهد گذاشت.

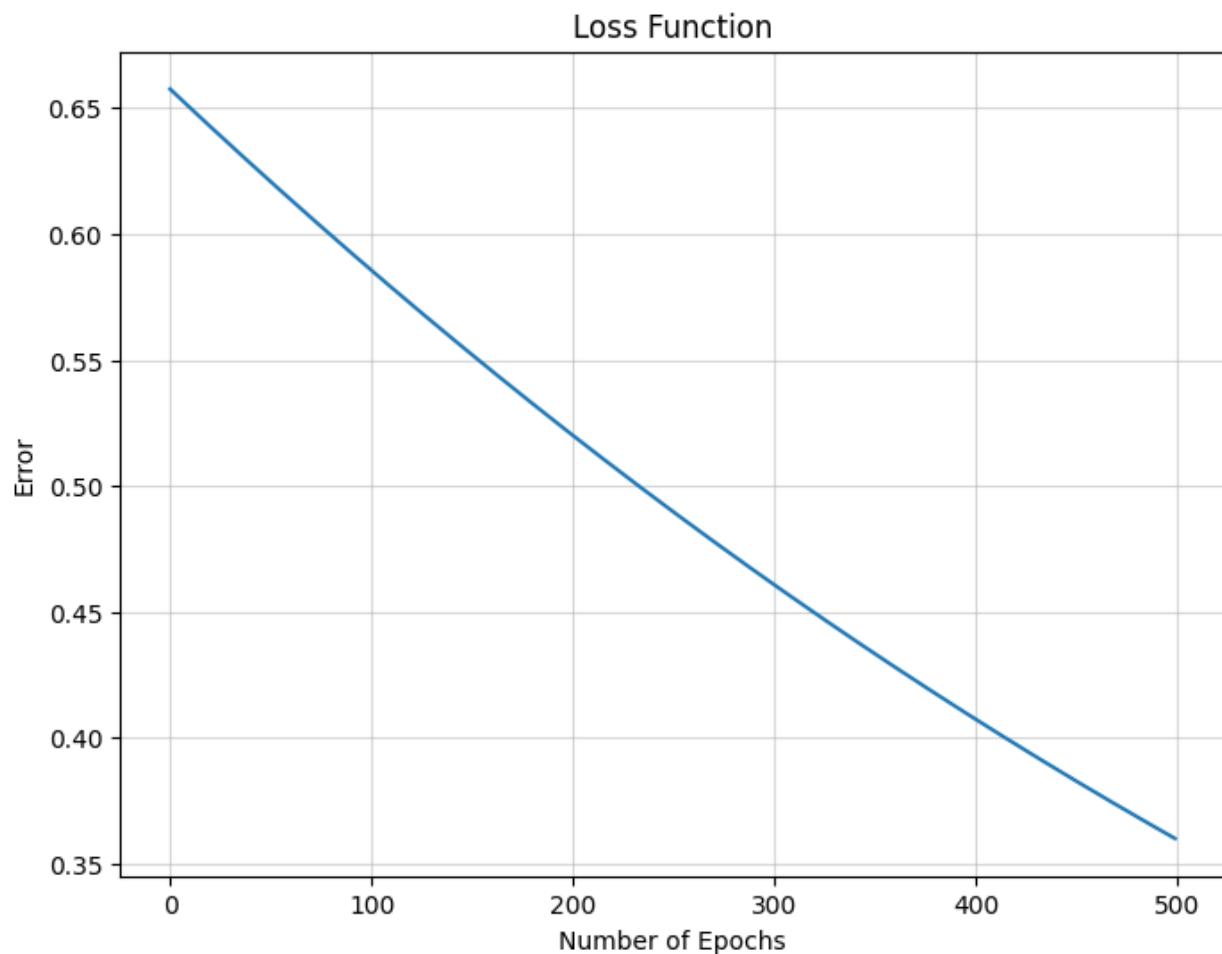
اگر مقدار بایاس را از ۰.۱ به مقدار دیگر مثلا ۰.۰۰۸ تغییر دهیم:

خطا و پارامترها:

```
Iter=0, Loss=0.6575
Iter=10, Loss=0.65
Iter=20, Loss=0.6426
Iter=30, Loss=0.6353
Iter=40, Loss=0.628
Iter=50, Loss=0.6207
Iter=60, Loss=0.6136
Iter=70, Loss=0.6065
Iter=80, Loss=0.5994
Iter=90, Loss=0.5925
Iter=100, Loss=0.5856
Iter=110, Loss=0.5787
Iter=120, Loss=0.572
Iter=130, Loss=0.5653
Iter=140, Loss=0.5586
Iter=150, Loss=0.552
Iter=160, Loss=0.5455
Iter=170, Loss=0.5391
Iter=180, Loss=0.5327
Iter=190, Loss=0.5263
Iter=200, Loss=0.5201
Iter=210, Loss=0.5139
Iter=220, Loss=0.5077
Iter=230, Loss=0.5016
Iter=240, Loss=0.4956
Iter=250, Loss=0.4897
Iter=260, Loss=0.4838
Iter=270, Loss=0.4779
Iter=280, Loss=0.4721
Iter=290, Loss=0.4664
Iter=300, Loss=0.4608
Iter=310, Loss=0.4552
Iter=320, Loss=0.4496
Iter=330, Loss=0.4442
Iter=340, Loss=0.4387
Iter=350, Loss=0.4334
Iter=360, Loss=0.4281
Iter=370, Loss=0.4228
Iter=380, Loss=0.4176
Iter=390, Loss=0.4125
Iter=400, Loss=0.4074
```

```
Iter=410, Loss=0.4024
Iter=420, Loss=0.3974
Iter=430, Loss=0.3925
Iter=440, Loss=0.3876
Iter=450, Loss=0.3828
Iter=460, Loss=0.3781
Iter=470, Loss=0.3734
Iter=480, Loss=0.3687
Iter=490, Loss=0.3642
Neuron specification: Neuron(2, 0.008, sigmoid, bce, 500, 0.1, True)
Neuron parameters: {'w': array([[ 0.51148957],
   [-1.44087921]]), 'threshold': 0.008}
```

نمودار خط:



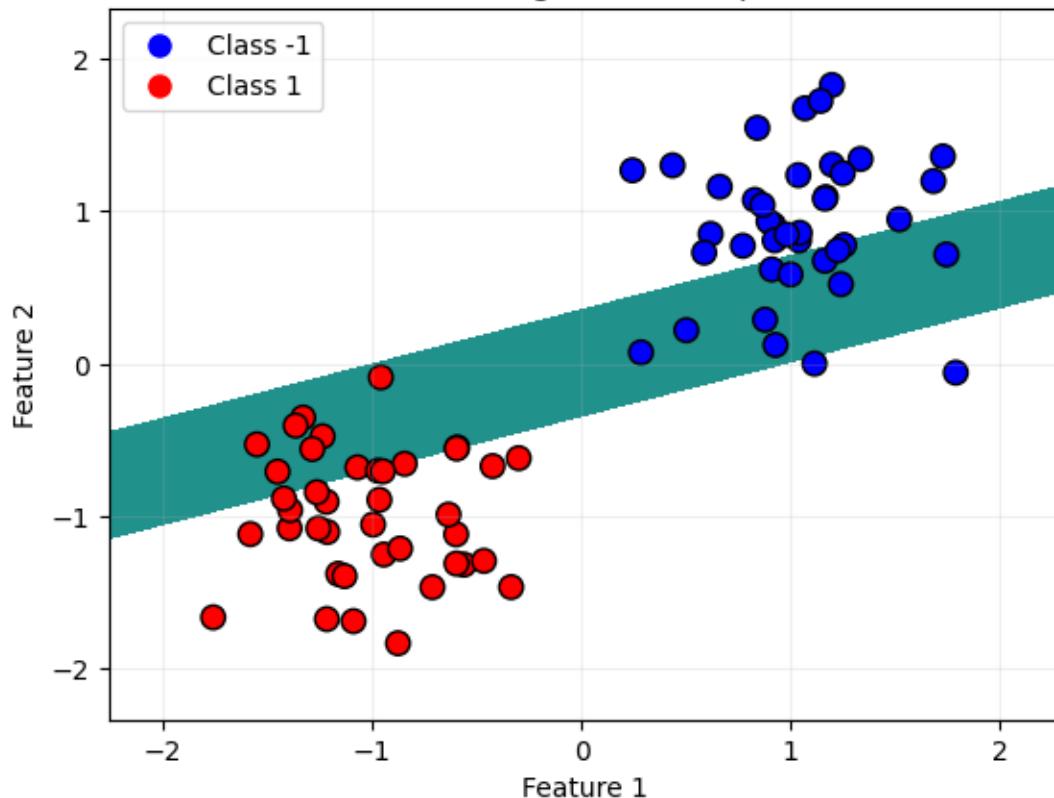
بررسی دقت:

0.8875

در این حالت دقیق‌تر می‌شود.

:Decision Region نمودار

Decision Region of Perceptron



مشخص است که دقیق‌تر کلاس بندی در اینجا کمتر شده است.

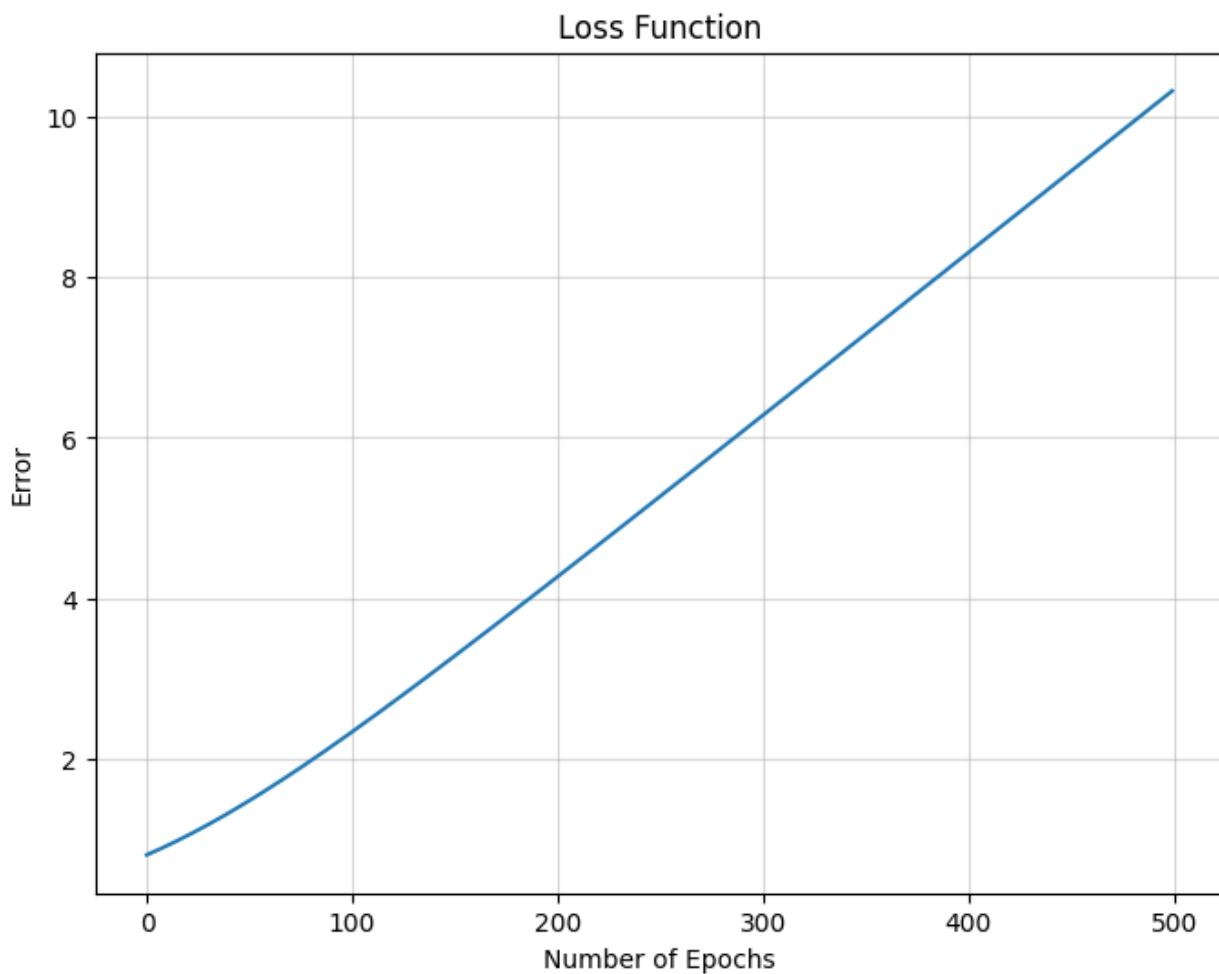
تغییر بازیابی از ۰.۱ به -۰.۱:

خطا و پارامترها:

```
Iter=0, Loss=0.8146
Iter=10, Loss=0.9309
Iter=20, Loss=1.057
Iter=30, Loss=1.193
Iter=40, Loss=1.338
Iter=50, Loss=1.491
Iter=60, Loss=1.651
Iter=70, Loss=1.817
Iter=80, Loss=1.989
Iter=90, Loss=2.165
Iter=100, Loss=2.346
Iter=110, Loss=2.53
Iter=120, Loss=2.717
```

```
Iter=130, Loss=2.906
Iter=140, Loss=3.098
Iter=150, Loss=3.291
Iter=160, Loss=3.486
Iter=170, Loss=3.682
Iter=180, Loss=3.879
Iter=190, Loss=4.077
Iter=200, Loss=4.275
Iter=210, Loss=4.475
Iter=220, Loss=4.674
Iter=230, Loss=4.874
Iter=240, Loss=5.075
Iter=250, Loss=5.276
Iter=260, Loss=5.477
Iter=270, Loss=5.679
Iter=280, Loss=5.88
Iter=290, Loss=6.082
Iter=300, Loss=6.284
Iter=310, Loss=6.486
Iter=320, Loss=6.688
Iter=330, Loss=6.89
Iter=340, Loss=7.092
Iter=350, Loss=7.295
Iter=360, Loss=7.497
Iter=370, Loss=7.7
Iter=380, Loss=7.902
Iter=390, Loss=8.105
Iter=400, Loss=8.307
Iter=410, Loss=8.51
Iter=420, Loss=8.713
Iter=430, Loss=8.915
Iter=440, Loss=9.118
Iter=450, Loss=9.321
Iter=460, Loss=9.524
Iter=470, Loss=9.726
Iter=480, Loss=9.929
Iter=490, Loss=10.13
Neuron specification: Neuron(2, -0.1, sigmoid, bce, 500, 0.1, True)
Neuron parameters: {'w': array([[4.34385462],
 [5.82858599]]), 'threshold': -0.1}
```

نمودار خطای:



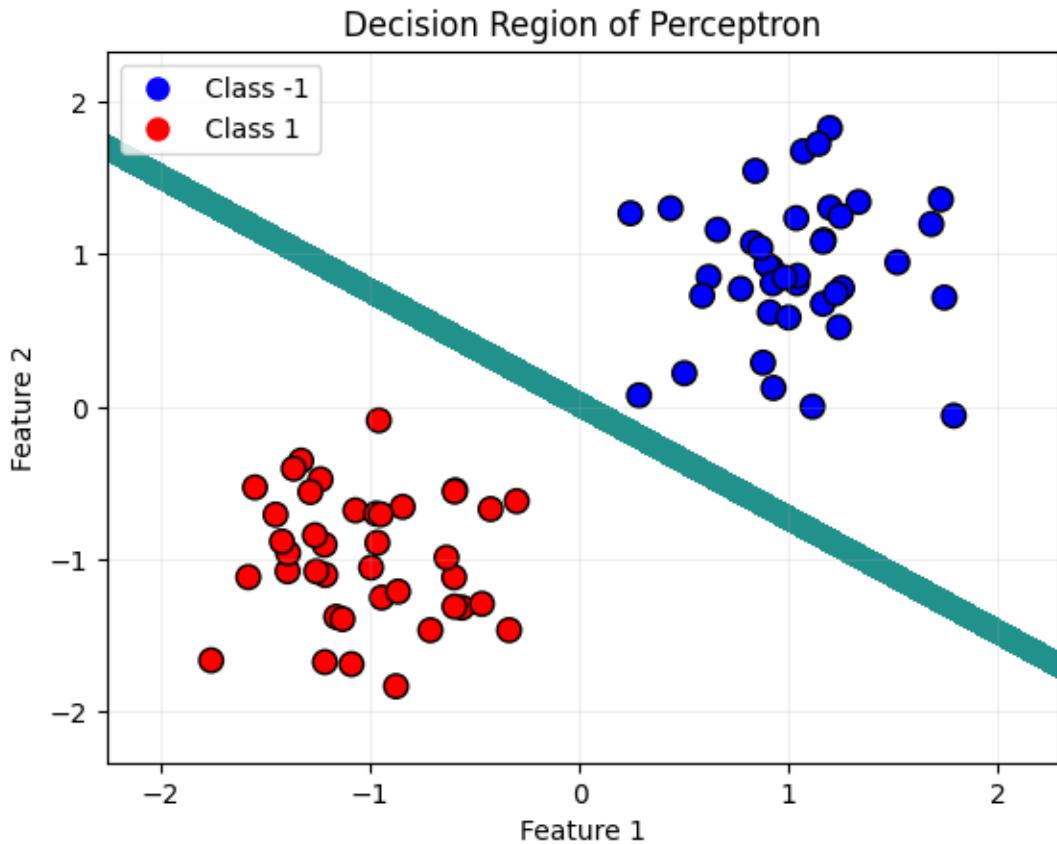
مشخص است که خطای رفته زیاد تر می شود.

بررسی دقت:

0.0

در این حالت دقت صفر می شود.

:Decision Region نمودار

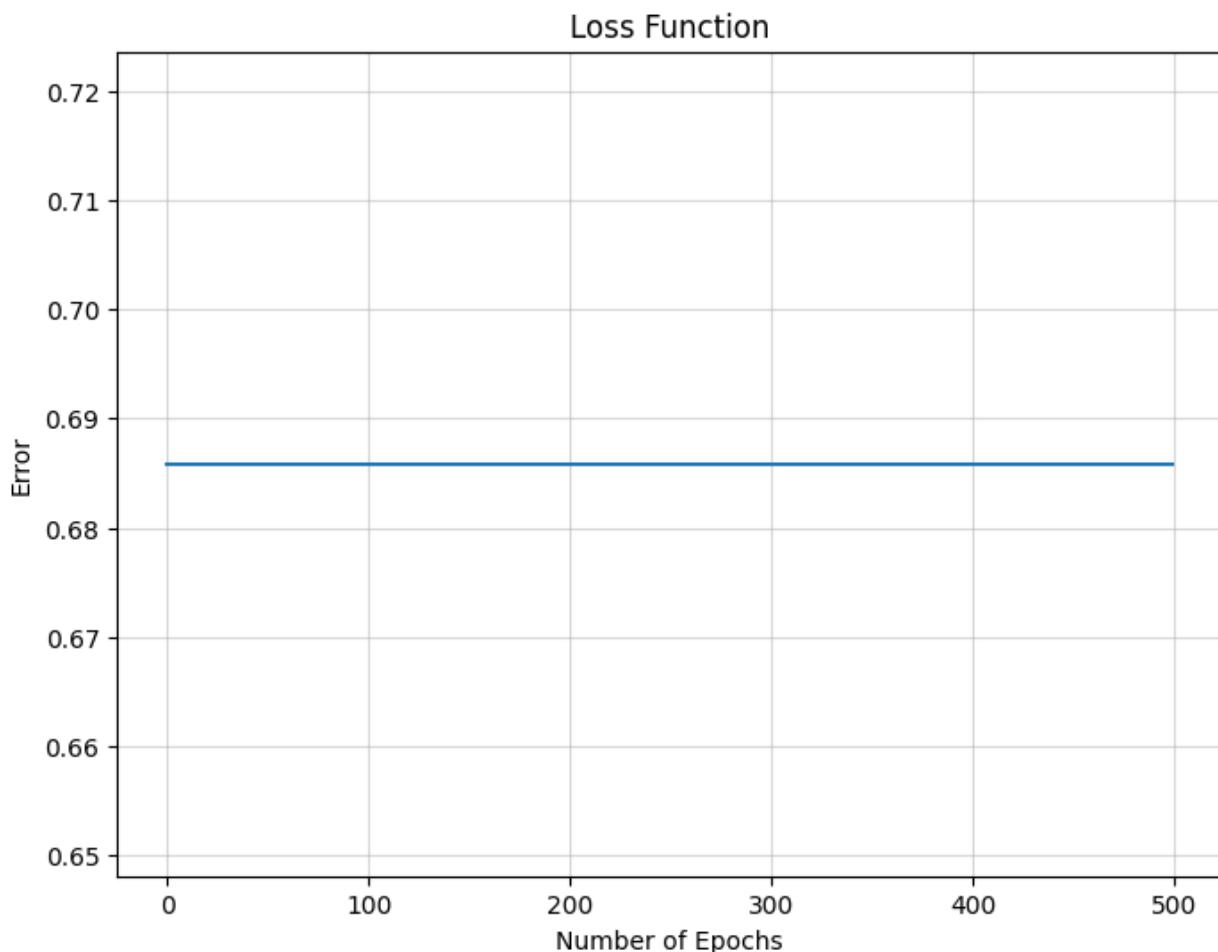


مشخص است که داده ها در اینجا برعکس کلاس بندی شده اند.

مشخص است که تغییر بایاس آن هم به این میزان کم چه تاثیر زیادی بر روی دقت دارد و دقت را از ۱۰۰ به صفر رسانده است، برای همین است که در کلاس نورون علاوه بر ضرایب  $w$  مقدار بایاس  $b$  را هم آپدیت می کنیم.

حذف مقدار بایاس:

نمودار خط:



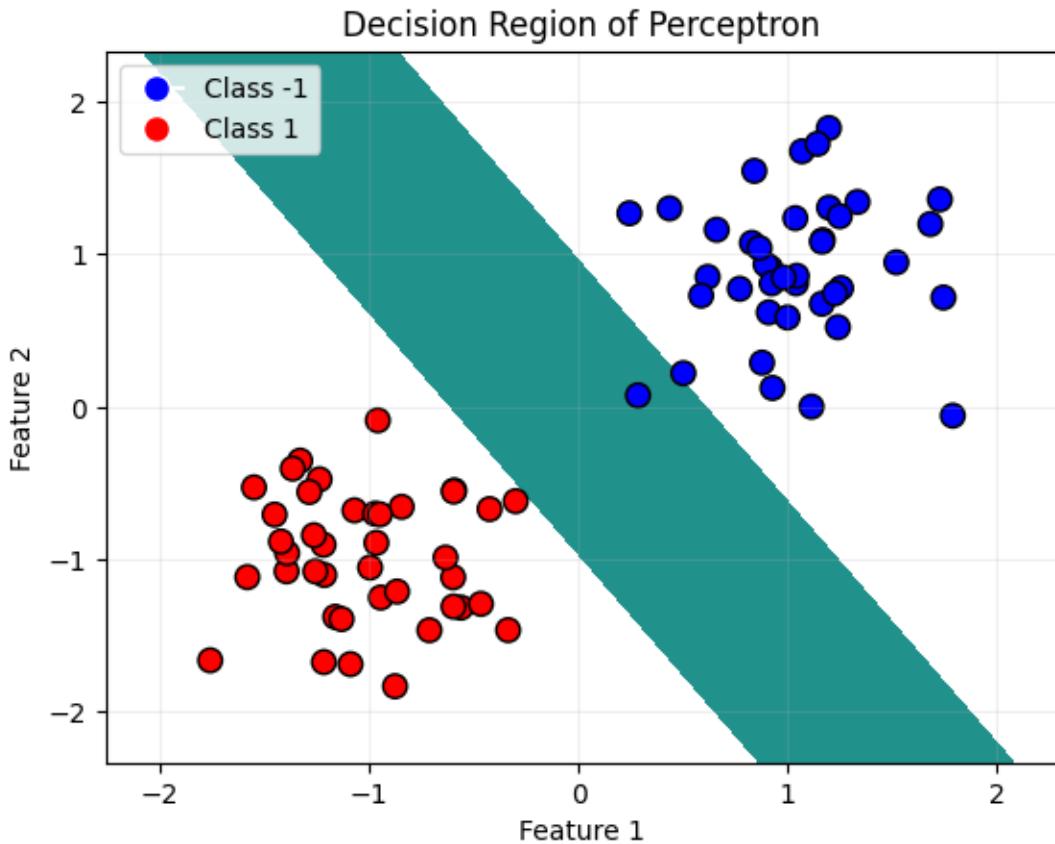
مشخص است که خطأ ثابت است و تغيير نمی کند.

