دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

مبانی سیستم‌های هوشمند

پروژه دوم

استاد درس: دکتر علیاری

نام و نام خانوادگی: زهرا ایران‌پور مبارکه

شماره دانشجویی: ۹۸۱۹۸۹۳

زمستان ۱۴۰۲

فهرست

سوال ۱

ابتدا کتابخانه‌های مورد نیاز در این سوال اضافه می‌شود:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification, make_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from mlxtend.plotting import plot_decision_regions
import __main__
```

سپس دیتاست داده شده را وارد می‌کنیم:

```
!pip install --no-cache-dir gdown
!gdown 1k2soBitLn6oU23YnEgkJh3byScFGLgc1
df = pd.read_csv('Perceptron.csv')
df
```

	x1	x2	y
0	1.028503	0.973218	-1.0
1	0.252505	0.955872	-1.0
2	1.508085	0.672058	-1.0
3	1.940002	1.721370	-1.0
4	-1.048819	-0.844999	1.0
...	...	...	...
395	0.574634	0.782211	-1.0
396	-1.413307	-0.673049	1.0
397	-0.465114	-1.290830	1.0
398	1.522055	0.948007	-1.0
399	0.834118	0.926710	-1.0

۱. در این قسمت با استفاده از train-test-split داده‌ها را به دو بخش آموزش و تست تقسیم می‌کنیم:

```
X = df.iloc[:, 0:2]
y = df.iloc[:, 2:3]
y = np.where(y == -1, 0, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

چک می‌کنیم که در داده‌ها خانه خالی نباشد:

```
df.isna().sum()
```

```
x1    0
x2    0
y     0
dtype: int64
```

و در ادامه طبق قائده پرسپترون یک نورون روی داده‌های آموزش، آموزش می‌دهیم.

برای استفاده از داده‌ها در کد نیاز است که تغییراتی روی نحوه آن‌ها صورت گیرد برای مثال reshape شوند:

```
X_train=np.asarray(X_train)
y_train=np.array(y_train)
y_train=y_train.reshape(-1,1)
y_train.shape,X_train.shape
```

سپس توابع فعال‌سازی را تعریف می‌کنیم:

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

تابع تلفات و دقت نیز باید تعریف شود:

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


def mse(y, y_hat):
   return np.mean((y - y_hat)**2)


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
```

و نهایتا به سراغ تعریف نورون می‌رویم:

```
class Neuron:
   def __init__(self, in_features, af=None, loss_fn=mse, n_iter=100, eta=0.1, verbose=True):
```

تعداد ویژگی‌های ورودی به عنوان ویژگی‌های نورون:

```
      self.in_features = in_features
```

وزن‌ها و انحراف: (بایاس را روی ۵ به دلخواه قرار می‌دهیم تا آستانه را مشخص کرده باشیم)

```
      self.w = np.random.randn(in_features, 1)
      self.b = 5
```

تابع فعال‌سازی - (Activation Function) می‌تواند None باشد:

```
      self.af = af
```

تابع خطا و لیستی برای ذخیره خطا در هر مرحله:

```
      self.loss_fn = loss_fn
      self.loss_hist = []
```

گرادیان‌های وزن و انحراف:

```
      self.w_grad, self.b_grad = None, None
```

تعداد تکرارها در فرآیند آموزش، نرخ یادگیری و نمایش پیام‌ها در هر مرحله یادگیری:

```
      self.n_iter = n_iter
      self.eta = eta
      self.verbose = verbose


   def predict(self, x):
```

x: [تعداد نمونه‌ها, تعداد ویژگی‌ها]

```
    y_hat = x @ self.w + self.b
```

اعمال تابع فعال‌سازی اگر تعیین شده باشد:

```
    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):
```

پیش‌بینی خروجی با استفاده از نورون:

```
        y_hat = self.predict(x)
```

محاسبه خطا و ذخیره خطا در لیست تاریخچه:

```
        loss = self.loss_fn(y, y_hat)
        self.loss_hist.append(loss)
```

محاسبه گرادیان‌ها و اعمال گرادیان به وزن‌ها و انحراف:

```
        self.gradient(x, y, y_hat)
        self.gradient_descent()
```

نمایش وضعیت هر چند مرحله

```
        if self.verbose & (i % 10 == 0):
            print(f'Iter={i}, Loss={loss.mean():.4f}')

def gradient(self, x, y, y_hat):
```

محاسبه گرادیان وزن و انحراف:

```
    self.w_grad = (x.T @ (y_hat - y)) / len(y)
    self.b_grad = (y_hat - y).mean()

def gradient_descent(self):
```

اعمال گرادیان به وزن‌ها و انحراف:

```
    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):
```

بازگرداندن وزن‌ها و انحراف به عنوان پارامترهای نورون:

```
    return {'w': self.w, 'b': self.b}
```

فیت کردن مدل روی دیتا:

```
neuron = Neuron(in_features=2, af=sigmoid, loss_fn=bce, n_iter=100, eta=0.1, verbose=False)
neuron.fit(X_train, y_train)
neuron.parameters()
```