بررسی دقت:

1.0

در این حالت دقت باز هم درصد است.

## نمودار Decision Region



مشخص است که داده ها باز هم به درستی طبقه بندی شده اند اما بستگی به داده ها حذف بایاس می تواند دقت را خراب کند.

حذف بایاس:

- ❖ حذف بایاس سبب می شود که خط جدا کننده داده ها از مبدأ بگذرد و قطعاً این خط نمی تواند هر مجموعه داده ای را به طور دقیق داده ها را از هم جدا کند در نتیجه بایاس به مدل اجازه می دهد که بیشترین انعطاف را در تصمیم گیری نسبت به مبدأ داشته باشد.
- ❖ فرض کنید یک مجموعه داده در یک نمونه تمام ویژگی هاییش برابر صفر باشد، اما کلاس آن کلاس ۱ باشد اگر هیچ مقدار بایاسی نداشته باشیم به هیچ وجه نمی تواند این داده را درست کلاس بندی کند.
- ❖ برخی دیتاستها و مسائل ممکن است نیازمند افزودن بایاس باشند تا مدل بتواند الگوهای پیچیده تری را یاد بگیرد.
- ❖ حذف بایاس ممکن است باعث محدودیت در توانایی مدل در یادگیری بازه ها و شبکه های مختلف شود.

## سوال ۲

### بخش ۱

یک ضرب کننده باینری بسازید که دو ورودی دوبیتی را گرفته و به کمک نورون Pitts-McCulloch توسعه یافته آن ها را ضرب کند. برای این کار به دو ورودی دوبیتی (در واقع چهار نورون برای همه ورودی ها) نیاز داریم. هم چنین چهار بیت خروجی (چهار نورون) مورد نیاز است. توجه شود که تمامی نورون های ورودی و خروجی باینری هستند (صفر و یک). ترتیب زمانی انجام عملیات در این سوال مهم نیست؛ بنابراین، نیازی به در نظر گرفتن تأخیر برای انجام عملیات نیست.

ضمن رسم جدول ورودی-خروجی، شبکه هر خروجی را به همراه توضیحات مختصری رسم کنید (نیازی به کدنویسی در این قسمت نیست). دقت داشته باشید که شبکه ای که برای هر خروجی رسم می کنید تا حد ممکن دارای کم ترین تعداد نورون و کم ترین آستانه باشد (تعداد نورون کم تر دارای اهمیت بالاتری نسبت به آستانه کوچک تر است). هم چنین توجه کنید که تمام شبکه برای یک خروجی دارای آستانه یکسان باشد.

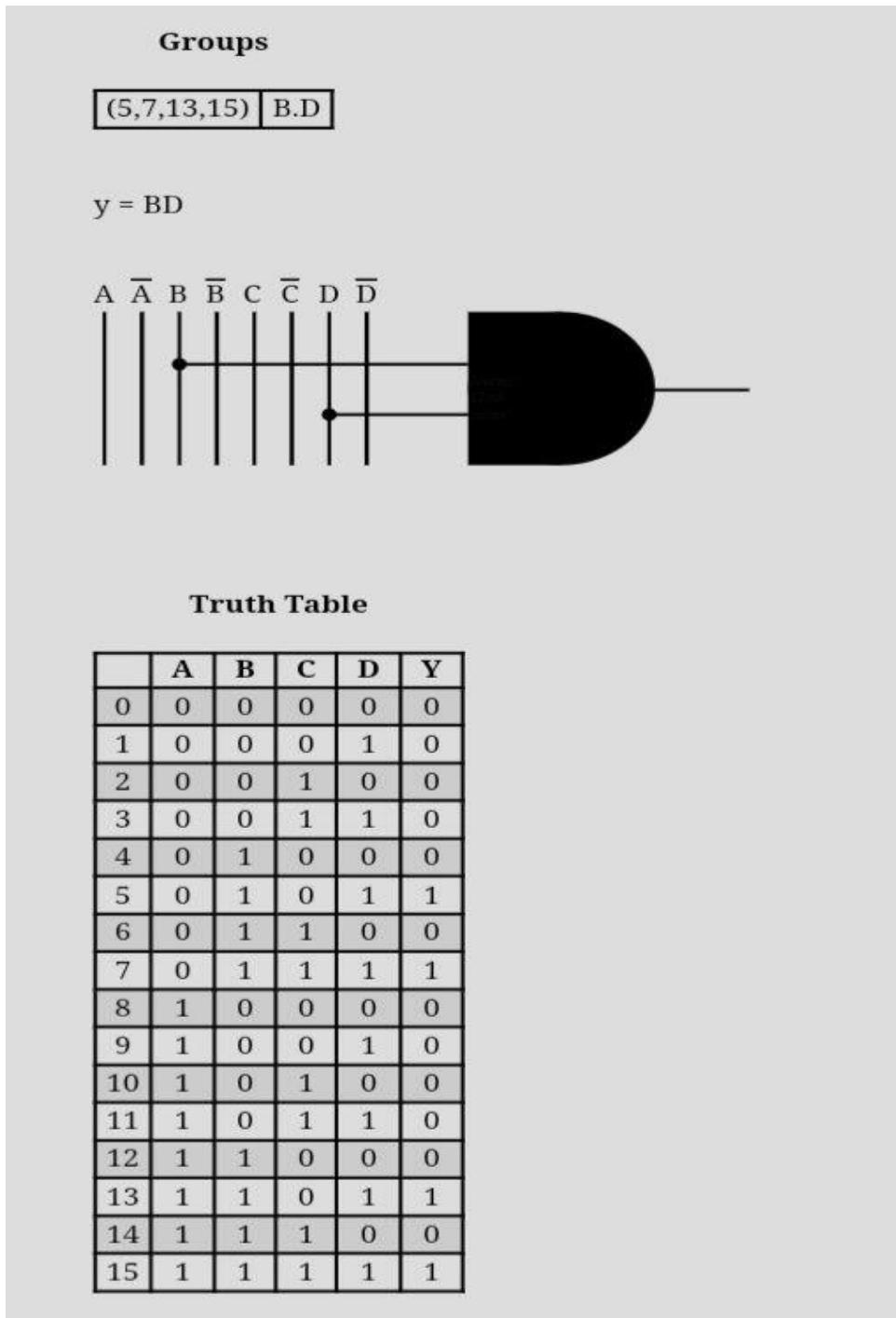
جدول درستی ضرب کننده باینری دو بیتی:

Table 1. Truth Table

INPUT A		INPUT B		OUTPUT			
A <sub>1</sub>	A <sub>0</sub>	B <sub>1</sub>	B <sub>0</sub>	C <sub>3</sub>	C <sub>2</sub>	C <sub>1</sub>	C <sub>0</sub>
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

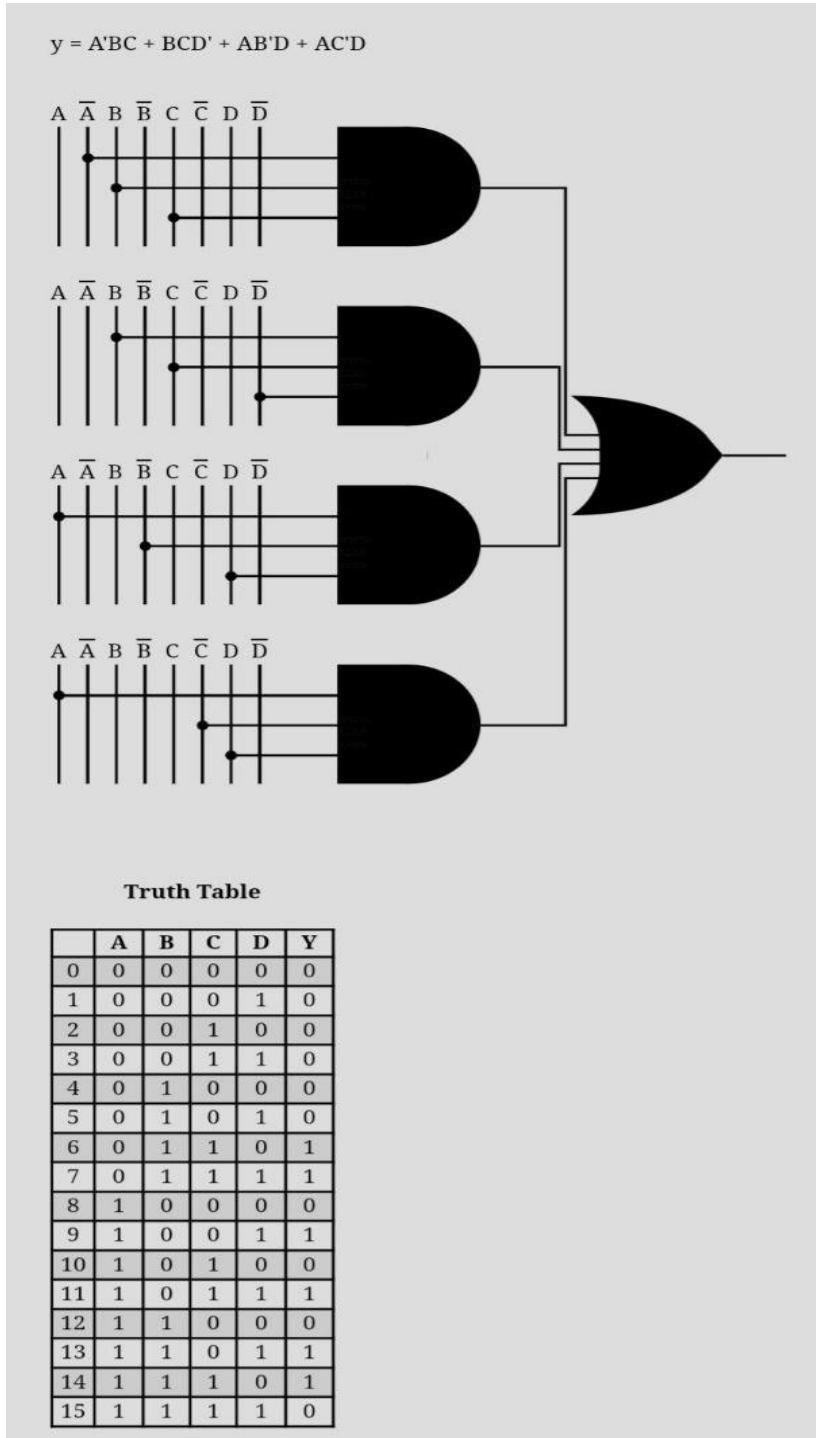
برای پیدا کردن خروجی از سایت 32x8.com استفاده می کنیم:

متناظر با خروجی C0 :



$$C_0 = A_0 \cdot B_0$$

متناظر با خروجی C1 :



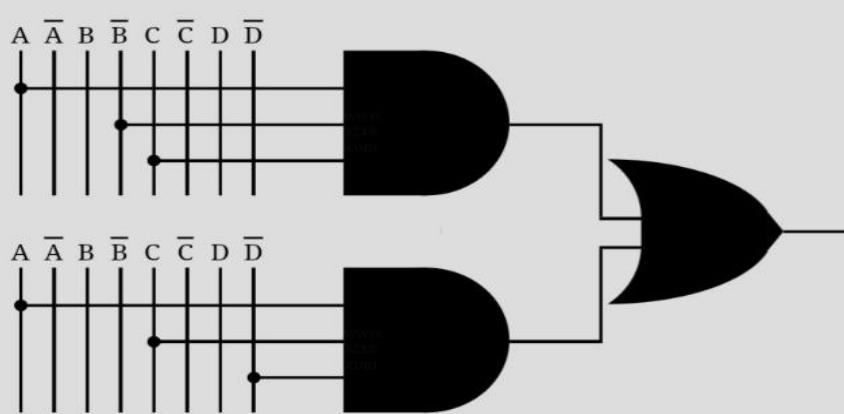
$$C_1 = \bar{A}_1 \cdot A_0 \cdot B_1 + A_0 \cdot B_1 \cdot \bar{B}_0 + A_1 \cdot \bar{A}_0 \cdot B_0 + A_1 \cdot \bar{B}_1 \cdot B_0$$

متناظر با خروجی C2 :

### Groups

(10,11)	$A \cdot \bar{B} \cdot C$
(10,14)	$A \cdot C \cdot \bar{D}$

$$y = AB'C + ACD'$$



### Truth Table

	A	B	C	D	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

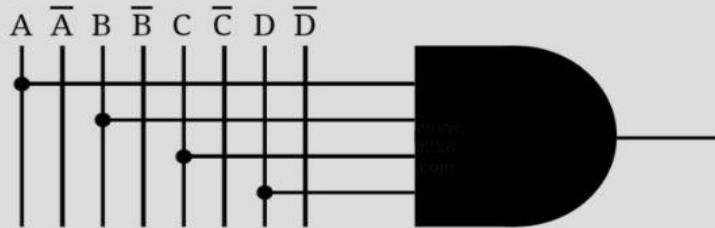
$$C_2 = A_1 \cdot \overline{A_0} \cdot B_1 + A_1 \cdot B_1 \cdot \overline{B_0}$$

متناظر با خروجی C3 :

### Groups

(15) A.B.C.D

$$y = ABCD$$



### Truth Table

	A	B	C	D	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1



$$C_3 = A_1 \cdot A_0 \cdot B_1 \cdot B_0$$

$$\varphi(n) = \begin{cases} 1 & n \geq 1 \\ 0 & n < 1 \end{cases}$$

$x_1$  and  $x_0$ :

$$= \boxed{\begin{matrix} 1 \\ 0 \end{matrix}}$$

$x_1$	$x_0$	$n_1, n_0$
0	0	0
0	1	0
1	0	1
1	1	1

$$\Rightarrow n_1 \text{ and } n_0 = \varphi(x_1 + x_0 - 1)$$

$n_p$  and  $n_r$  and  $n_1$  and  $n_0$ :

$n_p$	$n_r$	$n_1$	$n_0$	$n_p, n_r, n_1, n_0$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$$\begin{aligned} &\text{expand } n_p \text{ and } n_r \text{ and } n_1 \\ &n_p + n_r + n_1 + n_0 - 1 \end{aligned}$$

$n_1$  OR  $n_0$ :

$n_1$	$n_0$	$n_1 + n_0$
0	0	0
0	1	1
1	0	1
1	1	1

$$\rightarrow n_1 \text{ OR } n_0 = \varphi(n_1 + n_0 - 1)$$

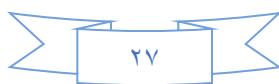
$$n_1 \bar{n}_0 y_1 + n_1 y_1 \bar{y}_0 : \rightarrow n_1 \bar{n}_0 y_1 + n_1 y_1 \bar{y}_0 = \varphi(n_1 + y_1 - n_0 - y_0 - 1)$$

$$n_1 \text{ and } n_0 = \varphi(n_1 + n_0 - 1)$$

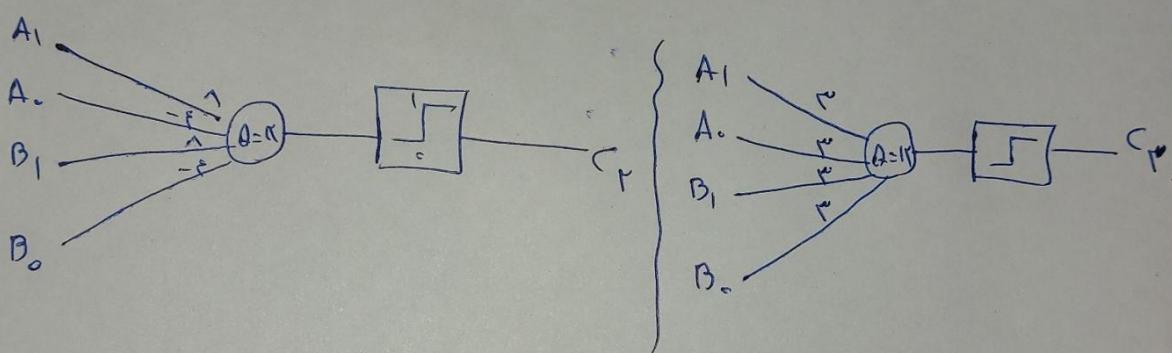
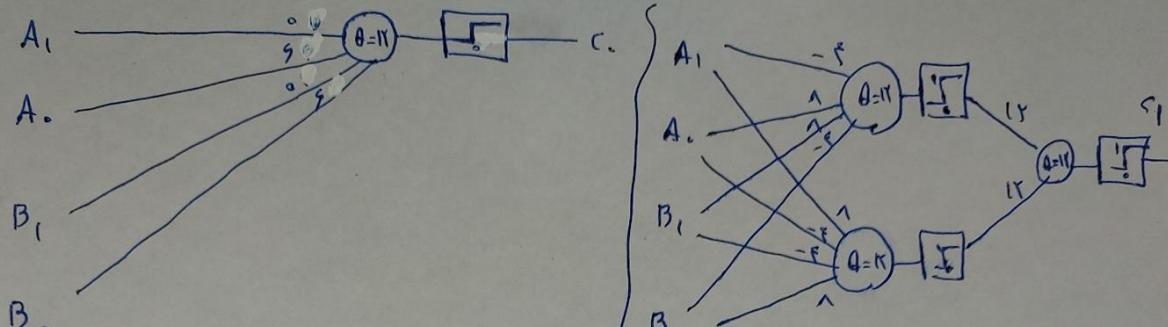
$$n_p \text{ and } n_r \text{ and } n_1 \text{ and } n_0 = \varphi(n_p + n_r + n_1 + n_0 - 1)$$

$$n_1 \text{ OR } n_0 = \varphi(n_1 + n_0 - 1)$$

$$n_1 \bar{n}_0 y_1 + n_1 y_1 \bar{y}_0 = \varphi(n_1 + y_1 - n_0 - y_0 - 1)$$

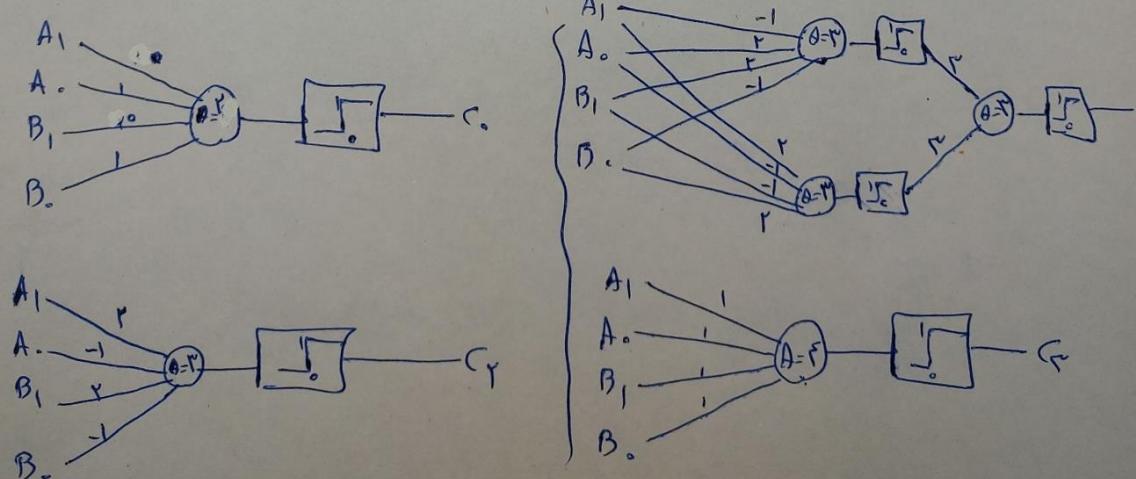


بنابراین پایه دستگاهی به صورت  $\rightarrow$  می‌باشد



این پایه دستگاهی دو قطب تعداد نومن / ادارد.

این طریق برای این بروکرهای آستانه ها کمیاب باشند اگر تراویث شده دو قطب تعداد آستانه ها را بخواهیم  
درزرسی آستانه ها کمیاب باشند طریق بروکر را سه



## بخش ۲

با استفاده از زبان پایتون شبکه های طراحی شده در قسمت «۱» را پیاده سازی کرده و تمامی حالات ممکن را به صورت مناسبی نشان دهید.

ابتدا کتابخانه های لازم را فراخوانی می کنیم:

```
#import library
import numpy as np
import itertools
```

سپس نورون muculloch pitts را به صورت گفته شده در کلاس تعریف می کنیم:

```
#define muculloch pitts
class McCulloch_Pitts_neuron():

    def __init__(self , weights , threshold):
        self.weights = weights      #define weights
        self.threshold = threshold   #define threshold

    def model(self , x):
        #define model with threshold
        if self.weights @ x >= self.threshold:
            return 1
        else:
            return 0
```

و طراحی مد نظر را به کمک ۶ نورون به صورت زیر تعریف می کنیم:

```
#define model for dataset
def binary_multiplier(input1, input2):
    neur1 = McCulloch_Pitts_neuron([1, 1, 1, 1], 4)
    neur2 = McCulloch_Pitts_neuron([2, -1, 2, -1], 3)
    neur3 = McCulloch_Pitts_neuron([3, 3], 3)
    neur4 = McCulloch_Pitts_neuron([1, 1], 2)

    M3 = neur1.model(np.array([input2[0], input2[1], input1[0], input1[1]]))
    M2 = neur2.model(np.array([input2[0], input2[1], input1[0], input1[1]]))
    M1_1 = neur2.model(np.array([input2[1], input2[0], input1[0],
    input1[1]]))
    M1_0 = neur2.model(np.array([input2[0], input2[1], input1[1],
    input1[0]]))
    M1 = neur3.model(np.array([M1_1, M1_0]))
    M0 = neur4.model(np.array([input2[1], input1[1]]))

    # 3 bit output
```

```
    return list([M3, M2, M1, M0])
```

نتایج:

```
import itertools
# inputs
input = [1, 0]
X1 = list(itertools.product(input, input))
X =list(itertools.product(X1, X1))

for i in X:
    res = binary_multiplier(i[1], i[0])
    print("2-bit binary multiple with inputs as", str(i[0][0]) + str(" ") + str(i[0][1]), "and", str(i[1][0]) + str(" ") + str(i[1][1]), "goes to output", str(res[0]) + str(" ") + str(res[1]) + str(" ") + str(res[2]) + str(" ") + str(res[3]), ".")
```

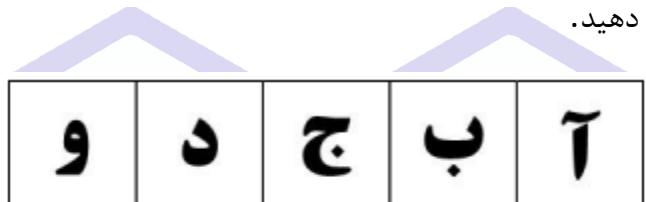
نتایج خروجی:

```
2-bit binary multiple with inputs as 1 1 and 1 1 goes to output 1 0 0 1 .
2-bit binary multiple with inputs as 1 1 and 1 0 goes to output 0 1 1 0 .
2-bit binary multiple with inputs as 1 1 and 0 1 goes to output 0 0 1 1 .
2-bit binary multiple with inputs as 1 1 and 0 0 goes to output 0 0 0 0 .
2-bit binary multiple with inputs as 1 0 and 1 1 goes to output 0 1 1 0 .
2-bit binary multiple with inputs as 1 0 and 1 0 goes to output 0 1 0 0 .
2-bit binary multiple with inputs as 1 0 and 0 1 goes to output 0 0 1 0 .
2-bit binary multiple with inputs as 1 0 and 0 0 goes to output 0 0 0 0 .
2-bit binary multiple with inputs as 0 1 and 1 1 goes to output 0 0 1 1 .
2-bit binary multiple with inputs as 0 1 and 1 0 goes to output 0 0 1 0 .
2-bit binary multiple with inputs as 0 1 and 0 1 goes to output 0 0 0 1 .
2-bit binary multiple with inputs as 0 1 and 0 0 goes to output 0 0 0 0 .
2-bit binary multiple with inputs as 0 0 and 1 1 goes to output 0 0 0 0 .
2-bit binary multiple with inputs as 0 0 and 1 0 goes to output 0 0 0 0 .
2-bit binary multiple with inputs as 0 0 and 0 1 goes to output 0 0 0 0 .
2-bit binary multiple with inputs as 0 0 and 0 0 goes to output 0 0 0 0 .
```

که همان نتایجی است که انتظار داشتیم.

### سوال ۳

به این دفترچه کد مراجعه کنید و با اجرای سلول اول، ۵ داده تصویری مربوط به حروف الفبای فارسی که در شکل ۲ نشان داده شده است را دریافت کنید و سپس به سوالات زیر پاسخ دهید. دقت داشته باشید که در هر مرحله ارائه توضیحات متى و دیداری مناسب لازم است. مثلا می توانید ورودی نویزی و خروجی پیش بینی شده را در یک تصویر در کنار هم قرار دهید.



شکل ۲: نمونه داده‌ها.

#### بخش ۱

دوتابع پایتونی در سلول های دوم و سوم این دفترچه کد نوشته شده اند. اولین تابع تصویر را در ورودی خود دریافت و به صورت نمایش باینری درمی آورد و دومین تابع با افزودن نویز به داده ها، داده های جدید نویزی تولید می کند. در مورد نحوه عملکرد هریک از این توابع توضیح دهید. هم چنین، می توانید این دستورات را به صورتی بهتر و کارآمدتر بازنویسی کنید.

پایتون از کتابخانه (PIL) Python Imaging Library استفاده می کند تا یک تصویر را به یک نمایش دودویی تبدیل کند بر اساس شدت پیکسل ها.

این کد پایتون یک الگوریتم ساده برای تشخیص تصویر متن است. زمانی که یک تصویر از یک متن گرفته می شود، این الگوریتم تلاش می کند با استفاده از تفکیک کردن فاصله ها و تشخیص حروف مختلف، متن را تشخیص دهد. به طور کلی، این الگوریتم از شبکه های عصبی کانولوشنال استفاده می کند که با استفاده از مدل های پیش آموزش دیده شده، می تواند تصویر را تجزیه کند و حروف را شناسایی کند.

کد تابع اول:

۱. وارد کردن کتابخانه ها: این کد از کتابخانه های مورد نیاز `Image` و `ImageDraw` از `PIL` و همچنین کتابخانه `random` استفاده می کند، اگرچه کتابخانه `random` در تابع ارائه شده استفاده نشده است.

## ۲. تابع `:convertImageToBinary`

- یک مسیر فایل (path) را به عنوان آرگومان دریافت می‌کند که تصویر ورودی را نمایان می‌کند.
- با استفاده از `Image.open(path)` فایل تصویر را باز می‌کند.
- یک شیء نقاشی (draw) برای دستکاری تصویر ایجاد می‌کند.
- عرض و ارتفاع تصویر را دریافت می‌کند.
- مقادیر پیکسل‌های تصویر را بارگذاری می‌کند.
- یک لیست (binary\_representation) را برای ذخیره نمایش دودویی تصویر مقداردهی اولیه می‌کند.

## ۳. Pixel Processing

- با استفاده از حلقه‌های تودرتو `for j in range(height)` و `for i in range(width)` از تمام پیکسل‌های تصویر عبور می‌کند.
- مقادیر RGB هر پیکسل را دریافت می‌کند.
- شدت کلی پیکسل را با جمع مقادیر قرمز، سبز و آبی محاسبه می‌کند.
- شدت کلی را با یک آستانه factor مقایسه می‌کند تا تصمیم بگیرد آیا پیکسل باید به عنوان سفید یا سیاه در نظر گرفته شود.
- اگر شدت کلی بیشتر از یک آستانه خاص باشد، رنگ پیکسل را به سفید (۰, ۰, ۰) تغییر می‌دهد و
- را به لیست binary\_representation اضافه می‌کند. در غیر این صورت، رنگ پیکسل را به سیاه (۲۵۵, ۲۵۵, ۲۵۵) تغییر می‌دهد و ۱ را به لیست اضافه می‌کند.
- با استفاده از `draw.point(i, j), (red, green, blue)` رنگ پیکسل را در تصویر بهروزرسانی می‌کند.

:return .۴

### object - draw نقاشی را حذف می کند .del

- لیست binary\_representation را بازمی گرداند که نمایش دودویی تصویر را نشان می دهد. در این نمایش، پیکسل های سفید به عنوان ۰ و پیکسل های سیاه به عنوان ۱ نمایش داده می شوند.

این تابع به طور اصلی تصویر را به یک فرمت دودویی براساس شدت پیکسل تبدیل می کند، جایی که پیکسل های سفید به عنوان ۰ و پیکسل های سیاه به عنوان ۱ نمایش داده می شوند، با در نظر گرفتن یک آستانه شدت مشخص. توجه داشته باشید که روش تعیین آستانه در اینجا ممکن است نیاز به تنظیم داشته باشد و بسته به تصاویری که در حال پردازش هستند، نمایش دودویی مطلوب را به دست آورید.

این کد پایتون یک برنامه برای ایجاد نویز در تصاویر اجرا می کند و تصاویر نویزی جدید را برای تصاویر ورودی ایجاد می کند. الگوریتم از کتابخانه PIL برای کار با تصاویر استفاده می کند و با اضافه کردن نویز به پیکسل های تصویر، تصاویر نویزی جدید ایجاد می شوند.

تابع دوم:

### 1. تابع: generateNoisyImages

- یک لیست از مسیرهای فایل های تصویر را تعیین می کند.
- با استفاده از حلقه for از هر مسیر فایل تصویر، یک تصویر نویزی جدید ایجاد می کند.
- نام فایل های تصویر نویزی جدید را گزارش می دهد و آنها را ذخیره می کند.

### 2. تابع: getNoisyBinaryImage

- این تابع یک تصویر ورودی را باز می کند.
- یک ابزار نقاشی برای دستکاری تصویر ایجاد می کند.
- عرض و ارتفاع تصویر را دریافت می کند.
- مقادیر پیکسل های تصویر را بارگذاری می کند.

- یک عامل برای معرفی نویز تعریف می‌کند.
- با استفاده از حلقه `for` از هر پیکسل تصویر عبور می‌کند و نویز را به مقادیر RGB هر پیکسل اضافه می‌کند.
- مقادیر RGB را مطمئن می‌شود که در محدوده معتبر (۰-۲۵۵) باقی مانده و در صورت عبور از این محدوده، به حداقل یا حداقل مقدار معتبر تنظیم می‌شود.

- تصویر نویزی را به عنوان یک فایل جدید ذخیره می‌کند.

### ۳. تابع : generateNoisyImages

- ابتدا مسیرهای فایل‌های تصویر ورودی را مشخص می‌کند.
- سپس از هر مسیر فایل تصویر، تابع `getNoisyBinaryImage` را فراخوانی می‌کند تا یک تصویر نویزی جدید بسازد.
- نام و مسیر فایل‌های تصویر نویزی جدید را چاپ می‌کند.

در نهایت، این کد به صورت خاص نویز به هر پیکسل تصویر اضافه می‌کند و تصاویر نویزی را به عنوان خروجی ایجاد می‌کند.

به صورت زیر می‌توانیم تابع دوم را ارتقا دهیم:

```
from PIL import Image, ImageDraw
import random

def getNoisyBinaryImage(input_path, output_path, num_missing_points,
conversion_percentage):
    """
    Add noise to an image, generate missing points, and save it as a new
    file.

    Args:
        input_path (str): The file path to the input image.
        output_path (str): The file path to save the noisy image.
        num_missing_points (int): The number of missing points to
        generate.
```

```
    conversion_percentage (float): The percentage of black pixels to
convert to white.

"""
# Open the input image.
image = Image.open(input_path)

# Create a drawing tool for manipulating the image.
draw = ImageDraw.Draw(image)

# Determine the image's width and height in pixels.
width = image.size[0]
height = image.size[1]

# Load pixel values for the image.
pix = image.load()

# Define a factor for introducing noise.
noise_factor = 5

# Loop through all pixels in the image.
for i in range(width):
    for j in range(height):
        # Generate a random noise value within the specified factor.
        rand = random.randint(-noise_factor, noise_factor)

        # Add the noise to the Red, Green, and Blue (RGB) values of
the pixel.
        red = pix[i, j][0] + rand
        green = pix[i, j][1] + rand
        blue = pix[i, j][2] + rand

        # Ensure that RGB values stay within the valid range (0-255).
        if red < 0:
            red = 0
        if green < 0:
            green = 0
        if blue < 0:
            blue = 0
        if red > 255:
            red = 255
        if green > 255:
            green = 255
        if blue > 255:
            blue = 255
```



```

        # Convert some black pixels to white based on the conversion
percentage.
        if (red, green, blue) == (0, 0, 0) and random.random() <
conversion_percentage:
            red, green, blue = 255, 255, 255

        # Set the pixel color accordingly.
        draw.point((i, j), (red, green, blue))

    # Generate missing points in the image.
    for _ in range(num_missing_points):
        x = random.randint(0, width - 1)
        y = random.randint(0, height - 1)
        draw.point((x, y), (255, 255, 255)) # Set the missing point to
white

    # Save the noisy image as a file.
    image.save(output_path, "JPEG")

    # Clean up the drawing tool.
    del draw

from PIL import Image, ImageDraw
import random

def generateNoisyImages():
    # List of image file paths
    image_paths = [
        "/content/1.jpg",
        "/content/2.jpg",
        "/content/3.jpg",
        "/content/4.jpg",
        "/content/5.jpg"
    ]

    for i, image_path in enumerate(image_paths, start=1):
        noisy_image_path = f"/content/noisy{i}.jpg"
        # Specify the number of missing points and conversion percentage
here
        getNoisyBinaryImage(image_path, noisy_image_path,
num_missing_points=500, conversion_percentage=0.1)
        print(f"Noisy image for {image_path} generated and saved as
{noisy_image_path}")

# Generate noisy images with missing points and black-to-white conversion

```



```
generateNoisyImages()
```

تفاوت این کد با کد قبلی در این است که:

کد اول یک تابع به نام `getNoisyBinaryImage` دارد که برای ایجاد تصاویر نویزی از تصاویر ورودی استفاده می‌شود. این تابع تصاویر ورودی را با افزودن نویز به هر پیکسل تصویر نویزی می‌کند. در اینجا هیچ نقطه‌ای از تصویر حذف یا تغییر رنگ نمی‌شود.

کد دوم یک تابع با نام `getNoisyBinaryImage` دیگر دارد که نسخه پیشرفته‌تری از نویزی کردن تصاویر است. این تابع علاوه بر افزودن نویز به تصویر، تعداد مشخصی از نقاط تصویر را حذف (`missing points`) و یک درصد معین از پیکسل‌های سیاه را به سفید تبدیل می‌کند. این تغییرات باعث تولید تصاویر نویزی با ویژگی‌های بیشتر می‌شود.

در کد نهایی شما، از تابع دوم برای تولید تصاویر نویزی با نقاط حذف شده و تبدیل بلک به وايت با احتمال مشخص استفاده می‌شود. این امکانات باعث تولید تصاویر نویزی با خصوصیات متنوع‌تر می‌شود

## بخش ۲

یک شبکه عصبی (همینگ یا هاپفیلد) طراحی کنید که با اعمال ورودی دارای میزان مشخصی نویز برای هر یک از داده ها، خروجی متناسب با آن داده نویزی را بیابد. میزان نویز را تا حدی که شبکه شما ناموفق عمل کند افزایش دهید و نتایج را مقایسه و تحلیل کنید.

در این حالت ابتدا شبکه مورد نظر را ایجاد می کنیم و سپس پارامترهای نویز را آنقدر تغییر می دهیم تا ببینیم شبکه تا چه میزان تصاویر نویزی را می تواند تشخیص دهد. ابتدا یک شبکه همینگ ایجاد کرده مطابق شبکه تعریف شده در کلاس حل تمرین داریم:

```
from pylab import *
from math import sqrt
import matplotlib.pyplot as plt
import os

# Define the path to the input image
IMAGE_PATH = "/content/noisy3.jpg"

def show(matrix):
    """
    Display a matrix in a formatted manner.

    Args:
        matrix (list of lists): The matrix to be displayed.
    """
    for j in range(len(matrix)):
        for i in range(len(matrix[0])):
            print("{:.3f}".format(matrix[j][i]), end=" ")
        print(sep="")

def change(vector, a, b):
    """
    Transform a vector into a matrix of specified dimensions.

    Args:
        vector (list): The vector to be transformed.
        a (int): The number of columns in the resulting matrix.
        b (int): The number of rows in the resulting matrix.

    Returns:
        list of lists: The transformed matrix.
    """
    matrix = [[0 for j in range(a)] for i in range(b)]
    for i in range(len(vector)):
        matrix[i % b][i // b] = vector[i]
    return matrix
```

```

k = 0
j = 0
while k < b:
    i = 0
    while i < a:
        matrix[k][i] = vector[j]
        j += 1
        i += 1
    k += 1
return matrix

def product(matrix, vector, T):
    """
    Multiply a matrix by a vector.

    Args:
        matrix (list of lists): The matrix to be multiplied.
        vector (list): The vector to be multiplied.
        T (float): The threshold parameter for the activation function.

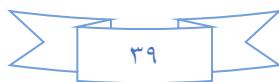
    Returns:
        list: The resulting vector after multiplication.
    """
    result_vector = []
    for i in range(len(matrix)):
        x = 0
        for j in range(len(vector)):
            x = x + matrix[i][j] * vector[j]
        result_vector.append((x + T))
    return result_vector

def action(vector, T, Emax):
    """
    Activation function to process a vector.

    Args:
        vector (list): The input vector to be processed.
        T (float): The threshold parameter for the activation function.
        Emax (float): The maximum allowable value for the difference in
                      output vectors between consecutive iterations.

    Returns:
        list: The output vector after activation.
    """
    result_vector = []

```



```

for value in vector:
    if value <= 0:
        result_vector.append(0)
    elif 0 < value <= T:
        result_vector.append(Emax * value)
    elif value > T:
        result_vector.append(T)
return result_vector

def mysum(vector, j):
    """
    Calculate the sum of vector values excluding the element at index j.

    Args:
        vector (list): The input vector.
        j (int): The index of the element to be excluded from the sum.

    Returns:
        float: The sum of vector values with the element at index j
    excluded.
    """
    p = 0
    total_sum = 0
    while p < len(vector):
        if p != j:
            total_sum = total_sum + vector[p]
        p += 1
    return total_sum

def norm(vector, p):
    """
    Calculate the difference between two vectors and compute the norm of
    the resulting vector.

    Args:
        vector (list): The first vector.
        p (list): The second vector for subtraction.

    Returns:
        float: The Euclidean norm of the difference between the two
    vectors.
    """
    difference = []
    for i in range(len(vector)):
        difference.append(vector[i] - p[i])

```



```

sum = 0
for element in difference:
    sum += element * element
return sqrt(sum)

# List of paths to example images
path = [
    '/content/1.jpg',
    '/content/2.jpg',
    '/content/3.jpg',
    '/content/4.jpg',
    '/content/5.jpg',
]

x = [] # Binary representations of example images
print(os.path.basename(IMAGE_PATH))

# Convert and store binary representations of example images
for i in path:
    x.append(convertImageToBinary(i))

y = convertImageToBinary(IMAGE_PATH) # Binary representation of the input
image
entr = y
k = len(x) # Number of example images
a = 96 # Number of columns in the transformed matrix
b = 96 # Number of rows in the transformed matrix
entr = y
q = change(y, a, b) # Transformation of the input image into a matrix
plt.matshow(q)
plt.colorbar()

m = len(x[0])
w = [[(x[i][j]) / 2 for j in range(m)] for i in range(k)] # Weight matrix
T = m / 2 # Activation function threshold parameter
e = round(1 / len(x), 1)
E = [[0 for j in range(k)] for i in range(k)] # Synaptic connection
matrix
Emax = 0.000001 # Maximum allowable difference norm between output
vectors in consecutive iterations
U = 1 / Emax

# Set values for the synaptic connection matrix
for i in range(k):
    for j in range(k):

```



```

        if j == i:
            E[i][j] = 1.0
        else:
            E[i][j] = -e

s = [product(w, y, T)] # Initial output vector
p = action(s[0], U, Emax)
y = [p]
i = 0
j = []
p = [0 for j in range(len(s[0]))]

# Iterate until the difference norm is less than Emax
while norm(y[i], p) >= Emax:
    s.append([0 for j in range(len(s[0]))])
    for j in range(len(s[0])):
        s[i + 1][j] = y[i][j] - e * mysum(y[i], j)
    y.append((action(s[i + 1], U, Emax)))
    i += 1
    p = y[i - 1]

print('Output Vectors Table:')
for idx, output_vector in enumerate(y):
    print(f'Iteration {idx + 1}:', *output_vector)
    print('Weights (x, y):')
    for j in range(len(x[0])):
        print(f'x{j}: {w[0][j]:.3f}, y{j}: {w[1][j]:.3f}')
    print('Error:', norm(y[idx], p))
    print()

print('Last Output Vector:', *y[len(y) - 1])

# Determine the class with the highest output value
result_index = y[len(y) - 1].index(max(y[len(y) - 1])) + 1

if max(y[len(y) - 1]) == 0:
    print("The Hamming network cannot make a preference between classes.")
    print("In the case of a small number of input characteristics, the network may not be able to classify the image.")
    plt.show()
    exit()
else:
    q = change(x[result_index - 1], a, b)
    print('The highest positive output value is associated with class', result_index)

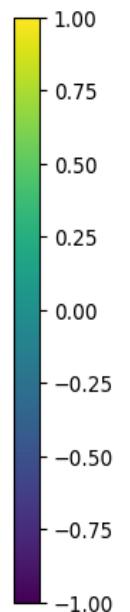
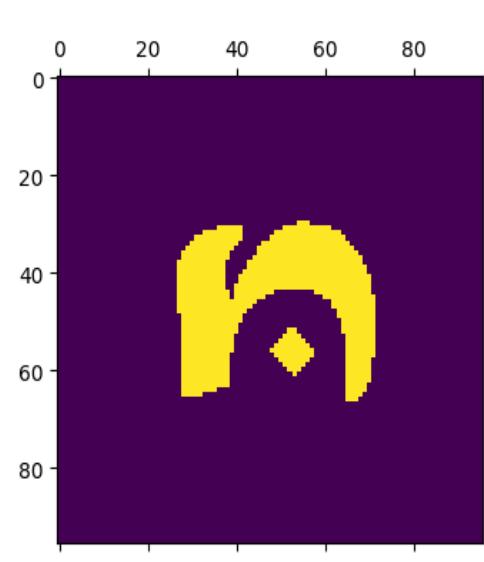
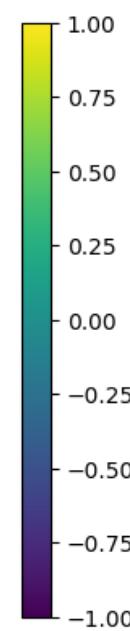
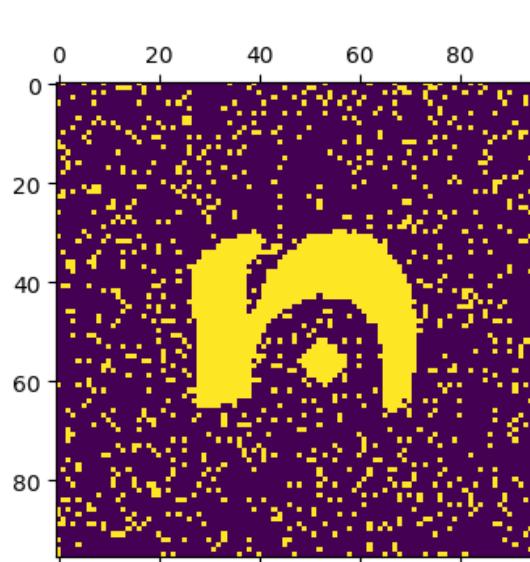
```



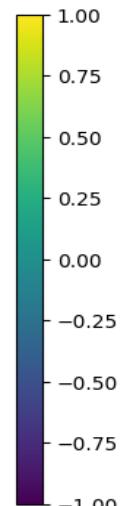
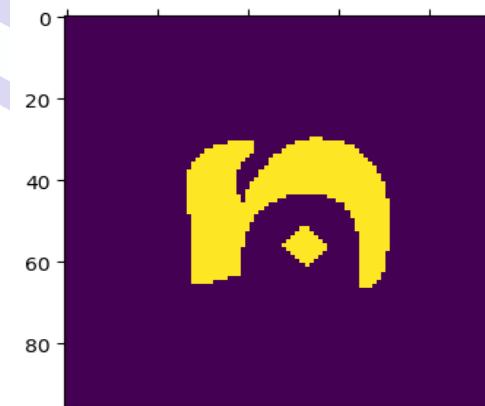
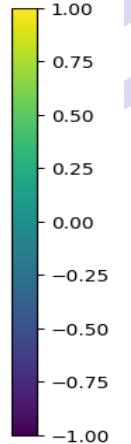
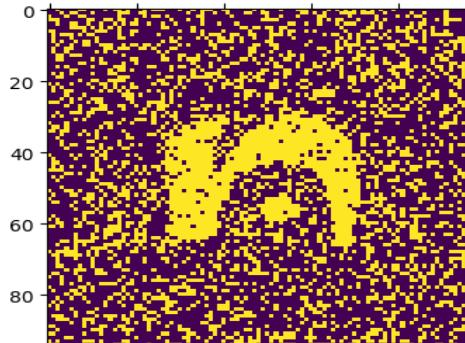
```
plt.matshow(q)
plt.colorbar()
plt.show()
```

```
# Define a factor for introducing noise.
noise_factor = 70
```