```
{'w': array([[-2.17219826],
        [-2.73640453]]),
 'b': 2.7389818597666107}
```

۲. در این قسمت مدل را روی داده‌های آزمون تست کرده و دقت را بدست می‌آوریم

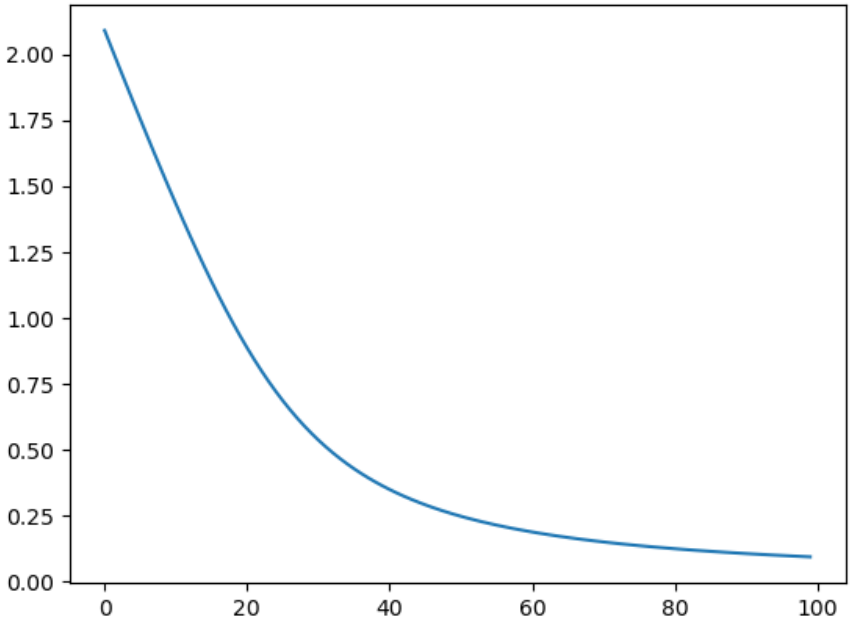
پیش‌بینی خروجی دیتای آزمون:

```
y_hat = neuron.predict(X_test)
accuracy(y_test, y_hat, t=0.5)
```

```
0.975
```

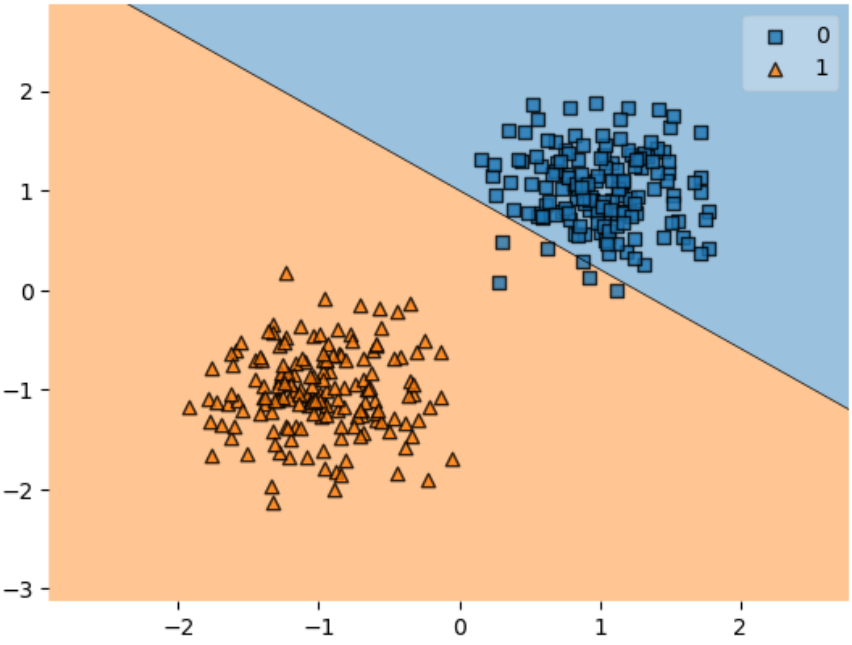
رسم تابع تلفات:

```
plt.plot(neuron.loss_hist)
```



رسم داده‌های تقسیم‌بندی و جدا شده به دو بخش:

```
X_train=np.asarray(X_train)
y_train_1d = np.ravel(y_train)
plot_decision_regions(X_train, y_train_1d, clf=neuron)
```



۳. در این قسمت از سوال آستانه (ترشهولد) را تغییر می‌دهیم تا نتیجه آن بر روی خروجی را ببینیم. در قسمت تعریف نورون، عدد بایاس را از ۵ به ۱۰ تغییر می‌دهیم و مدل را فیت کرده و باقی کد به همان صورت پیش می‌رود. خروجی‌ها به این صورت خواهند بود:

پارامترهای مدل:

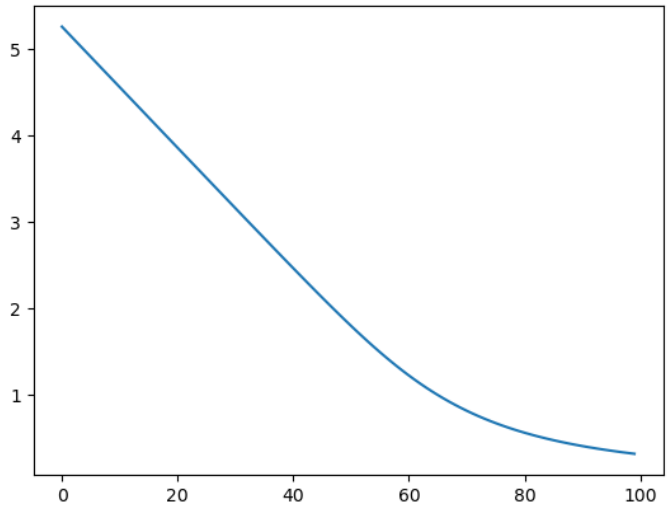
```
{'w': array([[-2.60960353],
        [-4.21637602]]),
 'b': 6.021208076198224}
```

دقت:

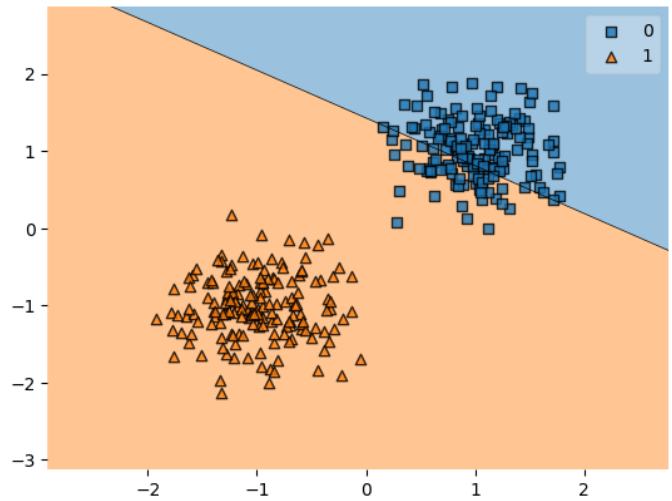
```
accuracy
```