نتایج: noise factor=100

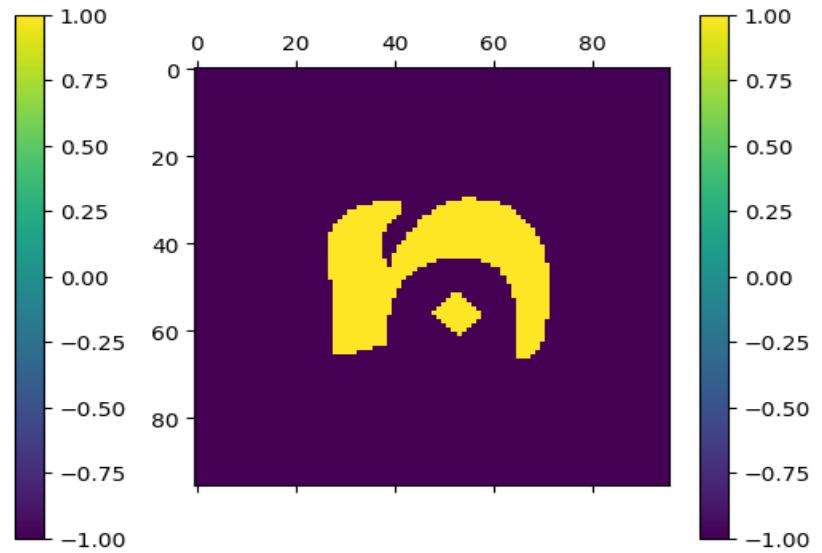
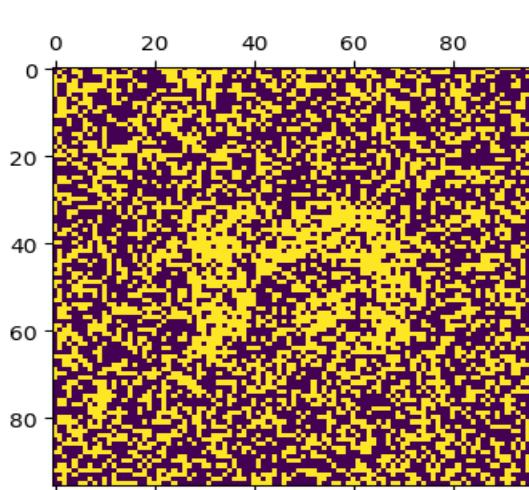


noise factor=200

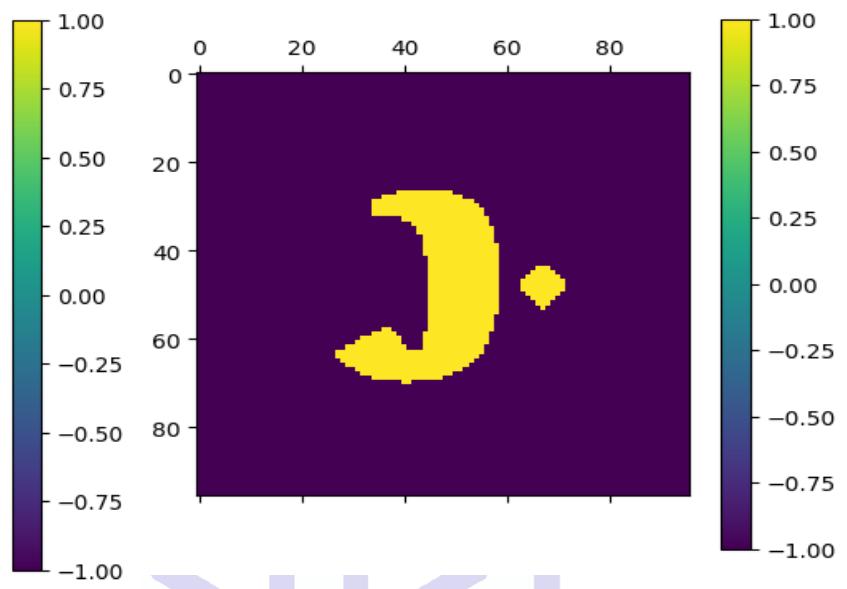
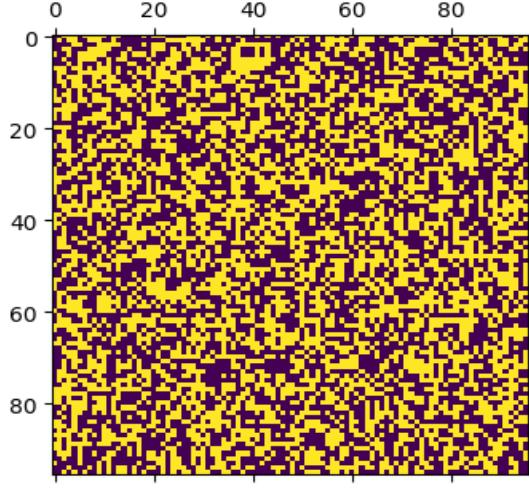


با تغییر ضریب نویز خواهیم داشت:

noise factor=400



noise factor=4500



همانطور که مشخص است با تغییر ضریب نویز به ۴۵۰۰ در تشخیص حرف ج دچار مشکل شدیم، به طور مشابه برای سایر تصاویر نیز این کار را می توان انجام داد.

این معیار می تواند معیاری برای عملکرد این سیستم یاشد و قدرت تشخیص این سیستم را نشان دهد.

### بخش ۳

الهام گرفتن ازتابع نوشته شده برای تولید داده های نویزی، یکتابع بنویسید که از داده های ورودی، خروجی های دارای Point Missing تولید کند. سپس عملکرد شبکه خود را با مقدار مشخصی Point Missing آزمایش و تحلیل کنید. اگر میزان Point Missing از چه حدی بیش تر شود عملکرد شبکه طراحی شده شما دچار اختلال می شود؟ راه حل چیست؟ (راهنمایی: نمونه داده دارای Point Missing در شکل ۳ نشان داده شده است.)

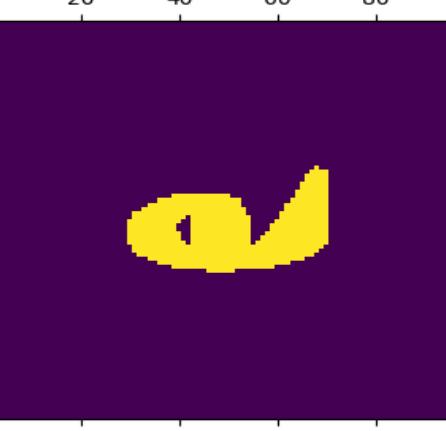
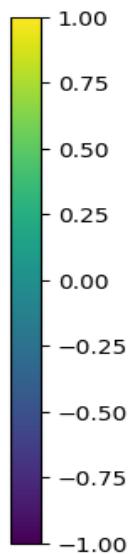
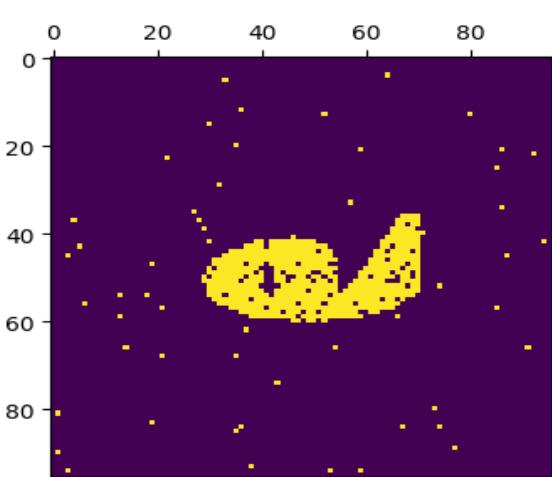
شکل ۳: نمونه داده دارای Missing Point

از همان کد ارتقا یافته ای که در بخش ۱ برای missing point درست کردیم، استفاده می کنیم:

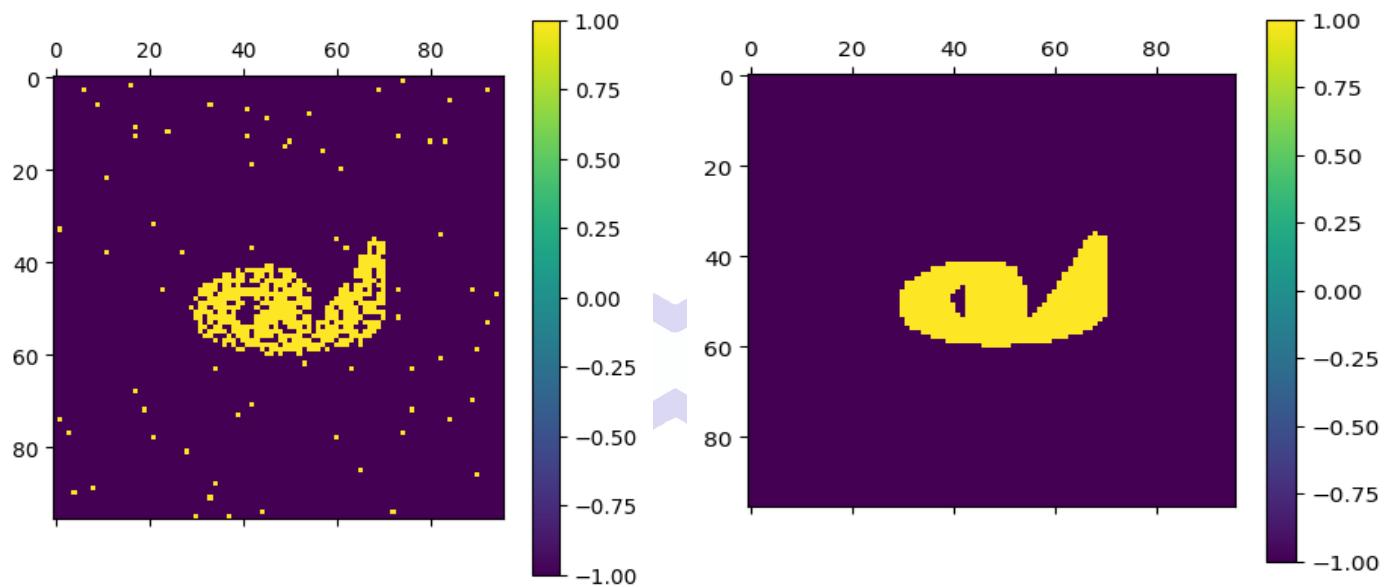
```
for i, image_path in enumerate(image_paths, start=1):
    noisy_image_path = f"/content/noisy{i}.jpg"
    # Specify the number of missing points and conversion percentage
here
    getNoisyBinaryImage(image_path, noisy_image_path,
num_missing_points=500, conversion_percentage=0.1)
    print(f"Noisy image for {image_path} generated and saved as
{noisy_image_path}")
```

نتایج : در این حالت noise factor را برابر ۷۰ قرار می دهیم.

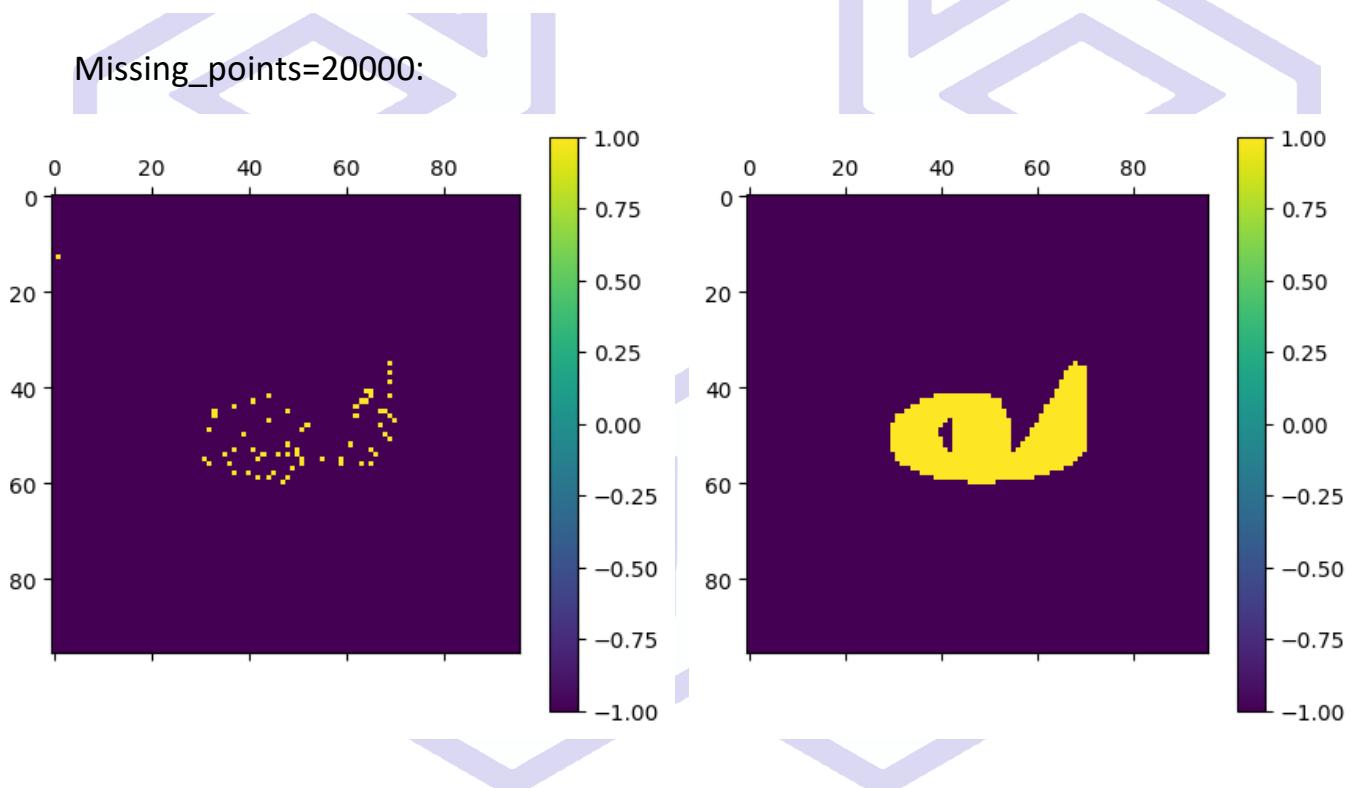
Missing\_points=500:



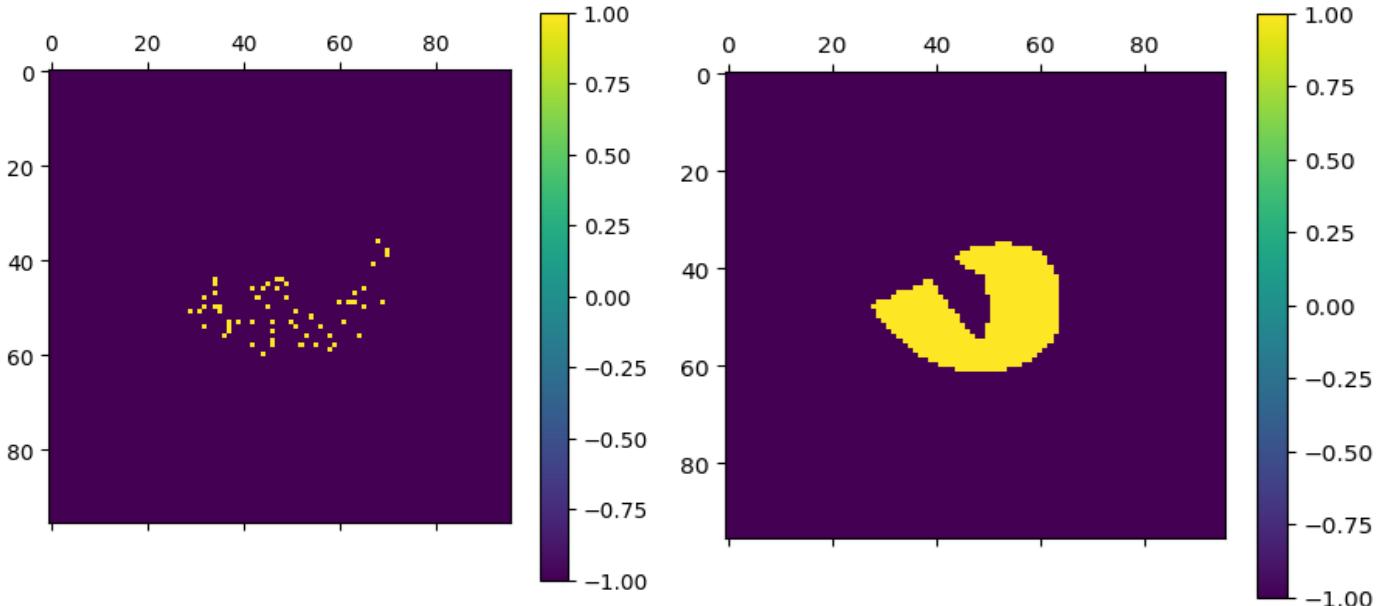
Missing\_points=2000:



Missing\_points=20000:



Missing\_points=21000:



همانطور که مشخص است با تغییر Missing point به ۲۱۰۰۰ در تشخیص حرف و دچار مشکل شدیم، به طور مشابه برای سایر تصاویر نیز این کار را می‌توان انجام داد.

این معیار می‌تواند معیاری برای عملکرد این سیستم یاشد و قدرت تشخیص این سیستم را نشان دهد.

#### راه حل‌های ممکن برای تشخیص Missing Point

- ❖ میزان معقول Missing points
- ❖ تنظیمات دقیق نویز (تنظیمات Noise factor)
- ❖ تصویربرداری دقیق
- ❖ استفاده از روش‌های پیشرفته‌تر

در صورت نیاز، از روش‌های پیشرفته‌تری برای مدیریت نویز و نقاط گم شده استفاده کنید، مثل استفاده از شبکه‌های مولد GANs برای تولید تصاویر نویزدار و گم شده.

## سوال ۴

یک مجموعه داده برای پیش بینی قیمت خانه ها را از طریق این پیوند دانلود کنید و مراحل ذکر شده در سوالات بعدی را برای فایل آن انجام دهید. لازم است که هر قسمت و مورد خواسته شده را با استفاده از دستورات پایتون انجام دهید و در جاهایی که نیاز است، نتایج را به صورت دقیق و کامل نمایش داده و تحلیل کنید.

### بخش ۱

فایل CSV مربوط به این سوال را خوانده و سپس تابع `Pandas.info` را از فراخوانی کنید. تعداد داده هایی که `Nan` هستند را بحسب هر ستون نمایش دهید و اگر نیاز است دستوراتی برای رفع این مشکل بنویسید.

ابتدا کتابخانه های لازم را فراخوانی می کنیم.

فایل CSV را در درایو آپلود کرده و آنرا فراخوانی می کنیم.

اطلاعات این دیتا فریم را به صورت زیر نمایش می دهیم:

```
# Display information about the DataFrame  
df.info()
```

نتایج:

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 4600 entries, 0 to 4599  
Data columns (total 18 columns):  
 #   Column           Non-Null Count  Dtype     
---  --     
 0   date             4600 non-null    object    
 1   price            4600 non-null    float64  
 2   bedrooms         4600 non-null    float64  
 3   bathrooms        4600 non-null    float64  
 4   sqft_living      4600 non-null    int64     
 5   sqft_lot          4600 non-null    int64     
 6   floors            4600 non-null    float64  
 7   waterfront        4600 non-null    int64     
 8   view              4600 non-null    int64     
 9   condition         4600 non-null    int64     
 10  sqft_above        4600 non-null    int64     
 11  sqft_basement     4600 non-null    int64     
 12  yr_built          4600 non-null    int64     
 13  yr_renovated      4600 non-null    int64     
 14  street             4600 non-null    object    
 15  city               4600 non-null    object    
 16  statezip          4600 non-null    object    
 17  country            4600 non-null    object    
 dtypes: float64(4), int64(9), object(5)  
memory usage: 647.0+ KB
```

مشخص است که هیچ کدام از این داده ها `Nan` نیستند، همچنین نوع داده ها نیز مشخص شده است.

تعداد داده های `Nan` را به شکل زیر نیز می توانستیم مشخص کنیم و همچنین این مشکل را رفع کنیم:

```
df.isnull().sum()  
  
# Remove rows with any null values  
# df.dropna(inplace=True)
```

از آنجا که در اینجا هیچ داده `Nan` وجود ندارد پس نیاز به رفع آن هم نیست.

نتایج:

```
date      0  
price     0  
bedrooms  0  
bathrooms 0  
sqft_living 0  
sqft_lot   0  
floors     0  
waterfront 0  
view       0  
condition   0  
sqft_above  0  
sqft_basement 0  
yr_built    0  
yr_renovated 0  
street     0  
city       0  
statezip   0  
country    0  
dtype: int64
```

از آنجا که داده های کشور برای همه ثابت است پس آنها را حذف می کنیم، همچنین داده های `statezip` که مشخصاتی چون کد پستی هستند را نیز حذف می کنیم، در ضمن داده های `street` را نیز حذف می کنیم.

برای اینکار به صورت زیر عمل می کنیم:

```
# Drop the country and street and statezip columns from the DataFrame  
df = df.drop(['statezip', 'country', 'street'], axis=1)
```

## بخش ۲

ماتریس هم بستگی را رسم کنید. چه ویژگی ای با قیمت هم بستگی بیش تری دارد؟

برای محاسبه این ماتریس به صورت زیر عمل می کنیم:

```
# Calculate the correlation between columns and 'price', then sort them in
descending order
correlation_matrix = df.corr()['price'].sort_values(ascending=False)
correlation_matrix
```

نتایج:

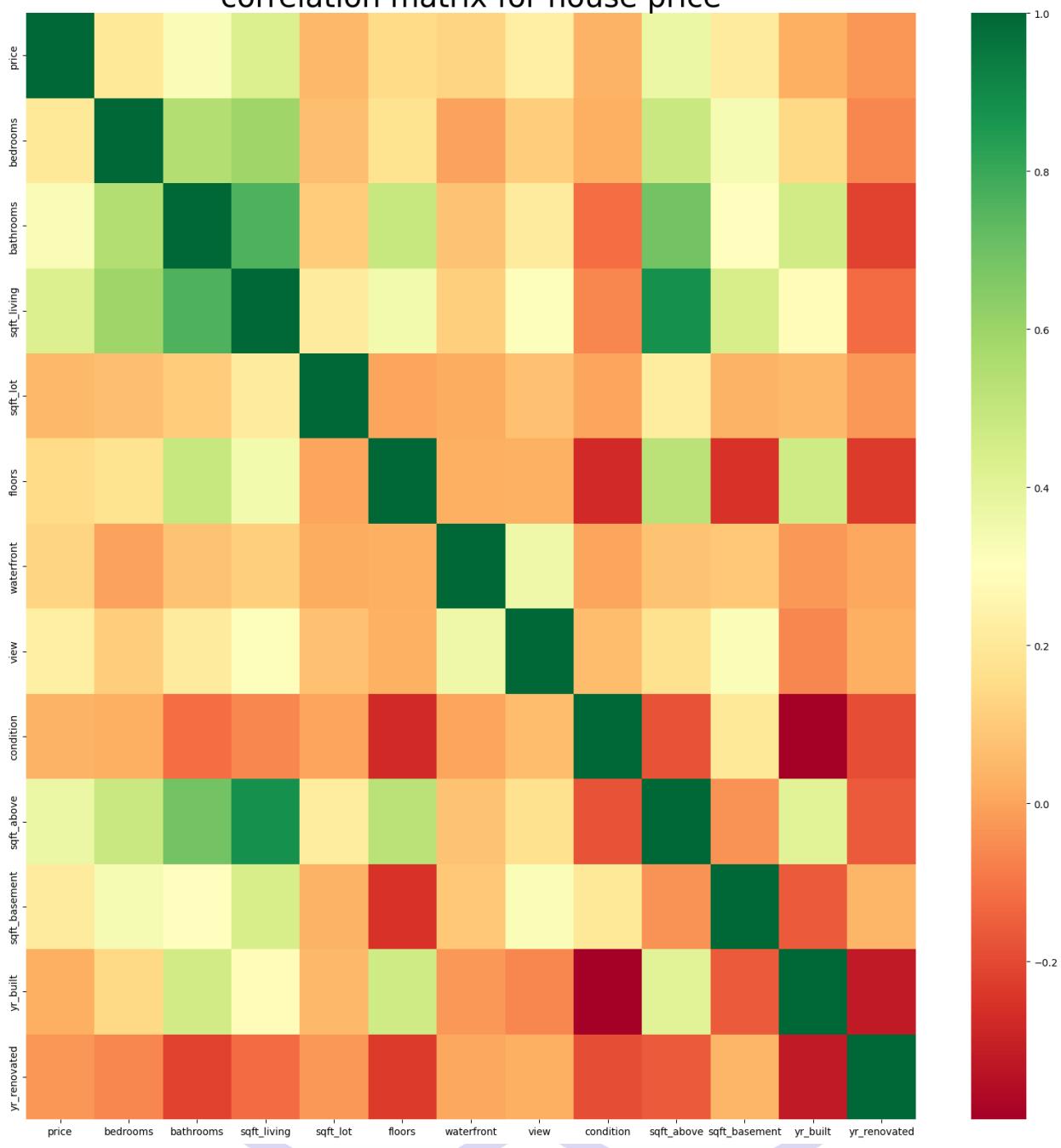
```
price      1.000000
sqft_living   0.430410
sqft_above    0.367570
bathrooms    0.327110
view        0.228504
sqft_basement 0.210427
bedrooms     0.200336
floors       0.151461
waterfront    0.135648
sqft_lot      0.050451
condition     0.034915
yr_built      0.021857
yr_renovated  -0.028774
Name: price, dtype: float64
```

همانطور که مشخص است داده sqft\_living که مربوط به مساحت بنا است بیشترین همبستگی را با قیمت دارد.

برای رسم این ماتریس به صورت زیر عمل می کنیم:

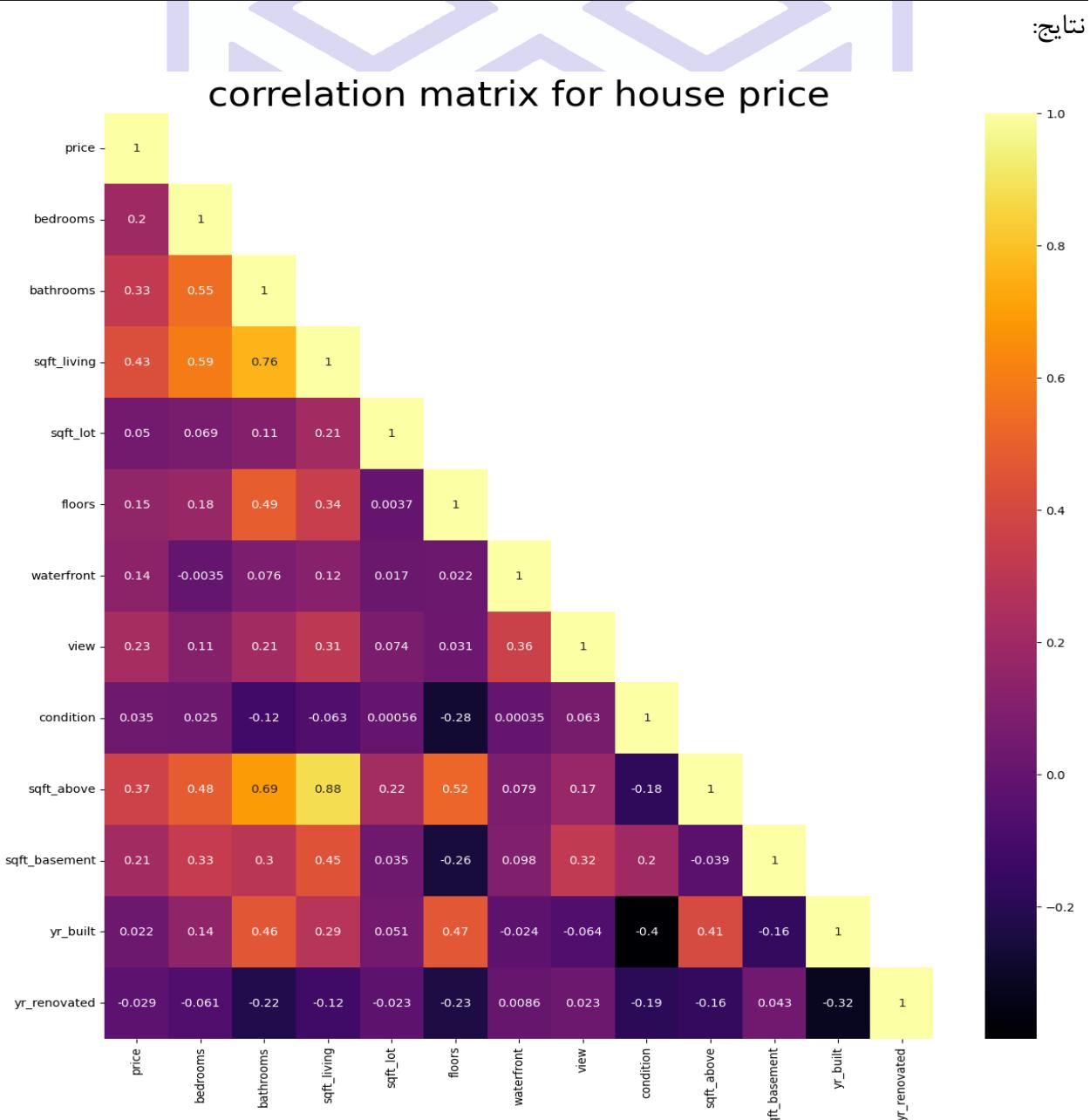
```
# Plot a heatmap to visualize the correlation matrix
plt.figure(figsize=(20, 20))
sns.heatmap(df.corr(), cmap="RdYlGn")
plt.title("correlation matrix for house price", fontsize=30)
plt.show()
```

correlation matrix for house price



همچنین برای نشان دادن عددی این ماتریس به صورت زیر عمل می کنیم:

```
# Select columns with numerical data types
num = df.select_dtypes(exclude=['object']).columns
num
# Create a heatmap to visualize the correlation matrix of numerical
columns
plt.figure(figsize=(15, 15))
sns.heatmap(df[num].corr(), annot=True, cmap='inferno',
mask=np.triu(df[num].corr(), k=1))
plt.title("correlation matrix for house price", fontsize=30)
```



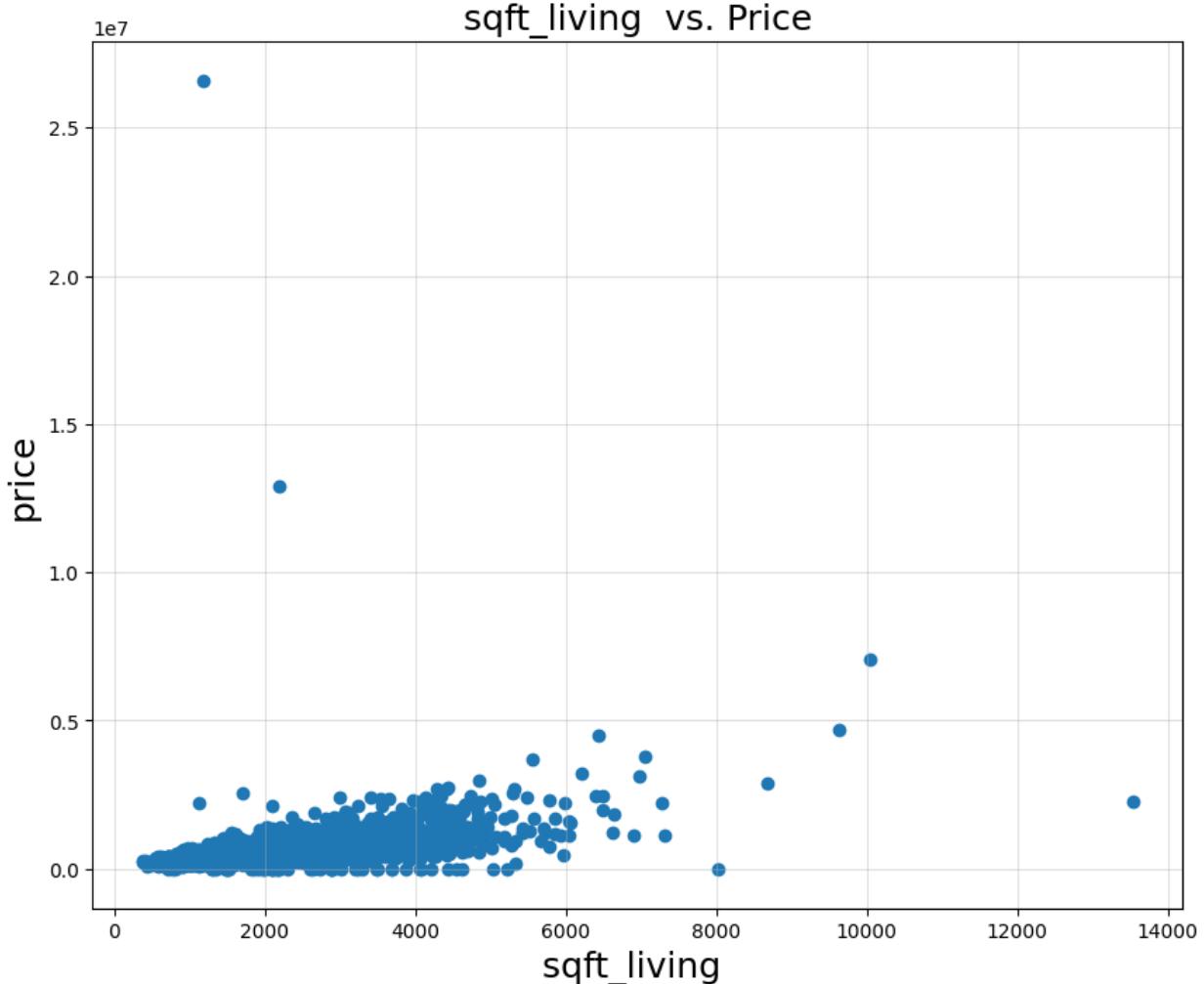
### بخش ۳

نمودار توزیع قیمت و نمودار قیمت و ویژگی ای که هم بستگی زیادی با قیمت دارد را رسم کنید.

برای رسم نمودار قیمت و `sqft_living` به صورت زیر عمل می کنیم:

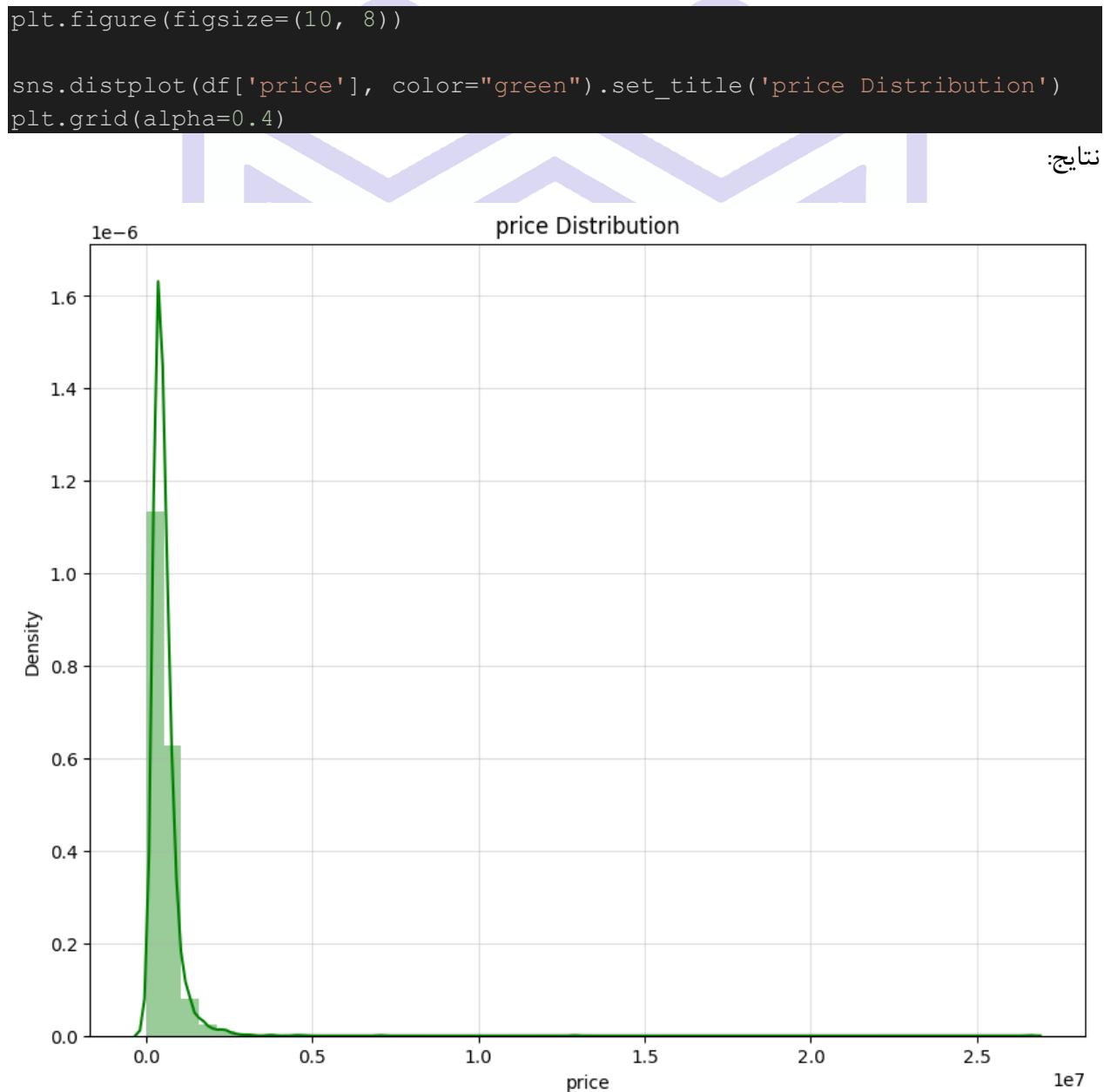
```
# Create a scatter plot of  against price
plt.figure(figsize=(10, 8))
plt.scatter(x='sqft_living', y='price', data=df)
plt.xlabel('sqft_living ', fontsize=18)
plt.ylabel('price', fontsize=18)
plt.title('sqft_living  vs. Price', fontsize=18)
plt.grid(alpha=0.4)
plt.show()
```

نتایج:



با توجه به اینکه این داده بیشترین همبستگی را با قیمت دارد، پس از بین همه داده ها نمودارش بیشترین شباهت را به خط دارد ولی چون میزان این همبستگی کمتر از ۵۰ درصد است پس نمودار آن خیلی هم به خط نزدیک نیست.

برای رسم توزیع داده های قیمت نیز به صورت زیر عمل می کنیم:



تا حدی می توان گفت داده های قیمت دارای توزیع نمایی هستند.

## بخش ۴

ستون Date را به دو ستون ماه و سال تبدیل کنید و این ستون را از دیتافریم حذف کنید.

برای تبدیل این داده ها به دو ستون و همچنین حذف ستون Date به صورت زیر عمل می کنیم:

```
# Extract 'year', 'month', and 'day' from the 'date' column
df['year'] = pd.to_datetime(df['date']).dt.year
df['month'] = pd.to_datetime(df['date']).dt.month

# Show the DataFrame with the separate 'year' and 'month' columns
df = df[['year', 'month']] + [col for col in df.columns if col not in ['year', 'month']]

# Drop the specified columns from the DataFrame
df = df.drop(['date'], axis=1)
df
```

نتایج:

	year	month	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	city
0	2014	5	3.130000e+05	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	Shoreline
1	2014	5	2.384000e+06	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	Seattle
2	2014	5	3.420000e+05	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	Kent
3	2014	5	4.200000e+05	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	Bellevue
4	2014	5	5.500000e+05	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	Redmond
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4595	2014	7	3.081667e+05	3.0	1.75	1510	6360	1.0	0	0	4	1510	0	1954	1979	Seattle
4596	2014	7	5.343333e+05	3.0	2.50	1460	7573	2.0	0	0	3	1460	0	1983	2009	Bellevue
4597	2014	7	4.169042e+05	3.0	2.50	3010	7014	2.0	0	0	3	3010	0	2009	0	Renton
4598	2014	7	2.034000e+05	4.0	2.00	2090	6630	1.0	0	0	3	1070	1020	1974	0	Seattle
4599	2014	7	2.206000e+05	3.0	2.50	1490	8102	2.0	0	0	4	1490	0	1990	0	Covington

همچنین از آنجا که داده سال برای همه داده های یکسان است می توانیم آنرا حذف کنیم:

```
# Drop year columns from the DataFrame
df = df.drop(['year'], axis=1)
df
```

نتایج:

	month	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	city
0	5	3.130000e+05	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	Shoreline
1	5	2.384000e+06	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	Seattle
2	5	3.420000e+05	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	Kent
3	5	4.200000e+05	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	Bellevue
4	5	5.500000e+05	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	Redmond
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4595	7	3.081667e+05	3.0	1.75	1510	6360	1.0	0	0	4	1510	0	1954	1979	Seattle
4596	7	5.343333e+05	3.0	2.50	1460	7573	2.0	0	0	3	1460	0	1983	2009	Bellevue
4597	7	4.169042e+05	3.0	2.50	3010	7014	2.0	0	0	3	3010	0	2009	0	Renton
4598	7	2.034000e+05	4.0	2.00	2090	6630	1.0	0	0	3	1070	1020	1974	0	Seattle
4599	7	2.206000e+05	3.0	2.50	1490	8102	2.0	0	0	4	1490	0	1990	0	Covington

همچنین داده های مربوط به شهر را نیز که داده های عددی نیستند را نیز باید به عدد تبدیل کنیم.

برای اینکار به صورت زیر عمل می کنیم:

```
# List of specified categorical columns
dummy = ['city']

# Convert categorical columns to numerical using one-hot encoding
df2 = pd.get_dummies(df, columns=dummy, drop_first=True)

# Display the first few rows of the modified DataFrame
df2.head()
```

## بخش ۵

داده ها را با نسبت ۸۰ به ۲۰ درصد به مجموعه های آموزش و آزمون تقسیم کنید و داده های آموزشی و آزمون را با استفاده از `MinMaxScaler` مقیاس کنید.

ابتدا داده های مربوط به قیمت را خروجی در نظر گرفته و بقیه داده ها را ویژگی در نظر می گیریم، سپس داده ها را به نسبت ۸۰ به ۲۰ تقسیم کرده، همچنین داده ها را `shuffle` می کنیم و برای خاصیت تکرار پذیری از استفاده `random_state` می کنیم:

```
# Separate features (X) and output (Y) data
X = df2.drop(["price"], axis=1) # features
y = df2["price"] # Output data

# Splitting the dataset into the Training set and Test set (80/20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=93, shuffle=True)

# Display the dimensions of the training and testing sets
print(f'Dimensions of the training features: {X_train.shape}')
print(f'Dimensions of the training target: {y_train.shape}')
print(f'Dimensions of the testing features: {X_test.shape}')
print(f'Dimensions of the testing target: {y_test.shape}')
```

نتایج:

```
Dimensions of the training features: (3680, 56)
Dimensions of the training target: (3680,)
Dimensions of the testing features: (920, 56)
Dimensions of the testing target: (920,)
```

که همان نتایج مورد انتظار ماست.

برای `scale` کردن داده ها به روش `Min_Max Scaler` نیز از روش `scale` که در کلاس حل تمرین گفته شد استفاده می کنیم، اما باید دقت داشته باشیم که داده های تست و آموزش را جدا باید `scale` کنیم، چون اگر این داده ها را با هم `scale` کنیم، در واقع به سیستم تقلب رسانده ایم و اطلاعاتی راجع به داده های تست به آن داده ایم. بنابراین نباید هیچ اطلاعاتی راجع به داده های تست به سیستم بدهیم. این مورد را برای خروجی نیز انجام می دهیم.

بنابراین به صورت زیر عمل می کنیم:

```
# Initialize Min-Max Scaler
scaler_1 = MinMaxScaler()

# Normalize the training input data
X_train = scaler_1.fit_transform(X_train)

# Normalize the test input data
X_test = scaler_1.transform(X_test)
```

```
# Convert y_train and y_test type to DataFrame
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)

scaler_2 = MinMaxScaler()

# Normalize outputs
y_train = scaler_2.fit_transform(y_train)
y_test = scaler_2.transform(y_test)
```



## بخش ۶

یک مدل Multi-Layer Perceptron (MLP) ساده با ۲ لایه پنهان یا بیش تر بسازید. بخشی از داده های آموزش را برای اعتبارسنجی کنار بگذارید و با انتخاب بهینه ساز وتابع اتلاف مناسب، مدل را آموزش دهید. نمودارهای اتلاف و R2 Score مربوط به آموزش و اعتبارسنجی را رسم و نتیجه را تحلیل کنید.

از MLP با ۳ لایه پنهان استفاده می کنیم، هر سه لایه دارای ۱۰ نورون و دارای تابع فعال ساز `relu` می باشند، همچنین لایه آخر که همان لایه خروجی است دارای یک نورون و تابع فعال ساز `linear` است، بنابراین مشابه موارد گفته شده در کلاس حل تمرین به صورت زیر عمل می کنیم:

```
model_3 = Sequential()

# Add the first hidden layer with 10 neurons and ReLU activation function
model_3.add(Dense(10, activation='relu', input_shape=(X_train.shape[1],)))

# Add the second hidden layer with 10 neurons and ReLU activation function
model_3.add(Dense(10, activation='relu'))

# Add the third hidden layer with 10 neurons and ReLU activation function
model_3.add(Dense(10, activation='relu'))

# Add an output layer with 1 neuron and linear activation function
model_3.add(Dense(1, activation='linear'))

model_3.summary()
```

نتایج:

Layer (type)	Output Shape	Param #
dense_139 (Dense)	(None, 10)	570
dense_140 (Dense)	(None, 10)	110
dense_141 (Dense)	(None, 10)	110
dense_142 (Dense)	(None, 1)	11
<hr/>		
Total params: 801 (3.13 KB)		
Trainable params: 801 (3.13 KB)		
Non-trainable params: 0 (0.00 Byte)		

حال باید مدل را بر داده ها فیت کنیم، برای اینکار از انتخاب بهینه ساز adam که راجع به آن گروه درس صحبت شد و تابع اتلاف mse استفاده می کنیم.

همچنین برای داده های اعتبار سنجی نیز داده های train را به دو دسته تقسیم می کنیم و ۲۰ درصد داده ها را برای validation استفاده می کنیم. تعداد ایپاک ها را صد در نظر می گیریم.