```
0.8375
```

تابع تلفات:



خط جداکننده دیتا:



متوجه می‌شویم که با افزایش ترشهولد، دقت کاهش پیدا کرد، تابع تلفات از حالت ایده‌آل دورتر شد و خط جدا کننده تعداد بیشتری از داده‌ها را به صورت اشتباه طبقه‌بندی کرد.

با حذف بایاس و تغییر آن به عدد صفر:

پارامترهای مدل:

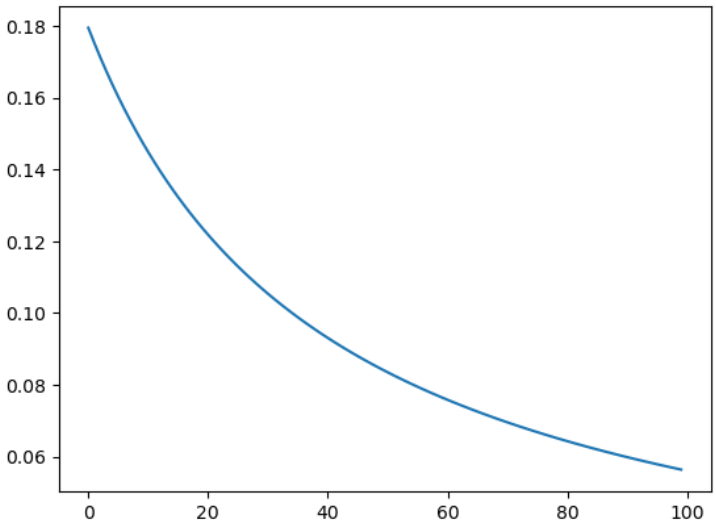
```
{'w': array([[-1.73493934],
       [-1.45195735]]),
 'b': 0.036838419246040754}
```

دقت:

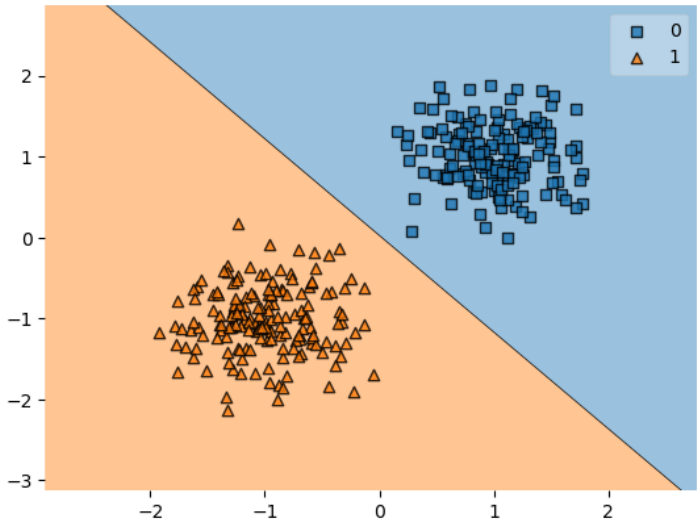
```
accuracy
```