```
model_3.compile(optimizer='adam', loss='mse')

# Split the data into training and validation sets
X_train1, X_val, y_train1, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=93, shuffle=True)

history = model_3.fit(X_train1, y_train1, validation_data=(X_val, y_val),
epochs=100, batch_size=10, verbose=0)
```

```
# Evaluate the model
loss = model_3.evaluate(X_test, y_test)
```

نتایج خط:

```
29/29 [=====] - 0s 1ms/step - loss: 9.0906e-05
```

برای بررسی دقیق نیز از معیار R2score استفاده می کنیم.

```
# Predictions on training and validation data
y_pred_3_train = model_3.predict(X_train1)
y_pred_3_val = model_3.predict(X_val)

# Calculate R2 score for training and validation data
rscore_train = r2_score(y_train1, y_pred_3_train)
rscore_val = r2_score(y_val, y_pred_3_val)

# Calculate R2 score for testing data
y_pred_3_test = model_3.predict(X_test)
rscore_3 = r2_score(y_test, y_pred_3_test)

print(f"Test R2score: {rscore_3}")
print(f"Train R2score: {rscore_train}")
print(f"Validation R2score: {rscore_val}")
```

نتایج:

```
92/92 [=====] - 0s 1ms/step  
23/23 [=====] - 0s 1ms/step  
29/29 [=====] - 0s 1ms/step  
Test R2score: 0.6380771480187429  
Train R2score: 0.35001078758643345  
Validation R2score: 0.4116297511019831
```

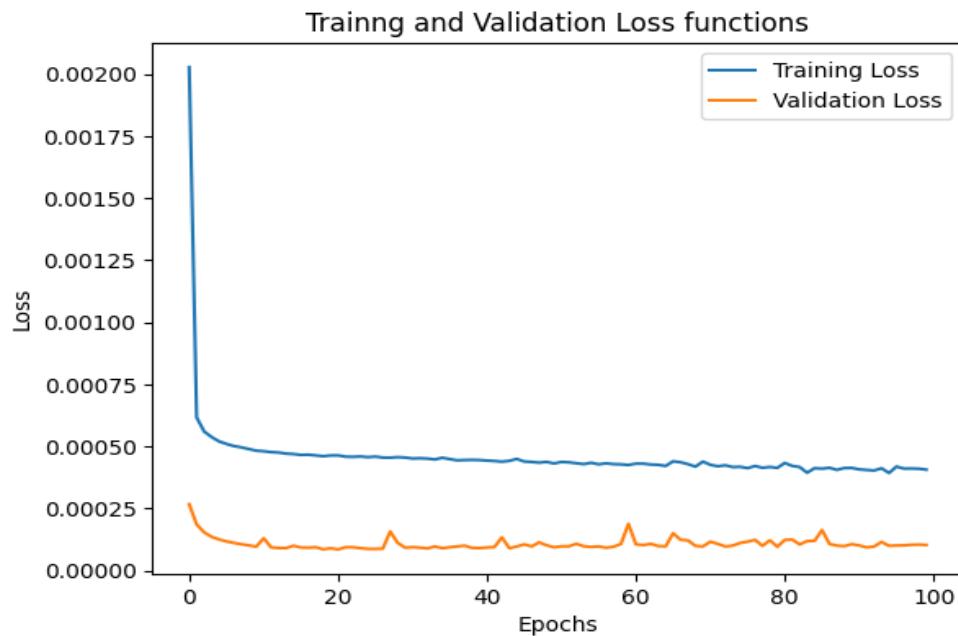
همانطور که مشخص است دقت چندان خوب نیست.

این بهترین دقتی بود که با تغییر پارامترها به آن دست یافتیم.

برای رسم تابع اتلاف روی داده های آموزش و اعتبار سنجی مشابه روشی که در کلاس حل تمرین گفته شد عمل می کنیم:

```
# Plot the training and validation loss  
plt.plot(history.history['loss'], label='train')    # Training loss  
plt.plot(history.history['val_loss'], label='val')   # Validation loss  
  
plt.legend(['Training Loss', 'Validation Loss'])  
plt.xlabel("Epochs")  
plt.ylabel("Loss")  
plt.title("Trainng and Validation Loss functions")  
plt.show()
```

نتایج:



## بخش ۷

فرآیند سوال قبل را با یک بهینه ساز و تابع اتلاف جدید انجام داده و نتایج را مقایسه و تحلیل کنید.

در این بخش از بهینه ساز گرادیان نزولی و همچنین تابع اتلاف (mean absolute error) mae استفاده می کنیم و بخش قبل را دوباره تکرار می کنیم:

```
# Compile model with stochastic gradient descent optimizer and mean
absolute error loss
model_3.compile(optimizer = 'sgd', loss = 'mae')

# Split the data into training and validation sets
X_train1, X_val, y_train1, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=93, shuffle=True)

history = model_3.fit(X_train1, y_train1, validation_data=(X_val, y_val),
epochs=100, batch_size=10, verbose=0)

# Evaluate the model
loss = model_3.evaluate(X_test , y_test)
```

نتایج:

```
29/29 [=====] - 0s 1ms/step - loss: 0.0044
```

```
# Predictions on training and validation data
y_pred_3_train = model_3.predict(X_train1)
y_pred_3_val = model_3.predict(X_val)

# Calculate R2 score for training and validation data
rscore_train = r2_score(y_train1, y_pred_3_train)
rscore_val = r2_score(y_val, y_pred_3_val)

# Calculate R2 score for testing data
y_pred_3_test = model_3.predict(X_test)
rscore_3 = r2_score(y_test , y_pred_3_test)

print(f"Test R2score: {rscore_3}")
print(f"Train R2score: {rscore_train}")
print(f"Validation R2score: {rscore_val}")
```

```

87/87 [=====] - 0s 1ms/step
29/29 [=====] - 0s 1ms/step
29/29 [=====] - 0s 1ms/step
Test R2score: 0.7025022605110858
Train R2score: 0.29695997451786105
Validation R2score: 0.34460826808219724

```

مشخص است که مدل فیت شده بر داده ها نسبت به حالت قبل دقیق پایین تری دارد، اما دقیق آن بر روی داده های تست بهتر شده است.

برای رسمتابع اتلاف روی داده های آموزش و اعتبار سنجی مشابه روشی که در کلاس حل تمرین گفته شد عمل می کنیم:

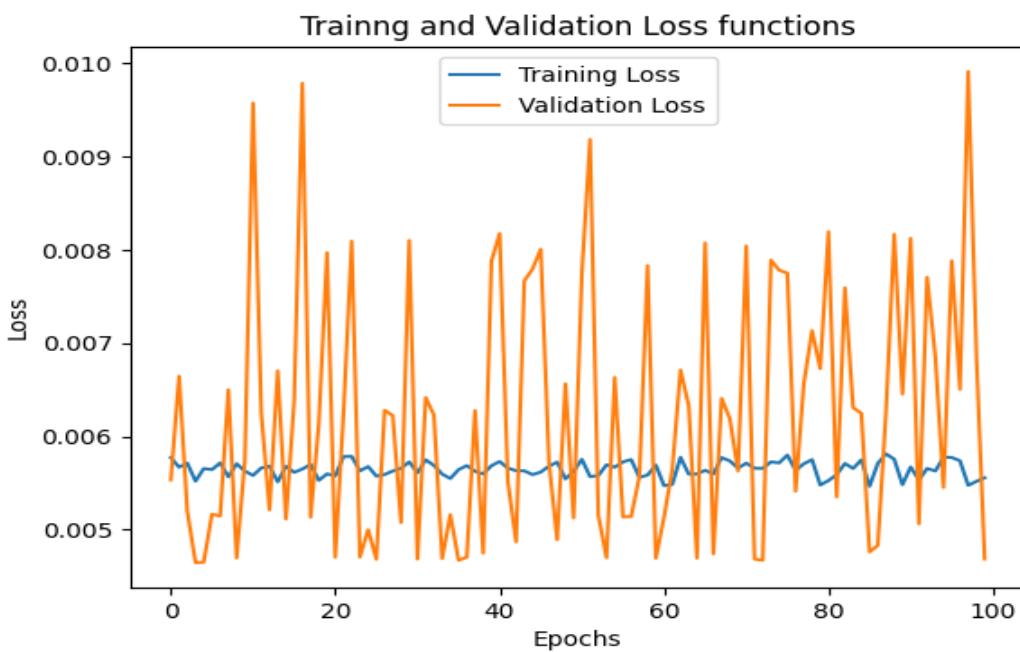
```

# Plot the training and validation loss
plt.plot(history.history['loss'], label='train')      # Training loss
plt.plot(history.history['val_loss'], label='val')    # Validation loss

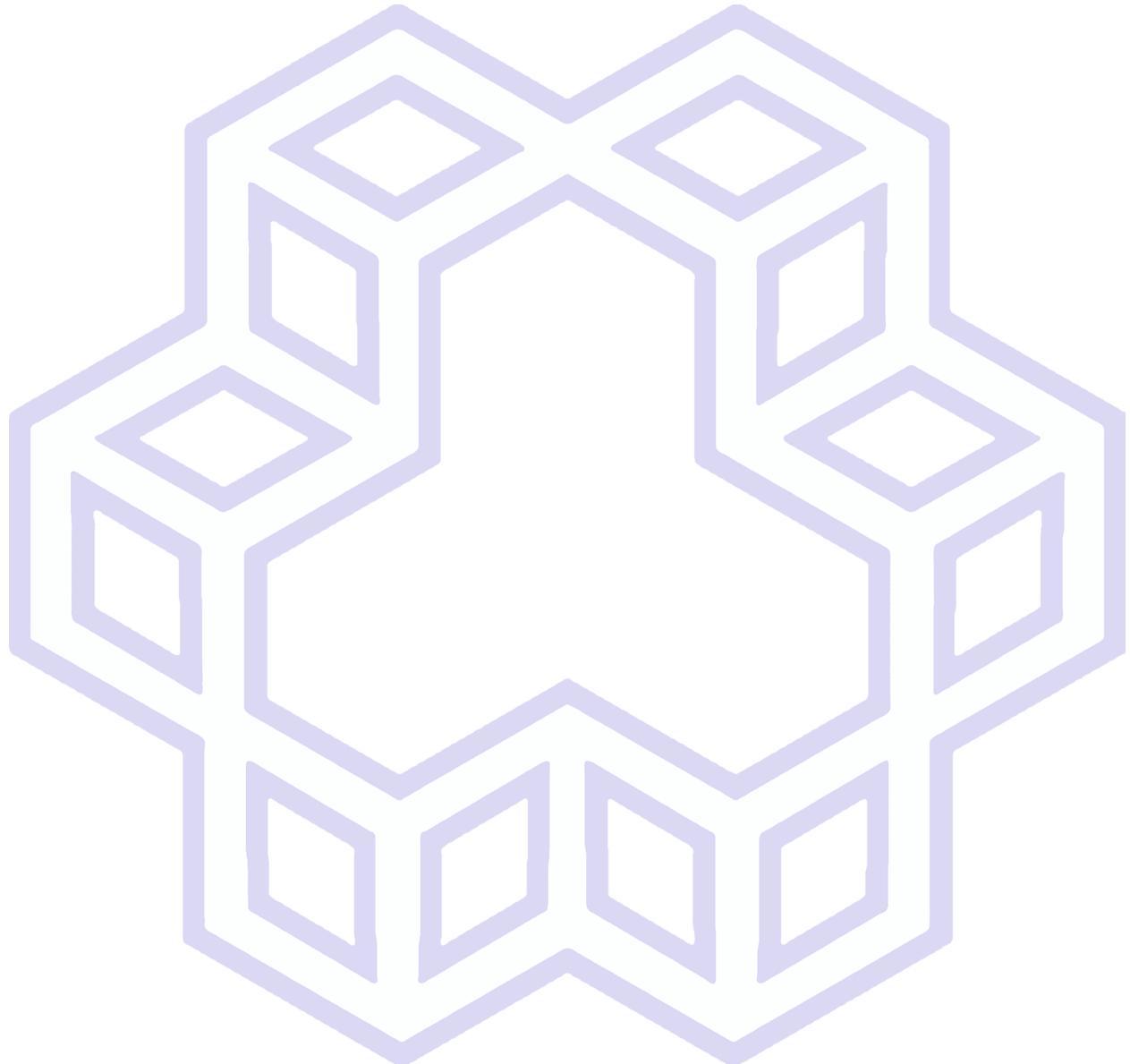
plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss functions")
plt.show()

```

نتایج:



همانطور که مشخص است در این حالت باز هم نتوانستیم برای داده های validation به همگرایی برسیم، برای رسیدن به این همگرایی می توان با تغییر تعداد ایپاک ها و همچنین تغییر نرخ ارزیابی به همگرایی رسید.



## بخش ۸

پنج داده را به صورت تصادفی از مجموعه ارزیابی انتخاب کرده و قیمت پیش بینی شده را به همراه قیمت واقعی نشان دهید. قیمت پیش بینی شده با قیمت واقعی چقدر تفاوت دارد؟ آیا این عملکرد مناسب است؟ برای بهبود آن چه پیشنهادی دارید؟

برای اینکار ابتدا داده های scale شده را unscale می کنیم و سپس مطابق روشی که در کلاس حل تمرین گفته شد عمل می کنیم:

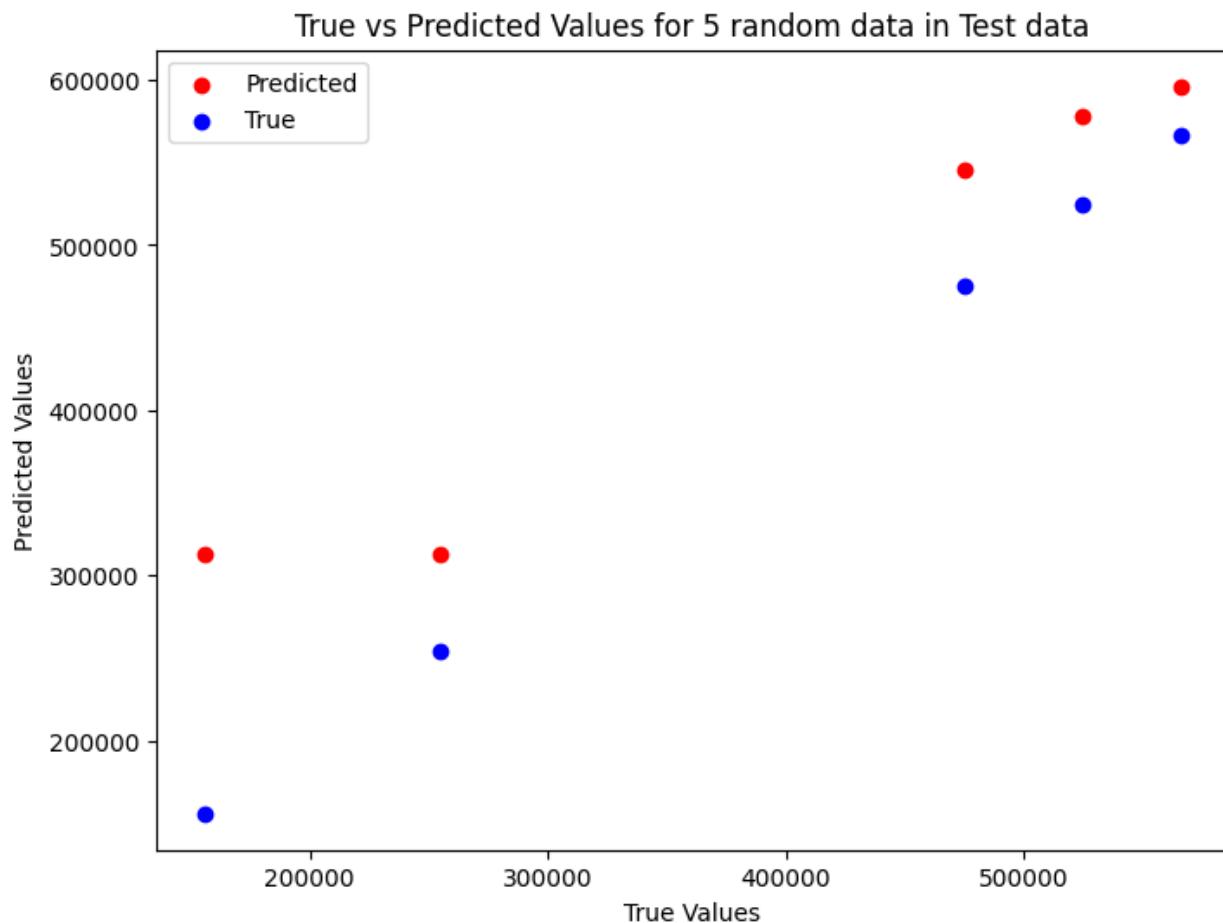
```
# Inverse transform the scaled test data and predictions
y_test_unscaled = scaler_2.inverse_transform(y_test)
y_pred_unscaled = scaler_2.inverse_transform(y_pred_3_test)

random_pred = []
random_test = []

for i in range(5):
    j = random.randint(0, len(y_test_unscaled))
    random_pred.append(y_pred_unscaled[i])
    random_test.append(y_test_unscaled[i])

# Plotting the unscaled true test data against predictions with different
colors
plt.figure(figsize=(8, 6))
plt.scatter(random_test, random_pred, color='red', label='Predicted')
plt.scatter(random_test, random_test, color='blue', label='True')
plt.title('True vs Predicted Values for 5 random data in Test data')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()
```

نتایج:



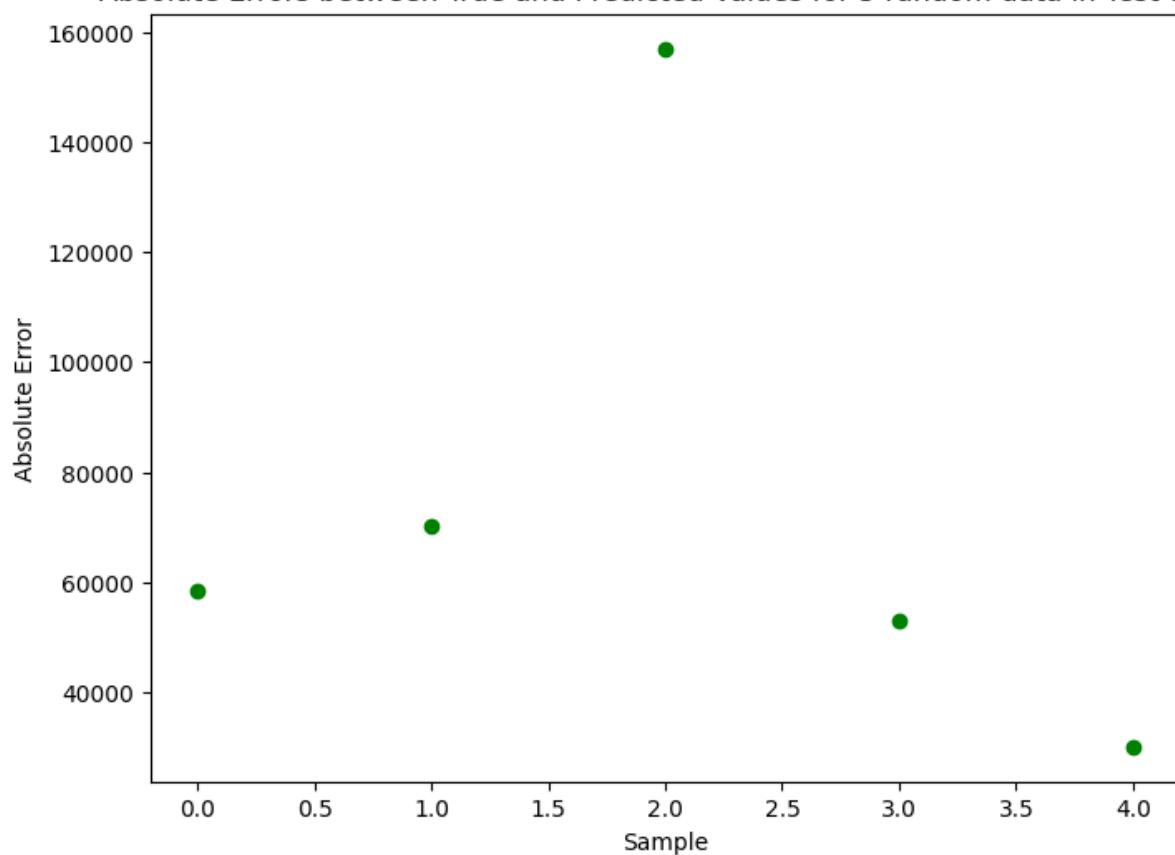
همینطور برای نشان دادن خطای این ۴ داده می توان به صورت زیر عمل کرد:

```
# Assuming random_test and random_pred are lists
random_test = np.array(random_test)
random_pred = np.array(random_pred)

# Calculate errors between true and predicted values
errors = np.abs(random_test - random_pred)

# Plotting the errors
plt.figure(figsize=(8, 6))
plt.plot(errors, marker='o', linestyle='', color='green')
plt.title('Absolute Errors between True and Predicted Values for 5 random data in Test data')
plt.xlabel('Sample')
plt.ylabel('Absolute Error')
plt.show()
```

Absolute Errors between True and Predicted Values for 5 random data in Test data



داده های واقعی و داده های تخمین زده شده:

```
print("True values:")
print(random_test)
print("\nPredicted values:")
print(random_pred)
```

نتایج:

```
True values:
[[254600.]
 [475000.]
 [156000.]
 [525000.]
 [566000.]]
```

```
Predicted values:
[[313017.22]
 [545077.56]
 [313017.22]
 [577813.25]
 [596068.56]]
```

مشخص است که عملکرد این مدل ها آنچنان هم خوب نیست.

برای بهبود عملکرد می توان به موارد زیر اشاره کرد:

❖ تنظیم پارامترها: بهینه سازی پارامترهای مدل و شبکه عصبی می تواند بهبود عملکرد را به همراه داشته باشد. از ابزارهایی مانند جستجوی فرا پارامترها، شبکه های عصبی با پیچیدگی مناسب، و تنظیم مناسب تعداد لایه ها و نورون ها بهره ببرید.

❖ استفاده از لایه های با پیچیدگی بالا: افزایش پیچیدگی مدل می تواند بهبود عملکرد آن را داشته باشد، اما باید از اورفیتینگ overfitting جلوگیری کرد.

❖ تعویض تابع اتلاف و بهینه ساز: استفاده از توابع اتلاف و روش های بهینه سازی مختلف و انتخاب یک تابع مناسب با توجه به مسئله ممکن است تأثیر بسزایی داشته باشد.

❖ کاهش ابعاد ویژگی ها: اگر تعداد ویژگی ها زیاد است، امکان دارد با کاهش ابعاد dimensionality feature selection یا حتی انتخاب ویژگی reduction دقیق مدل افزایش یابد.

همانطور که مشخص است داده های مربوط به street را حذف کردیم که می توان با نگهداری آن و تبدیل آن به داده عددی به دقت بهتری دست یافت. روش های تغییر تابع اتلاف و روش بهینه سازی و تغییر فرآپارتمترها و استفاده از داده های scale شده بر روی داده ها امتحان شد اما دقت بهتر نشد.

از روش های دیگر می توان به تغییر روش مثلا استفاده از RBF و یا سایر روش ها اشاره کرد.

## سوال ۵

### بخش ۱

مجموعه داده Iris را فراخوانی کنید و روش های تحلیل داده ای که آموخته اید را روی آن به کار بینیدید. داده ها را با نسبتی دلخواه و مناسب به مجموعه های آموزش و ارزیابی تقسیم کنید.

ابتدا کتابخانه های لازم را فراخوانی می کنیم:

ابتدا داده ها را قرایخوانی کرده و سپس خروجی و ویژگی ها را از هم جدا می کنیم:

```
X, y = load_iris(return_X_y=True)

# Display the dimensions of the dataset
print(f'Dimensions of the features: {X.shape}')
print(f'Dimensions of the target: {y.shape}')
```

نتایج:

```
Dimensions of the features: (150, 4)
Dimensions of the target: (150,)
```

همانطور که مشخص است این مجموعه داده دارای ۱۵۰ نمونه داده و ۴ ویژگی و تنها یک هدف (خروجی) است، برای مشخص کردن اطلاعات داده ها به صورت زیر عمل می کنیم:

```
iris = load_iris()
data = iris.data
target = iris.target
feature_names = iris.feature_names

df = pd.DataFrame(data, columns=feature_names)
df['target'] = target

df.head()
```

نتایج:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

برای مشاهده اطلاعات داده ها به صورت زیر عمل می کنیم:

```
# Display information about the DataFrame  
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 150 entries, 0 to 149  
Data columns (total 5 columns):  
 #   Column           Non-Null Count  Dtype     
---  --    
 0   sepal length (cm)    150 non-null   float64  
 1   sepal width (cm)     150 non-null   float64  
 2   petal length (cm)    150 non-null   float64  
 3   petal width (cm)     150 non-null   float64  
 4   target              150 non-null   int64  
 dtypes: float64(4), int64(1)  
 memory usage: 6.0 KB
```

همانطور که مشخص است هیچ کدام از داده ها Nan نیستند.

بررسی تعداد داده های هر کلاس:

```
# Count the number of samples in each class  
unique, counts = np.unique(y, return_counts=True)  
print("Class counts before balancing:", dict(zip(unique, counts)))  
  
# # Use RandomUnderSampler to balance the classes  
# rus = RandomUnderSampler(random_state=42)  
# X_resampled, y_resampled = rus.fit_resample(X, y)  
  
# # Count the number of samples in each class after balancing  
# unique_resampled, counts_resampled = np.unique(y_resampled,  
# return_counts=True)
```

```
# print("Class counts after balancing:", dict(zip(unique_resampled,
counts_resampled)))
# print('\n')
# print("New balanced dataset shape:", X_resampled.shape,
y_resampled.shape)
```

نتایج:

```
Class counts before balancing: {0: 50, 1: 50, 2: 50}
```

همانطور که مشخص است این مجموعه داده دارای ۳ کلاس است و تعداد داده های هر کلاس نیز ۵۰ می باشد پس داده ها دارای عدم تعادل نیستند و نیاز به انجام کاری نیست.(در صورتی که داده ها دارای عدم تعادل بودند به کمک under sampling می توانستیم این مشکل را حل کنیم).

: correlation Matrix بررسی

همانند سوال ۴ عمل می کنیم:

```
# Calculate the correlation between columns and 'price', then sort them in
descending order
correlation_matrix = df.corr()['target'].sort_values(ascending=False)
correlation_matrix
```

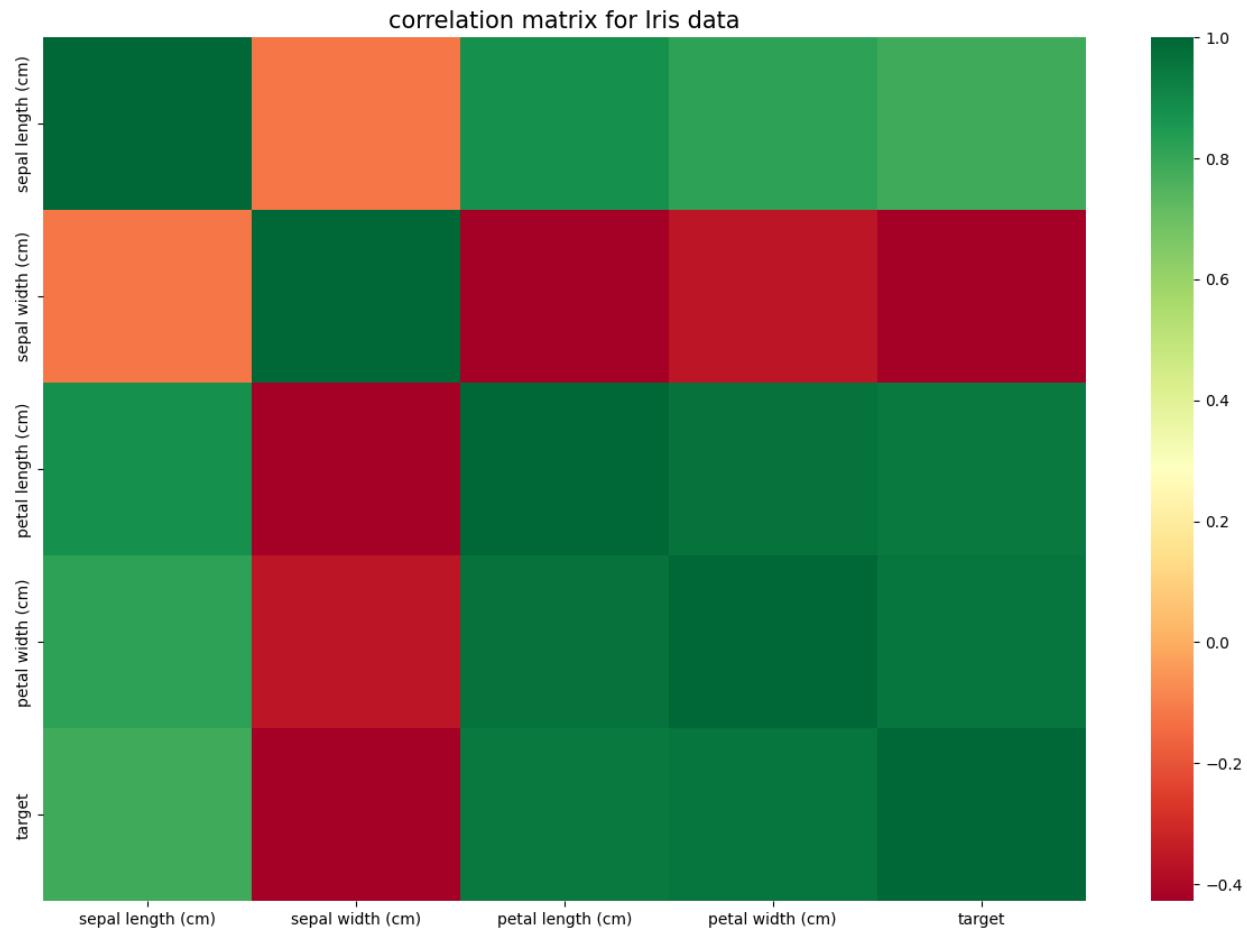
نتایج:

target	1.000000
petal width (cm)	0.956547
petal length (cm)	0.949035
sepal length (cm)	0.782561
sepal width (cm)	-0.426658
Name: target, dtype: float64	

همانطور که مشخص است ویژگی petal length و petal width به ترتیب بیشترین همبستگی را با داده های هدف دارند.(البته از آنجا که داده ها مربوط به کلاس بندی هستند بررسی correlation Matrix چندان با معنا نیست).

```
# Plot a heatmap to visualize the correlation matrix
plt.figure(figsize=(15, 10))
sns.heatmap(df.corr(), cmap="RdYlGn")
plt.title("correlation matrix for Iris data", fontsize=15)
plt.show()
```

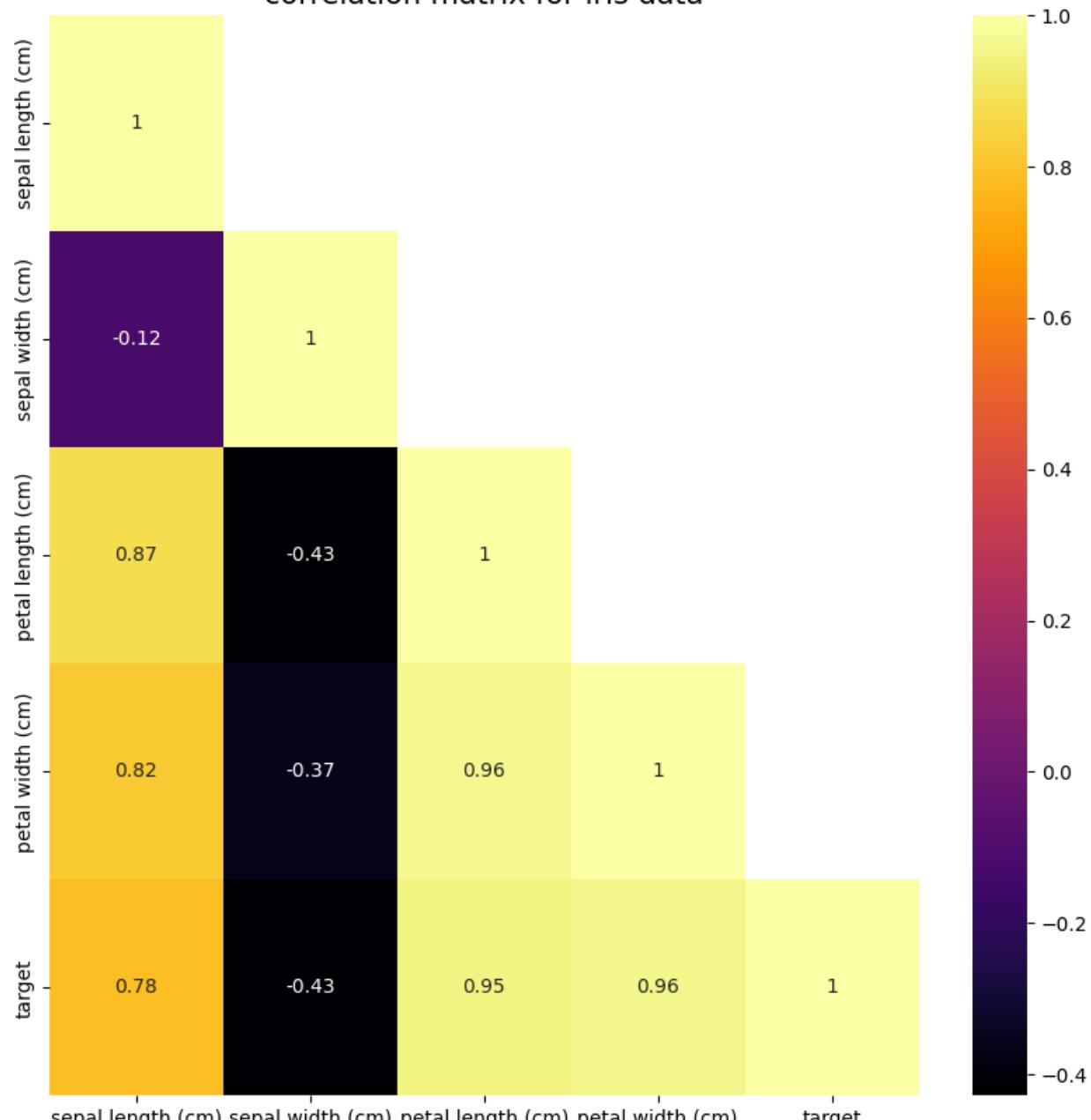
نتایج:



```
# Select columns with numerical data types
num = df.select_dtypes(exclude=['object']).columns
num

# Create a heatmap to visualize the correlation matrix of numerical
# columns
plt.figure(figsize=(10, 10))
sns.heatmap(df[num].corr(), annot=True, cmap='inferno',
mask=np.triu(df[num].corr(), k=1))
plt.title("correlation matrix for iris data", fontsize=15)
```

correlation matrix for iris data



تقسیم بندی داده ها به داده های آموزش و ارزیابی:

همانند قبل موارد گفته شده را رعایت می کنیم:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=93, stratify=y, shuffle=True)
```

```
# Display the dimensions of the training and testing sets
print(f'Dimensions of the training features: {X_train.shape}')
```

```
print(f'Dimensions of the training target: {y_train.shape}')
```

```
print(f'Dimensions of the testing features: {X_test.shape}')
```

```
print(f'Dimensions of the testing target: {y_test.shape}')
```

نتائج:

```
Dimensions of the training features: (120, 4)
```

```
Dimensions of the training target: (120,)
```

```
Dimensions of the testing features: (30, 4)
```

```
Dimensions of the testing target: (30,)
```



## بخش ۲

استفاده از روش های آماده پایتون، سه مدل بر مبنای رگرسیون لجستیک، MLP و شبکه های عصبی پایه شعاعی (RBF) را تعریف کرده و روی داده ها آموزش دهید. نتایج روی داده های ارزیابی را حداقل با چهار شاخص و ماتریس درهم ریختگی نشان داده و تحلیل کنید. در انتخاب فراپارامترها آزاد هستید؛ اما لازم است که نتایج را به صورت کامل مقایسه و تحلیل کنید. به دانشجویانی که این سوال را بدون استفاده از کتابخانه ها و مدل های آماده پایتونی انجام دهند، تا ۲۰ درصد نمره امتیازی تعلق خواهد گرفت.

پیاده سازی به کمک روش MLP :

همانند موارد گفته شده در کلاس حل تمرین پیاده سازی را انجام می دهیم:

```
model_1 = MLPClassifier(hidden_layer_sizes=(9,18,9), max_iter=800,  
random_state=93, verbose = False)  
model_1.fit(X_train, y_train)  
model_1.score(X_test, y_test)
```

نتایج:

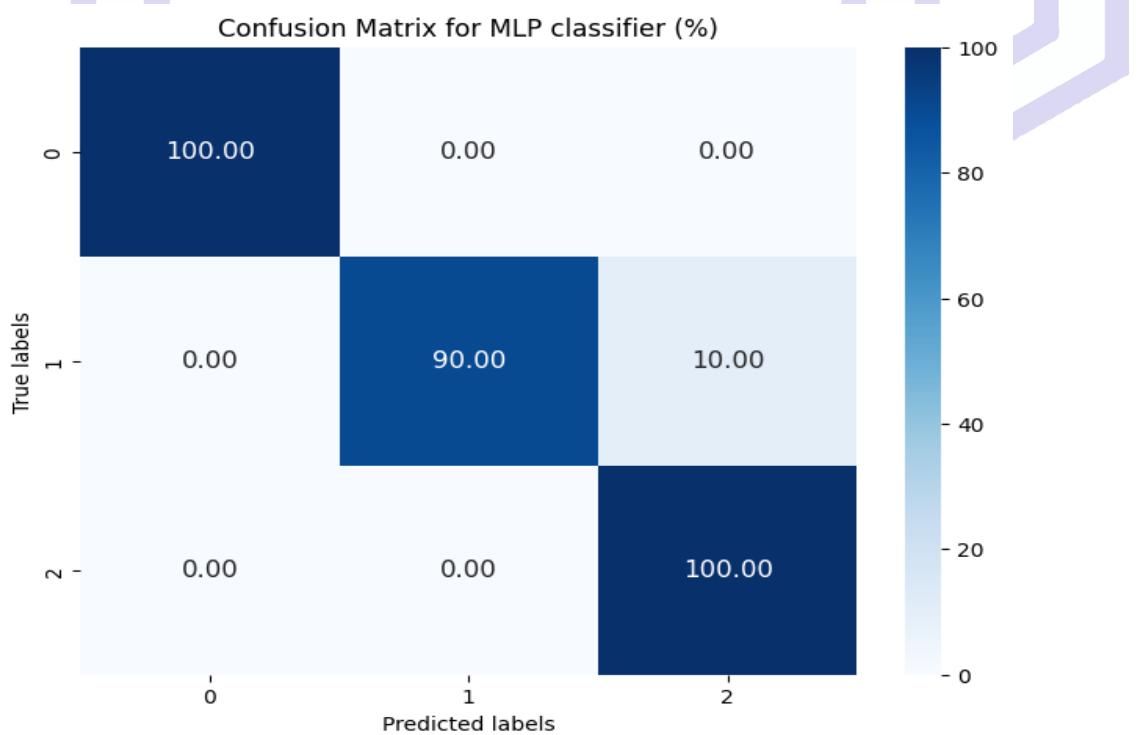
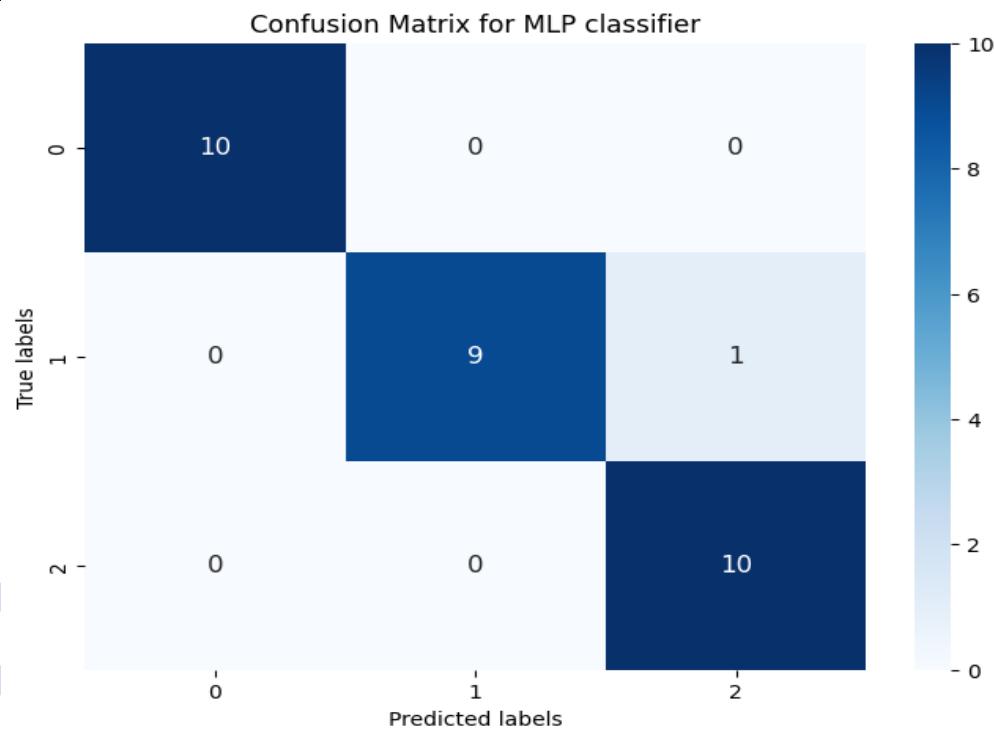
```
0.9666666666666667
```

همانطور که مشخص است به دقت مناسبی دست پیدا کردیم.

رسم ماتریس پیچیدگی:

```
# Making predictions on the test set  
y_pred = model_1.predict(X_test)  
  
# Calculating confusion matrix  
cf_matrix = confusion_matrix(y_test, y_pred)  
  
# Plotting confusion matrix as a heatmap with fitted text  
plt.figure(figsize=(8, 6))  
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',  
annot_kws={"size": 12})  
  
# Get the axis to modify layout  
plt.gca().set_yticks(len(np.unique(y_test)), 0) # Fix for matplotlib 3.1.1  
and 3.1.2  
plt.title('Confusion Matrix for MLP classifier')  
plt.xlabel('Predicted labels')  
plt.ylabel('True labels')  
  
# Printing classification report
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))
```



بررسی شاخص های ارزیابی:

Classification Report:					
	precision	recall	f1-score	support	
0	1.00	1.00	1.00	10	
1	1.00	0.90	0.95	10	
2	0.91	1.00	0.95	10	
accuracy			0.97	30	
macro avg	0.97	0.97	0.97	30	
weighted avg	0.97	0.97	0.97	30	