```
1.0
```

تابع تلفات:



خط جداکننده دیتا:



متوجه می‌شویم که با صفرشدن بایاس، دقت افزایش پیدا کرد و به بهترین حالت خود رسید، تابع تلفات بهتر شد و خط جدا کننده تمام داده‌ها را به صورت درست طبقه‌بندی کرد.

آستانه (Threshold) در پرسپترون یک مقدار حدی است که در پیش‌بینی خروجی نورون بر اساس مقدار خروجی نجومی (استفاده از تابع فعال‌سازی) تعیین می‌شود. انتخاب مناسب آستانه می‌تواند تأثیر زیادی در نتایج مدل داشته باشد. در زیر تاثیرات انتخاب آستانه در پرسپترون را بررسی می‌کنیم:

- مرز تصمیم (Decision Boundary): آستانه مقداری است که تعیین می‌کند کدام خروجی به عنوان کلاس ۰ و کدام خروجی به عنوان کلاس ۱ تشخیص داده شود. تغییر آستانه می‌تواند مرز تصمیم را به شکل قابل توجهی تغییر دهد.
- دقت (Accuracy): انتخاب مناسب آستانه می‌تواند به بهبود دقت مدل کمک کند. برخی مواقع با تغییر آستانه، مدل ممکن است بهترین تصمیمات خود را بگیرد و دقت کلی را افزایش دهد.
- حساسیت و ویژگی تشخیص (Sensitivity/Recall و Specificity): انتخاب آستانه می‌تواند حساسیت (توانایی تشخیص مثبت‌ها) و ویژگی تشخیص (توانایی تشخیص منفی‌ها) را تحت تأثیر قرار دهد. این ممکن است به تعادل میان دقت در کلاس‌های مختلف کمک کند.
- ماتریس درهم‌رباطی (Confusion Matrix): با تغییر آستانه، ماتریس درهم‌رباطی نیز تغییر می‌کند و ممکن است دسته‌های مختلف در ماتریس به صورت متفاوتی تشخیص داده شوند.
- تاثیر در مواجهه با داده‌های نویزی (Noise): در مواجهه با داده‌های نویزی، انتخاب آستانه مناسب می‌تواند تأثیرات داده‌های نویزی را کاهش دهد یا از آن جلوگیری کند.

به طور کلی، انتخاب آستانه در پرسپترون یکی از موارد مهم در تنظیم مدل است. ممکن است نیاز باشد آستانه را به صورت آزمایشی تغییر داده و تأثیرات آن را بر روی معیارهای ارزیابی مدل (مانند دقت، حساسیت، و ویژگی تشخیص) مشاهده کرد.

سوال ۲

۱. در این قسمت با نورون توسعه‌یافته یه ضرب‌کننده باینری ساخته و اولویت در شبکه خروجی این است که کمترین نورون و کمترین آستانه را داشته باشیم و تمام شبکه برای یک خروجی دارای آستانه یکسان باشد.

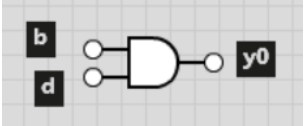
جدول درستی:

a	b	c	d	Y3	y2	y1	y0
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

جداول کارنو: (رسم با سایت http://tma.main.jp/logic/index_en.html)

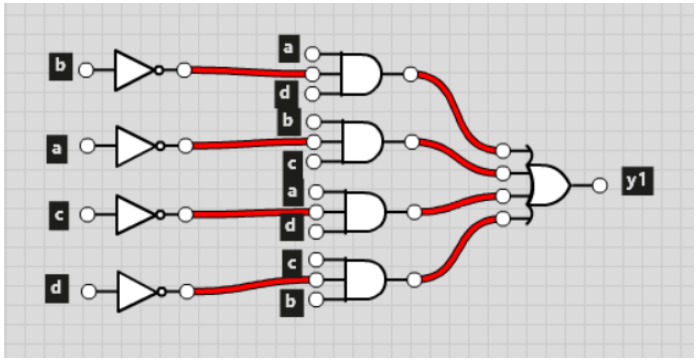
	$\bar{c}\bar{d}$	$\bar{c}d$	cd	$c\bar{d}$
$\bar{a}\bar{b}$	0	0	0	0
$\bar{a}b$	0	1	1	0
ab	0	1	1	0
$a\bar{b}$	0	0	0	0

y0 = bd



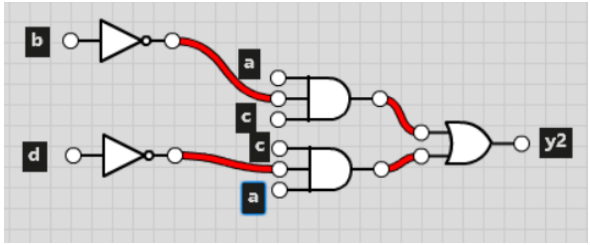
	$\bar{c}\bar{d}$	$\bar{c}d$	cd	$c\bar{d}$
$\bar{a}\bar{b}$	0	0	0	0
$\bar{a}b$	0	0	1	1
$a\bar{b}$	0	1	0	1
ab	0	1	1	0

y1 = ab’d+a’bc+c’da+cd’b



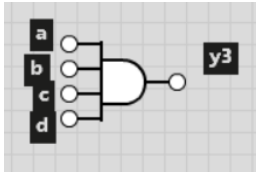
	$\bar{c}\bar{d}$	$\bar{c}d$	cd	$c\bar{d}$
$\bar{a}\bar{b}$	0	0	0	0
$\bar{a}b$	0	0	0	0
$a\bar{b}$	0	0	0	1
ab	0	0	1	1

y2 = ab’c+cd’a



	$\bar{c}\bar{d}$	$\bar{c}d$	cd	$c\bar{d}$
$\bar{a}\bar{b}$	0	0	0	0
$\bar{a}b$	0	0	0	0
$a\bar{b}$	0	0	1	0
ab	0	0	0	0

y3 = abcd



۲. پیاده‌سازی شبکه‌های طراحی شده به کمک پایتون:

ابتدا کتابخانه‌های مورد نیاز را وارد می‌کنیم:

```
import numpy as np
import itertools
```

سپس نورون را تعریف می‌کنیم:

```
class McCulloch_Pitts_neuron():
```

دریافت وزن و آستانه:

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

انجام ضرب داخلی و مقایسه آن با ترشهولد:

```
    if self.weights @ x >= self.threshold:
       return 1
    else:
       return 0
```

تعریف ورودی‌ها:

```
input = [0,1]
X = list(itertools.product(input, input, input, input))
```

در ادامه برای بیت اول نورون and را تعریف می‌کنیم که شامل دو ورودی با وزن ۱ و بایاس ۱.۵ است:

```
def y0(input):
  neur1 = McCulloch_Pitts_neuron([1,1],1.5)
  z1 = neur1.model(np.array([input[1],input[3]]))
  return list([z1])
```

سپس جدول حالات را برای بیت اول بدست می‌آوریم. به این صورت که ورودی که ۴بیت دارد را به نورون داده تا خروجی حاصل شود.

```
for i in X:
    res = y0(i)
    print("y0 for input:", str(i[0]) + str(" ")+str(i[1])+ str(" ")+str(i[2])+ str(" ")+str(i[3])+str(","), "is:",
str(res[0]))
```

```
y0 for input: 0 0 0 0, is: 0
y0 for input: 0 0 0 1, is: 0
y0 for input: 0 0 1 0, is: 0
y0 for input: 0 0 1 1, is: 0
y0 for input: 0 1 0 0, is: 0
y0 for input: 0 1 0 1, is: 1
y0 for input: 0 1 1 0, is: 0
y0 for input: 0 1 1 1, is: 1
y0 for input: 1 0 0 0, is: 0
y0 for input: 1 0 0 1, is: 0
y0 for input: 1 0 1 0, is: 0
y0 for input: 1 0 1 1, is: 0
y0 for input: 1 1 0 0, is: 0
y0 for input: 1 1 0 1, is: 1
y0 for input: 1 1 1 0, is: 0
y0 for input: 1 1 1 1, is: 1
```

برای بیت دوم، ۴ نورون and و یک نورون or نیاز داریم. در این مرحله حواسمان هست که بایاس نورون or را باید ۰.۵ قرار دهیم. وزن‌ها به ترتیب ورودی‌ها مشخص می‌شود. در جایی که نیاز به استفاده از گیت not هست، عدد -۱ قرار می‌دهیم.