: Logistic Regression روش

مطابق روش مطرح شده در کلاس و به کمک روش های قبلی برای رسم ماتریس پیچیدگی داریم:

```
model_2 = LogisticRegression(max_iter=200, random_state=93)

model_2.fit(X_train, y_train)

# Making predictions on the test set
y_pred = model_2.predict(X_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})

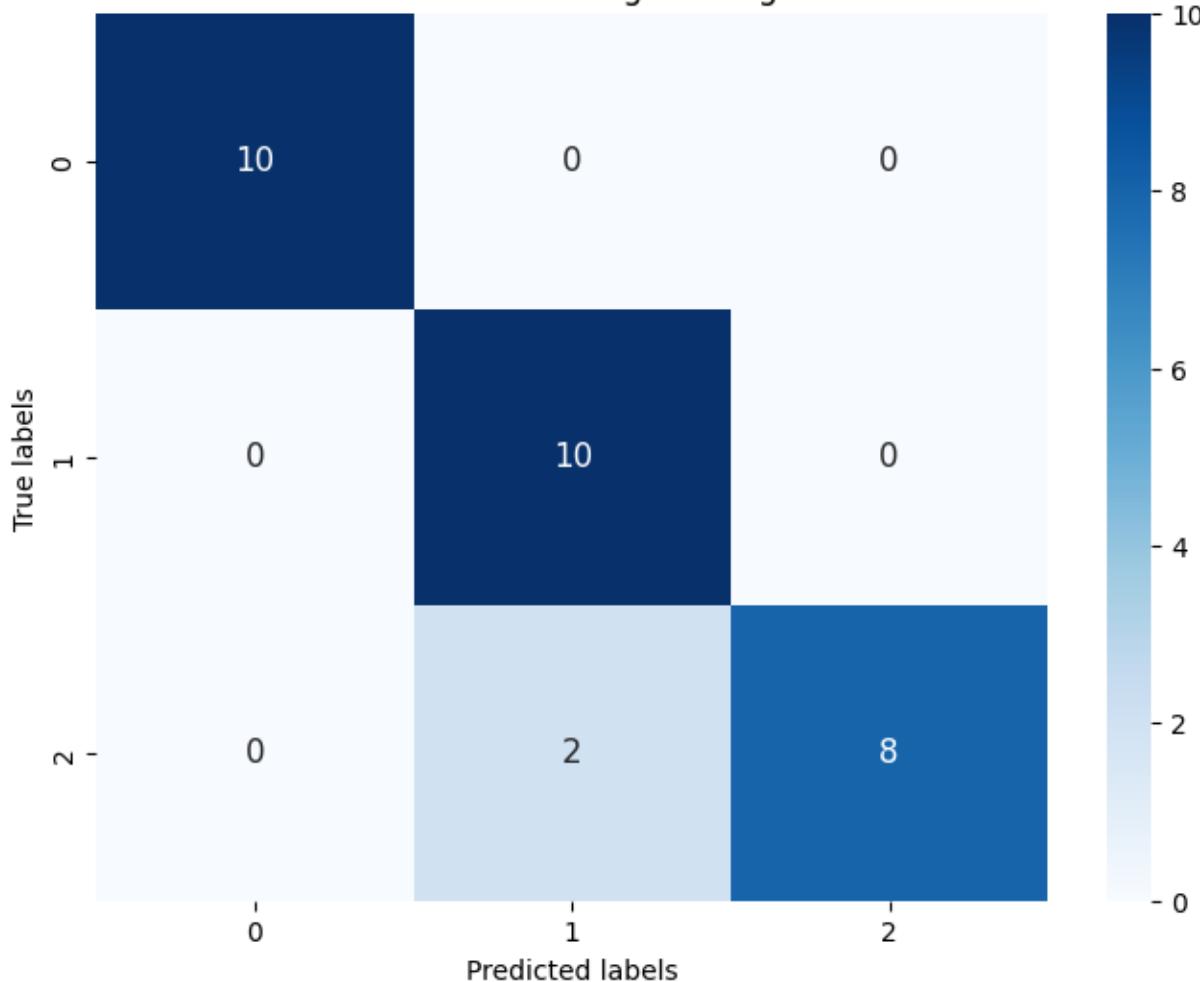
# Get the axis to modify layout
plt.gca().set_yticks(len(np.unique(y_test)), 0) # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix for Logistic Regression')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```



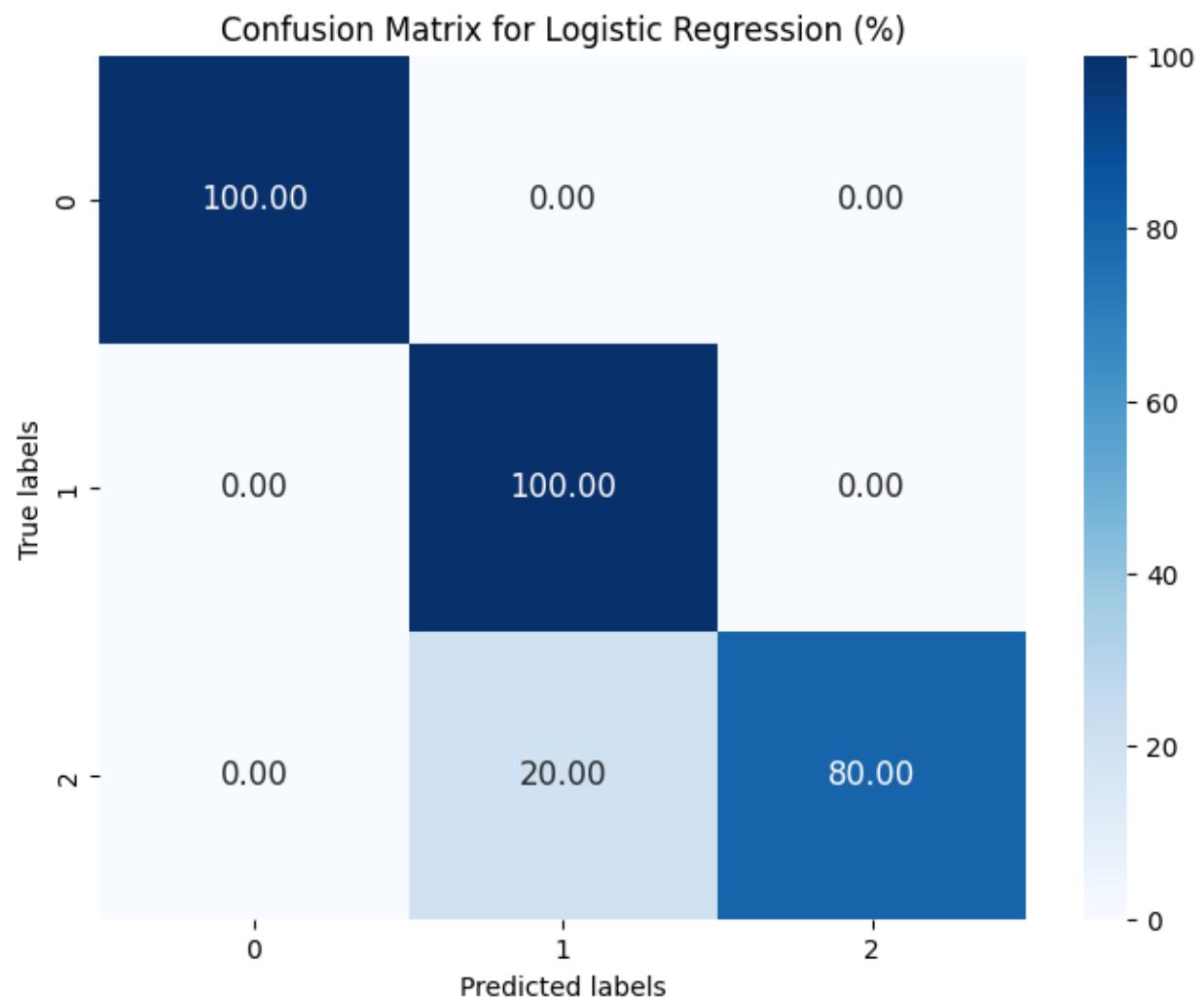
نتایج:

Confusion Matrix for Logistic Regression



بررسی به صورت درصدی:





بررسی شاخص های ارزیابی:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	0.83	1.00	0.91	10
2	1.00	0.80	0.89	10
accuracy			0.93	30
macro avg	0.94	0.93	0.93	30
weighted avg	0.94	0.93	0.93	30

## روش RBF

مطابق روش مطرح شده در کلاس و به کمک روش های قبلی برای رسم ماتریس پیچیدگی داریم:

```
# Assuming you've trained your model already
model_3 = SVC(kernel='rbf', random_state=93)
model_3.fit(X_train, y_train)

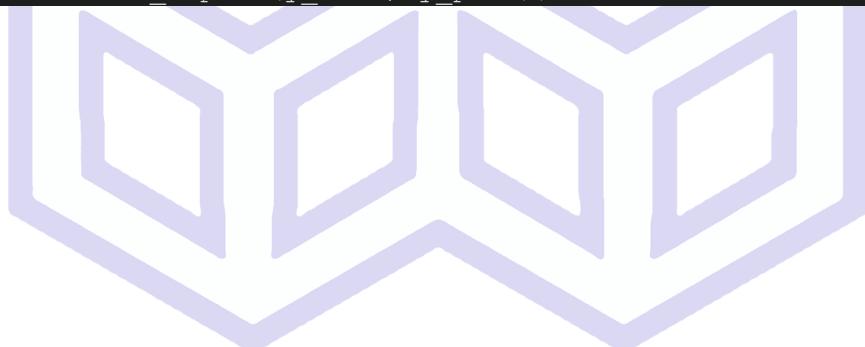
# Making predictions on the test set
y_pred = model_3.predict(X_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues',
annot_kws={"size": 12})

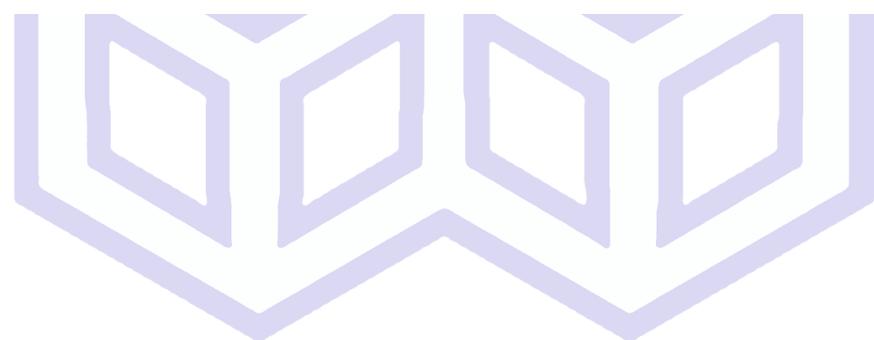
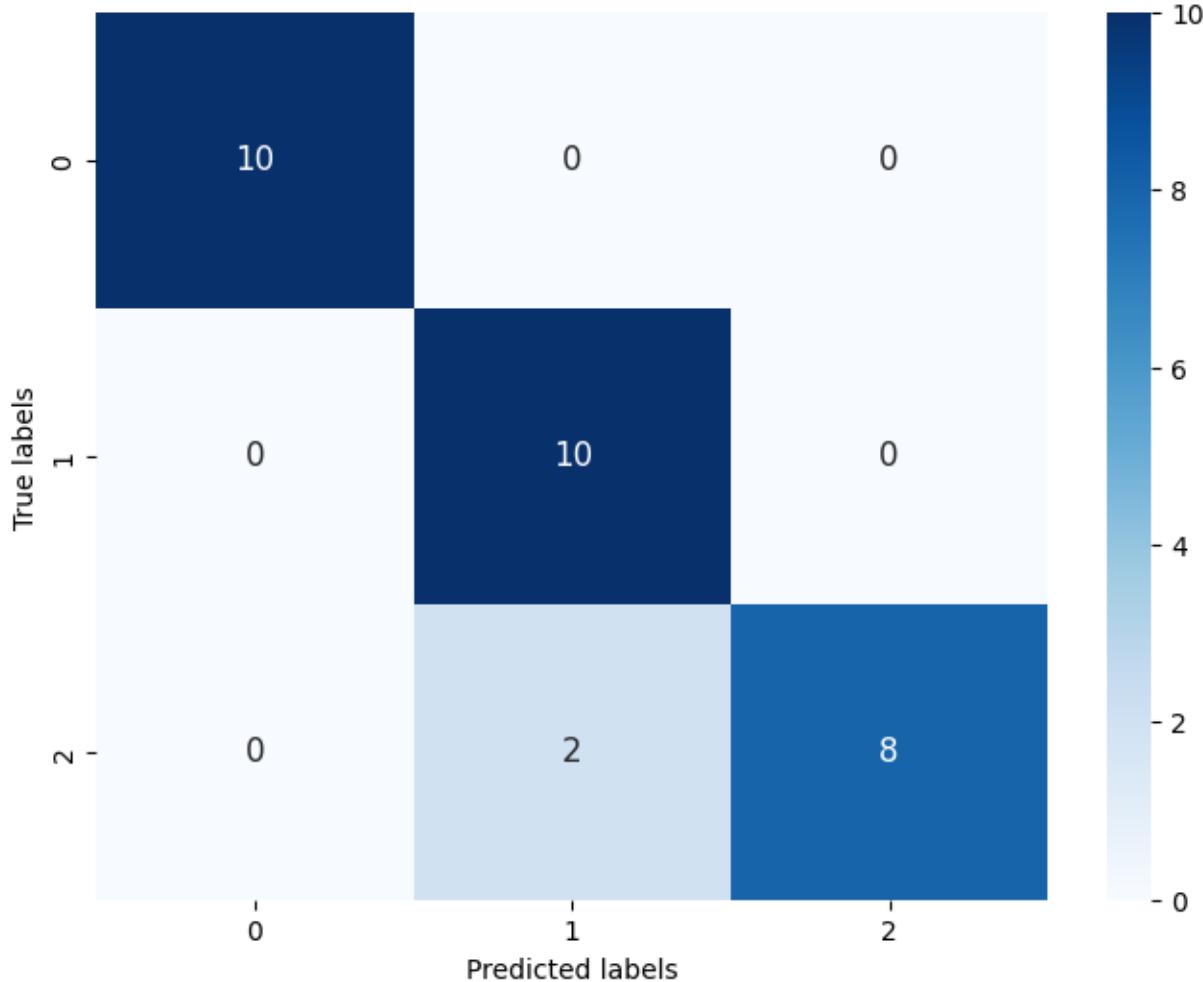
# Get the axis to modify layout
plt.gca().set_yticks(len(np.unique(y_test)), 0) # Fix for matplotlib 3.1.1
and 3.1.2
plt.title('Confusion Matrix for RBF')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

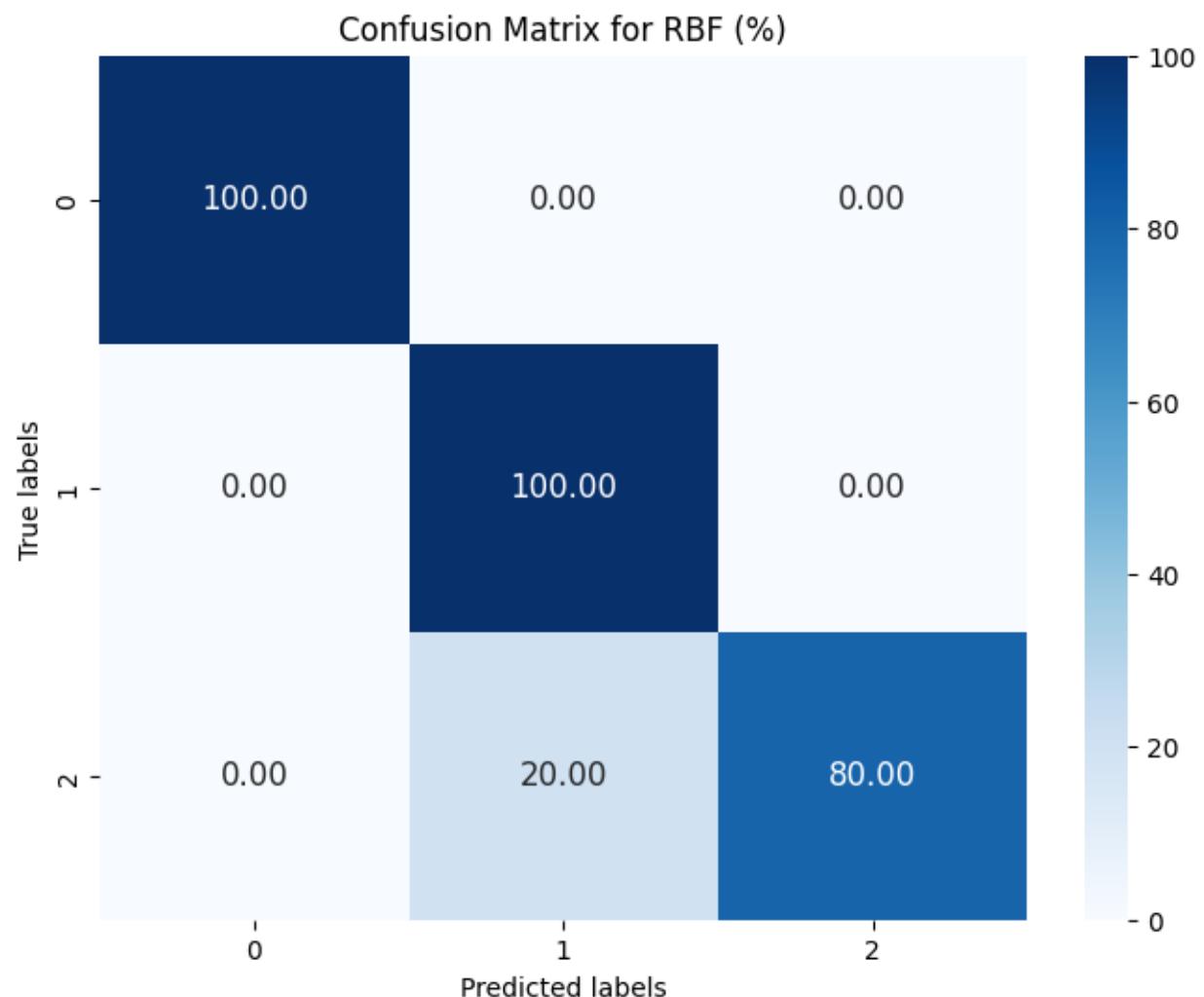


نتائج:

Confusion Matrix for RBF



بررسی به صورت درصدی:



معیار های ارزیابی:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	0.83	1.00	0.91	10
2	1.00	0.80	0.89	10
accuracy			0.93	30
macro avg	0.94	0.93	0.93	30
weighted avg	0.94	0.93	0.93	30

این معیارها کاملا در کلاس توضیح داده شده اند و همچنین در سوال سوم مینی پروژه اول توضیح داده شده اند.

معیار precision :

مشخص است که از لحاظ این معیار روش MLP دقیق‌تری دارد و بعد از آن دو روش Logistic Regression و RBF دقیق‌تری دارند. (برای کلاس صفر هر ۳ روش دقیق‌تری دارند و برای کلاس ۲ روش MLP دقیق‌تری دارد و بعد از آن دو روش Logistic Regression و RBF دقیق‌تری دارند. برای کلاس ۱ روش MLP دقیق‌تری دارد و بعد از آن دو روش Logistic Regression و RBF دقیق‌تری نسبت به دو روش Logistic Regression و RBF دارد.)

معیار recall :

مشخص است که این معیار در روش MLP دقیق‌تری دارد و بعد از آن دو روش Logistic Regression و RBF دقیق‌تری دارند. (برای کلاس صفر هر ۳ روش دقیق‌تری دارند و برای کلاس ۲ روش MLP دقیق‌تری دارد و بعد از آن دو روش Logistic Regression و RBF دقیق‌تری دارند. برای کلاس ۱ روش MLP دقیق‌تری دارد و بعد از آن دو روش Logistic Regression و RBF دقیق‌تری نسبت به دو روش Logistic Regression و RBF دارد.)

معیار f1 score :

مشخص است که این معیار در روش MLP دقیق‌تری دارد و بعد از آن دو روش Logistic Regression و RBF دقیق‌تری دارند. (برای کلاس صفر هر ۳ روش دقیق‌تری دارند و برای کلاس ۱ و ۲ روش MLP دقیق‌تری دارد و بعد از آن دو روش Logistic Regression و RBF دقیق‌تری دارند.)

معیار accuracy :

مشخص است که این معیار در روش MLP دقیق‌تری دارد و بعد از آن دو روش Logistic Regression و RBF دقیق‌تری دارند.

اگر بخواهیم این روش ها را بدون استفاده از کتابخانه های آماده انجام دهیم، مطابق با آنچه در کلاس حل تمرین گفته شد به صورت زیر می توانیم عمل کنیم:

ابتدا **activation function** و **loss** و **توابع accuracy** را تعریف می کنیم.

: MLP روش

```
class MLP:

    def __init__(self, hidden_layer_sizes, hidden_activation='relu',
                 output_size=1, output_activation='sigmoid',
                 n_iter=1000, loss_fn=bce, eta=0.1):

        self.hidden_layer_sizes = hidden_layer_sizes # List of hidden
        layer sizes
        self.hidden_activation = hidden_activation # Activation function
        for hidden layers
        self.output_size = output_size # Output layer size
        self.output_activation = output_activation # Activation function
        for output layer
        self.n_iter = n_iter # Number of iterations for training
        self.loss_fn = loss_fn # Loss function for training
        self.eta = eta # Learning rate

    def _init_weights(self):
        self.ws, self.bs = [], [] # Weight and bias lists for each layer
        all_layers = [self.input_size] + self.hidden_layer_sizes +
        [self.output_size] # All layer sizes
        num_layers = len(all_layers)
        for i in range(1, num_layers):
            w = np.random.randn(all_layers[i-1], all_layers[i]) # Randomly initialize weights
            b = np.random.randn(all_layers[i]) # Randomly initialize
            biases
            self.ws.append(w)
            self.bs.append(b)

    def fit(self, X, y):
        n, self.input_size = X.shape # Number of samples and input size
        self._init_weights()
        for _ in range(self.n_iter):
            y_hat = self.predict(X) # Make predictions
            loss = self.loss_fn(y, y_hat) # Compute loss
```



```

        self._gradient_descent(X, y, y_hat) # Update weights and
biases
        print(loss) # Print loss

    def _gradient_descent(self, X, y, y_hat):
        delta = y_hat - y # Compute difference between predicted and true
values
        for j in range(len(self.ws)-1, 0, -1):
            w_grad = (self.as_[j-1].T @ delta) / len(y) # Compute weight
gradient
            b_grad = delta.mean(0) # Compute bias gradient
            self.ws[j] -= self.eta * w_grad # Update weights
            self.bs[j] -= self.eta * b_grad # Update biases
            delta = (delta @ self.ws[j].T) *
(self._activation_derivative(self.hs[j-1], self.hidden_activation))

    def predict(self, X):
        self.hs = [] # Hidden layer outputs
        self.as_ = [] # Activation function outputs
        for i, (w, b) in enumerate(zip(self.ws[:-1], self.bs[:-1])):
            a = self.as_[i-1].copy() if i>0 else X.copy() # Input to the
hidden layer
            self.hs.append(a @ w + b) # Compute hidden layer output
            self.as_.append(self._activation_function(self.hs[i],
self.hidden_activation)) # Apply activation function
        y = self._activation_function(self.as_[-1] @ self.ws[-1] +
self.bs[-1], self.output_activation) # Output layer activation
        return y

    def _activation_function(self, x, activation):
        if activation == 'relu':
            return np.maximum(0, x) # ReLU activation
        elif activation == 'sigmoid':
            return 1 / (1 + np.exp(-x)) # Sigmoid activation
        else:
            raise ValueError("Invalid activation function.")

    def _activation_derivative(self, x, activation):
        if activation == 'relu':
            return np.where(x > 0, 1, 0) # Derivative of ReLU activation
        elif activation == 'sigmoid':
            sigmoid = self._activation_function(x, 'sigmoid')
            return sigmoid * (1 - sigmoid) # Derivative of Sigmoid
activation
        else:

```



```
raise ValueError("Invalid activation function.")
```

## روش Logistic regression

```
class Logistic_regression:

    def __init__(self, in_features, n_iter=100, eta=0.1, verbose=True):
        self.in_features = in_features
        # weight & bias
        self.w = np.random.randn(in_features, 1)
        self.b = np.random.randn()
        self.af = sigmoid
        self.loss_fn = bce
        self.loss_hist = []
        self.w_grad, self.b_grad = None, None
        self.n_iter = n_iter
        self.eta = eta
        self.verbose = verbose

    def predict(self, x):
        # x: [n_samples, in_features]
        y_hat = x @ self.w + self.b
        y_hat = y_hat if self.af is None else self.af(y_hat)
        return y_hat

    def fit(self, x, y):
        for i in range(self.n_iter):
            #model
            y_hat = self.predict(x)
            #loss
            loss = self.loss_fn(y, y_hat)
            self.loss_hist.append(loss)
            #grad
            self.gradient(x, y, y_hat)
            #optimize
            self.gradient_descent()
            #print results
            if self.verbose & (i % 10 == 0):
                print(f'Iter={i}, Loss={loss:.4f}')

    def gradient(self, x, y, y_hat):
        self.w_grad = (x.T @ (y_hat - y)) / len(y)
        self.b_grad = (y_hat - y).mean()

    def gradient_descent(self):
```



```

        self.w -= self.eta * self.w_grad
        self.b -= self.eta * self.b_grad

    def __repr__(self):
        return f'Neuron({self.in_features}, {self.af.__name__})'

    def parameters(self):
        return {'w': self.w, 'b': self.b}

```

: RBF روشن

```

class RBF:
    def __init__(self, num_centers, learning_rate=0.01, epochs=1000):
        self.num_centers = num_centers
        self.learning_rate = learning_rate
        self.epochs = epochs

    def gaussian_rbf(self, x, c, sigma):
        return np.exp(-np.linalg.norm(x - c) / (2 * sigma**2))

    def train(self, X, y):
        self.centers = X[np.random.choice(X.shape[0], self.num_centers,
replace=False)]
        self.sigma = np.std(X)

        self.weights = np.random.rand(self.num_centers)

        for epoch in range(self.epochs):
            for i in range(X.shape[0]):
                phi = np.array([self.gaussian_rbf(X[i], c, self.sigma) for
c in self.centers])
                prediction = np.dot(phi, self.weights)
                error = y[i] - prediction

                # Update weights
                self.weights += self.learning_rate * error * phi

    def predict(self, X):
        predictions = []
        for i in range(X.shape[0]):
            phi = np.array([self.gaussian_rbf(X[i], c, self.sigma) for c
in self.centers])
            prediction = np.dot(phi, self.weights)
            predictions.append(prediction)

        return predictions

```