```
def y1(input):
  neur2 = McCulloch_Pitts_neuron([1,-1, 0,1], 1.5)
  neur3 = McCulloch_Pitts_neuron([1,0, -1,1], 1.5)
  neur4 = McCulloch_Pitts_neuron([0,1,1,-1], 1.5)
  neur5 = McCulloch_Pitts_neuron([-1,1,1,0], 1.5)
  neur6 = McCulloch_Pitts_neuron([1,1,1,1], 0.5)
  z2 = neur2.model(np.array([input[0], input[1],input[2],input[3]]))
  z3 = neur3.model(np.array([input[0], input[1],input[2],input[3]]))
  z4 = neur4.model(np.array([input[0], input[1],input[2],input[3]]))
  z5 = neur5.model(np.array([input[0], input[1],input[2],input[3]]))
  z6 = neur6.model(np.array([z2,z3,z4,z5]))
  return list([z6])

for i in X:
    res = y1(i)
    print("y1 for input:", str(i[0]) + str(" ")+str(i[1])+ str(" ")+str(i[2])+ str(" ")+str(i[3])+str(","), "is:",
str(res[0]))
```

```
y1 for input: 0 0 0 0, is: 0
y1 for input: 0 0 0 1, is: 0
y1 for input: 0 0 1 0, is: 0
y1 for input: 0 0 1 1, is: 0
y1 for input: 0 1 0 0, is: 0
y1 for input: 0 1 0 1, is: 0
y1 for input: 0 1 1 0, is: 1
y1 for input: 0 1 1 1, is: 1
y1 for input: 1 0 0 0, is: 0
y1 for input: 1 0 0 1, is: 1
y1 for input: 1 0 1 0, is: 0
y1 for input: 1 0 1 1, is: 1
y1 for input: 1 1 0 0, is: 0
y1 for input: 1 1 0 1, is: 1
y1 for input: 1 1 1 0, is: 1
y1 for input: 1 1 1 1, is: 0
```

برای بیت سوم، ۲ نورون and و یک نورون or نیاز داریم. به همان روش قبل پیش رفته و خروجی نورون‌های and را به or می‌دهیم:

```
def y2(input):
  neur7 = McCulloch_Pitts_neuron([1,0,1,-1], 1.5)
  neur8 = McCulloch_Pitts_neuron([1,-1,1,0], 1.5)
  neur9 = McCulloch_Pitts_neuron([1,1], 0.5)
  z7 = neur7.model(np.array([input[0], input[1],input[2],input[3]]))
  z8 = neur8.model(np.array([input[0], input[1],input[2],input[3]]))
  z9 = neur9.model(np.array([z7,z8]))
  return list([z9])

for i in X:
```

```
    res = y2(i)
    print("y2 for input:", str(i[0]) + str(" ")+str(i[1])+ str(" ")+str(i[2])+ str(" ")+str(i[3])+str(","), "is:", str(res[0]))
```

```
y2 for input: 0 0 0 0, is: 0
y2 for input: 0 0 0 1, is: 0
y2 for input: 0 0 1 0, is: 0
y2 for input: 0 0 1 1, is: 0
y2 for input: 0 1 0 0, is: 0
y2 for input: 0 1 0 1, is: 0
y2 for input: 0 1 1 0, is: 0
y2 for input: 0 1 1 1, is: 0
y2 for input: 1 0 0 0, is: 0
y2 for input: 1 0 0 1, is: 0
y2 for input: 1 0 1 0, is: 1
y2 for input: 1 0 1 1, is: 1
y2 for input: 1 1 0 0, is: 0
y2 for input: 1 1 0 1, is: 0
y2 for input: 1 1 1 0, is: 1
y2 for input: 1 1 1 1, is: 0
```

بیت آخر فقط به یک نورون and نیاز دارد. فقط باید توجه داشت که برای گیت and ۴بیتی بایاس برابر ۳.۵ است.

```
def y3(input):
  neur10 = McCulloch_Pitts_neuron([1,1,1,1], 3.5)
  z10 = neur10.model(np.array([input[0], input[1],input[2],input[3]]))
  return list([z10])

for i in X:
    res = y3(i)
    print("y3 for input:", str(i[0]) + str(" ")+str(i[1])+ str(" ")+str(i[2])+ str(" ")+str(i[3])+str(","), "is:", str(res[0]))
```

```
y3 for input: 0 0 0 0, is: 0
y3 for input: 0 0 0 1, is: 0
y3 for input: 0 0 1 0, is: 0
y3 for input: 0 0 1 1, is: 0
y3 for input: 0 1 0 0, is: 0
y3 for input: 0 1 0 1, is: 0
y3 for input: 0 1 1 0, is: 0
y3 for input: 0 1 1 1, is: 0
y3 for input: 1 0 0 0, is: 0
y3 for input: 1 0 0 1, is: 0
y3 for input: 1 0 1 0, is: 0
y3 for input: 1 0 1 1, is: 0
y3 for input: 1 1 0 0, is: 0
y3 for input: 1 1 0 1, is: 0
y3 for input: 1 1 1 0, is: 0
y3 for input: 1 1 1 1, is: 1
```

سوال ۳

ابتدا کتابخانه‌های مورد نیاز در این سوال اضافه می‌شود:

```
from PIL import Image, ImageDraw
import random
from pylab import*
from math import sqrt
import matplotlib.pyplot as plt
import os
```

سپس ۵ داده تصویری الفبای فارسی را دریافت می‌کنیم:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1QTi7dJtNAfFR5mG0rd8K3ZGvEIfSn_DS
!unzip PersianData.zip
```

۱. در این مرحله به تشریح ۲ تابع پایتون می‌پردازیم

تابع اول، تصویر را در ورودی خود دریافت کرده و به صورت نمایش باینری در می‌آورد.

این کد یک تابع به نام `convertImageToBinary` را تعریف می‌کند که یک تصویر را به نمایش دودویی تبدیل می‌کند. این تابع از کتابخانه‌ی PIL (Python Imaging Library) برای کار با تصاویر استفاده می‌کند.

تابع با دریافت مسیر فایل تصویر (`path`) شروع می‌شود. سپس تصویر با استفاده از `Image.open(path)` باز می‌شود.

سپس یک ابزار ترسیم (`ImageDraw`) برای اعمال تغییرات بر روی تصویر ایجاد می‌شود.

عرض (`width`) و ارتفاع (`height`) تصویر استخراج می‌شوند و با استفاده از `image.load()`، اطلاعات پیکسل تصویر به متغیر `pix` اختصاص داده می‌شود.

یک عامل بازتاب (thresholding factor) به نام `factor` تعریف می‌شود. سپس یک لیست تهی به نام `binary_representation` برای ذخیره نمایش دودویی تصویر ایجاد می‌شود.

در این مرحله، یک حلقه دوتایی (`for` nested inside another `for`) برای اسکن کل پیکسل‌های تصویر شروع می‌شود. برای هر پیکسل، مقادیر RGB آن استخراج شده و مجموع این مقادیر به عنوان شدت (intensity) محاسبه می‌شود.

سپس بر اساس شدت، تصمیم گرفته می‌شود که آیا پیکسل مورد نظر سفید یا سیاه باشد. اگر شدت بیشتر از یک حد آستانه باشد، پیکسل به رنگ سفید تبدیل شده و مقدار -۱ به لیست `binary_representation` اضافه می‌شود. در غیر این صورت، پیکسل به رنگ سیاه تبدیل شده و مقدار ۱ به لیست اضافه می‌شود.

در ادامه، رنگ پیکسل با توجه به تصمیمات بالا تغییر داده و نقطه متناظر با آن در تصویر نیز تغییر می‌کند.

در نهایت، ابزار ترسیم به نام `draw` حذف می‌شود و لیست `binary_representation` به عنوان خروجی تابع برگردانده می‌شود.

برای بهتر و کارآمدتر کردن کد، کارهایی زیر را انجام می‌دهیم. این تغییرات به خوانایی کد و افزایش انعطاف‌پذیری کد کمک می‌کنند:

- استفاده از تاپل برای بازگرداندن ابعاد تصویر
- اضافه کردن یک آرگومان برای فاکتور آستانه به تابع
- استفاده از متغیرهای چندخطی برای اختصاص مقادیر RGB بهتر و خواناتر

کد اصلاح شده:

```
def convertImageToBinary(path, threshold_factor=100):
    image = Image.open(path)
    draw = ImageDraw.Draw(image)
    width, height = image.size
    pix = image.load()
    binary_representation[] =
    for i in range(width):
        for j in range(height):
            red, green, blue = pix[i, j]
            total_intensity = red + green + blue
            if total_intensity > (((255 + threshold_factor) // 2) * 3):
                red, green, blue = 255, 255, 255  # White pixel
                binary_representation.append(۱-)
            else:
                red, green, blue = 0, 0, 0  # Black pixel
                binary_representation.append(۱)
            draw.point((i, j), (red, green, blue))
    del draw
    return binary_representation
```

تابع دوم، با افزودن نویز به آن داده‌ها، داده‌های جدید نویزدار تولید می‌کند

این کد یک تابع به نام `getNoisyBinaryImage` و یک تابع به نام `generateNoisyImages` دارد که با استفاده از کتابخانه PIL (Python Imaging Library) یک تصویر را با افزودن نویز تغییر می‌دهد و تصویرهای نویزی جدید ایجاد می‌کند.

تابع `getNoisyBinaryImage` یک تصویر ورودی را باز می‌کند، نویز به آن اضافه می‌کند و تصویر نویزی جدید را به عنوان خروجی ذخیره می‌کند. عملیات افزودن نویز به این صورت است که برای هر پیکسل در

تصویر، یک مقدار تصادفی از یک بازه خاص برای نویز تولید می‌شود و سپس این مقدار به مقدار RGB هر پیکسل اضافه می‌شود. سپس اطمینان حاصل می‌شود که مقدار RGB در بازه معتبر (۰ تا ۲۵۵) باقی مانده و در نهایت تصویر نویزی به عنوان خروجی ذخیره می‌شود.

تابع `generateNoisyImages` یک لیست از مسیرهای فایل تصویر ایجاد کرده و برای هر تصویر این لیست، یک تصویر نویزی جدید ایجاد می‌کند و آن را به عنوان یک فایل جدید ذخیره می‌کند. نام فایل‌های تصویر نویزی با افزایش یک شماره از ۱ شروع می‌شود (برای مثال، noisy1.jpg، noisy2.jpg و غیره).

برای بهتر و کارآمدتر کردن کد، کارهایی زیر را انجام می‌دهیم. این تغییرات باعث افزایش خوانایی کد و بهبود قابلیت‌ها می‌شود:

- جداسازی توابع
- استفاده از getpixel و putpixel به جای load و point برای دسترسی به پیکسل‌ها
- افزودن مقدار پیش‌فرض به noise_factor در تابع get_noisy_imag

کد اصلاح شده:

```
def add_noise_to_pixel(pixel, noise_factor):
    return tuple(max(0, min(255, value + random.randint(-noise_factor, noise_factor))) for value in pixel)

def add_noise_to_image(image, noise_factor):
    width, height = image.size
    noisy_image = Image.new("RGB", (width, height))

    for i in range(width):
        for j in range(height):
            pixel = add_noise_to_pixel(image.getpixel((i, j)), noise_factor)
            noisy_image.putpixel((i, j), pixel)
    return noisy_image

def get_noisy_image(input_path, output_path, noise_factor=10000000):
    image = Image.open(input_path)
    noisy_image = add_noise_to_image(image, noise_factor)
    noisy_image.save(output_path, "JPEG")

def generate_noisy_images():
    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"
        get_noisy_image(image_path, noisy_image_path)
        print(f"Noisy image for {image_path} generated and saved as {noisy_image_path}")
```

```
generate_noisy_images()
```
سپس در مرحله آخر این تصاویر باینری شده و نویز گرفته را در پارامترهایی به صورت زیر ذخیره می‌کنیم:

```
x1 = convertImageToBinary("/content/1.jpg")
x2 = convertImageToBinary("/content/2.jpg")
x3 = convertImageToBinary("/content/3.jpg")
x4 = convertImageToBinary("/content/4.jpg")
x5 = convertImageToBinary("/content/5.jpg")
p1 = convertImageToBinary("/content/noisy1.jpg")
p2 = convertImageToBinary("/content/noisy2.jpg")
p3 = convertImageToBinary("/content/noisy3.jpg")
p4 = convertImageToBinary("/content/noisy4.jpg")
p5 = convertImageToBinary("/content/noisy5.jpg")
```

۲. طراحی یک شبکه عصبی همینگ که با اعمال ورودی دارای میزان مشخصی نویز برای هریک از داده‌ها، خروجی متناسب با آن داده نویزی را بیابد:

این کد یک شبکه Hamming را نمایش می‌دهد که با استفاده از فاصله Hamming، الگوی نزدیک‌ترین الگو به الگوی ورودی را پیدا می‌کند. شبکه Hamming به طور خاص برای تشخیص و بازیابی الگوهایی که با نویز از دست رفته‌اند یا تغییر کرده‌اند، مورد استفاده قرار می‌گیرد.

```
class HammingNetwork:
```
در این تابع، الگوهای اولیه با نام‌های مربوطه به عنوان ورودی گرفته می‌شوند و در دیکشنری `self.patterns` ذخیره می‌شوند:

```
  def __init__(self, patterns_with_names):
    self.patterns = {name: pattern for name, pattern in patterns_with_names}
```
این تابع خالی است چرا که شبکه Hamming به طور معمول آموزش نمی‌بیند. الگوها در ابتدا به عنوان الگوهای ذخیره‌شده وارد شبکه می‌شوند:

```
  def train(self):
    pass  # Training not needed as the Hamming distance doesn't require training
```
این تابع با گرفتن یک الگو به عنوان ورودی، همان الگو را به عنوان خروجی برمی‌گرداند. این عملیات بدون نیاز به آموزش انجام می‌شود:

```
  def recall(self, input_pattern, max_iterations=100):
    return input_pattern
```
این تابع فاصله Hamming بین دو الگو را محاسبه می‌کند:

```
  def hamming_distance(self, pattern1, pattern2):
    distance = sum(bit1 != bit2 for bit1, bit2 in zip(pattern1, pattern2))
    return distance
```
با گرفتن الگوی بازیابی شده، این تابع نزدیک‌ترین الگو را از میان الگوهای ذخیره‌شده با استفاده از فاصله Hamming پیدا می‌کند و نام آن را برمی‌گرداند:

```
  def find_closest_match(self, recalled_pattern):
    min_distance = float('inf')
    closest_pattern_name = None
    for pattern_name, pattern in self.patterns.items:()
      distance = self.hamming_distance(recalled_pattern, pattern)
```

```
            print(f"Distance from {pattern_name}: {distance}")  # Print distance for each pattern
            if distance < min_distance:
                min_distance = distance
                closest_pattern_name = pattern_name
        return closest_pattern_name
```

در انتها، یک نمونه از استفاده از این کد نشان داده شده است. الگوهای مختلف با نام‌های مختلف ایجاد شده‌اند. سپس یک الگو با نویز به عنوان ورودی در نظر گرفته شده و با استفاده از شبکه Hamming، الگوی نزدیک‌ترین را بازیابی می‌کند:

```
if __name__ == "__main:"__
    patterns_with_names] =
")        pattern1", x1,(
")        pattern2", x2,(
")        pattern3", x3,(
")        pattern4", x4,(
")        pattern5", x5(
[
    patterns_with_names = [(name, ".join(map(str, pattern))) for name, pattern in patterns_with_names]
    hamming_net = HammingNetwork(patterns_with_names)
    noisy_pattern = p3
    recalled_pattern = ".join(map(str, noisy_pattern))
    print("Noisy Pattern:", noisy_pattern)
    print("Recalled Pattern:", recalled_pattern)
    closest_match_name = hamming_net.find_closest_match(recalled_pattern)
    if closest_match_name is not None:
        print("Closest Match Found:", closest_match_name)
    else:
        print("No close match found with any stored pattern.")
```

با تغییر noise_factor میتوان مقدار نویز را تغییر کرد.

نویز را روی ۱۰۰ قرار می‌دهیم و روی هر ۵ داده تست می‌کنیم که درست پیش‌بینی می‌کند: (مثال p2)

```
    noisy_pattern = p2
    recalled_pattern = ''.join(
    print("Noisy Pattern:", noi
    print("Recalled Pattern:",

    # Find the closest match to
    closest_match_name = hammin

    if closest_match_name is no
        print("Closest Match Fo
    else:
        print("No close match f
```

```
Noisy Pattern: [0, 0, 0, 0, 0,
Recalled Pattern: 0000000000000
Distance from pattern1: 1147
Distance from pattern2: 206
Distance from pattern3: 1207
Distance from pattern4: 954
Distance from pattern5: 1059
Closest Match Found: pattern2
```

نویز را روی ۵۰۰ قرار می‌دهیم و روی هر ۵ داده تست می‌کنیم و می‌بینیم که درست پیش‌بینی می‌کند.

نویز را روی ۸۰۰ قرار می‌دهیم و روی هر ۵ داده تست می‌کنیم و می‌بینیم که درست پیش‌بینی می‌کند.

نویز را روی ۱۰۰۰ قرار می‌دهیم و روی هر ۵ داده تست می‌کنیم و می‌بینیم که روی ۳تا از داده‌ها پیش‌بینی اشتباه انجام داده است:

```
    noisy_pattern = p5
    recalled_pattern = ''.joir
    print("Noisy Pattern:", nc
    print("Recalled Pattern:",

    # Find the closest match t
    closest_match_name = hammi

    if closest_match_name is r
        print("Closest Match F
    else:
        print("No close match
```

```
oisy Pattern: [1, 0, 1, 1, 0,
ecalled Pattern: 101101101011
istance from pattern1: 5613
istance from pattern2: 5592
istance from pattern3: 5507
istance from pattern4: 5578
istance from pattern5: 5543
losest Match Found: pattern3
```

۳. در این قسمت از داده‌های ورودی، عکس‌های دارای نقطه خالی تولید می‌کنیم

این کد یک پیاده‌سازی از شبکه همینگ را نشان می‌دهد. شبکه همینگ یک نوع شبکه عصبی مصنوعی است که برای الگوریتم همینگ برنامه‌ریزی شده است. این الگوریتم معمولاً برای تشخیص الگوهای مشابه یا گروه‌بندی اطلاعات در داده‌ها استفاده می‌شود.

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

IMAGE_PATH = "/content/noisy3.jpg"
```

نمایش یک ماتریس به صورت فرمت‌داده شده:

```
def show(matrix):
    for j in range(len(matrix)):
        for i in range(len(matrix[0])):
            print("{:3f}".format(matrix[j][i]), end=" ")
        print(sep="")
```

تبدیل یک بردار به یک ماتریس با ابعاد مشخص:

```
def change(vector, a, b):
    matrix = [[0 for j in range(a)] for i in range(b)]
    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
```

ضرب یک ماتریس در یک بردار با استفاده از یک آستانه (threshold) برای تابع فعال‌سازی:

```
def product(matrix, vector, T):
    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
```

تابع فعال‌سازی برای پردازش یک بردار:

```
def action(vector, T, Emax):
    result_vector[] =
    for value in vector:
        if value <= 0:
            result_vector.append(۰)
        elif 0 < value <= T:
            result_vector.append(Emax * value)
        elif value > T:
            result_vector.append(T)
    return result_vector
```

محاسبه مجموع مقادیر بردار با حذف عنصر با شاخص j:

```
def mysum(vector, j):
    p = 0
    total_sum = 0
    while p < len(vector):
        if p != j:
            total_sum = total_sum + vector[p]
        p += 1
    return total_sum
```

محاسبه اختلاف بین دو بردار و محاسبه نرم اقلیدسی این اختلاف:

```
def norm(vector, p):
    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)
```

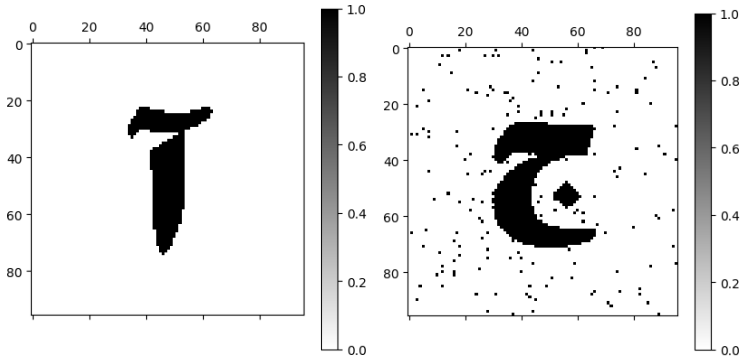
سپس در ادامه کد، از توابع فوق برای پیاده‌سازی شبکه همینگ استفاده می‌شود. این شبکه در نهایت تصویر ورودی را در یک کلاس مرتبط با خروجی بیشترین ارزش به تصویر نهایی تخصیص می‌دهد. اگر ارزش‌ها همگی صفر باشند، شبکه نمی‌تواند ترجیحی میان کلاس‌ها قائل شود. اگر تصویر برچسبی نداشته باشد یا تعداد کمی ویژگی ورودی داشته باشیم، شبکه ممکن است نتواند تصمیمی بگیرد.

```
path] =
/'   content/1.jpg,'
/'   content/2.jpg,'
/'   content/3.jpg,'
/'   content/4.jpg,'
/'   content/5.jpg,'
[

x  [] =
print(os.path.basename(IMAGE_PATH))

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 input image into a matrix
rotated_q = np.rot90(q)
flipped_rotated_q = np.flipud(rotated_q)
plt.matshow(flipped_rotated_q,cmap='binary')
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 = 0.00001  # Maximum allowable difference norm between output vectors in consecutive iterations
U = 1 / Emax

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]))]

while norm(y[i], p) >= Emax:
   s.append([0 for j in range(len(s[0]))])
```

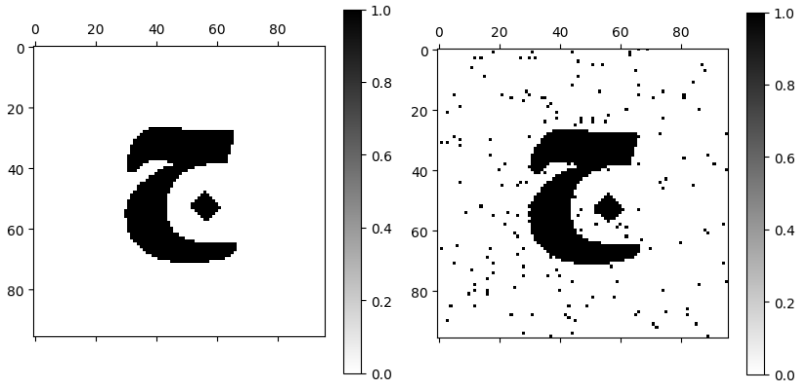
```
    for j in range(len(s[0])):
        s[i + 1][j] = y[i][j] - e * mysum(y[i], j)
    y.append((action(s[i + 1], U, Emax)))
    i += 1
    p = y[i - 1]

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

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)
    rotated_q = np.rot90(q)
    flipped_rotated_q = np.flipud(rotated_q)
    print('The highest positive output value is associated with class', result_index)
    plt.matshow(flipped_rotated_q, cmap='binary')
    plt.colorbar()
    plt.show()
```

نویز را روی ۱۰۰ ثابت نگه داشته و تعداد نقاط خالی را کم و زیاد می‌کنیم.

Emax = 0.1 (تشخیص اشتباه)



Emax = 0.0001 (تشخیص درست)



سوال ۴

وارد کردن کتابخانه‌های مورد نیاز:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import r2_score
import tensorflow as tf
from tensorflow import keras
from keras import preprocessing
from keras.models import Sequential
from keras.layers import Dense
import warnings
warnings.filterwarnings("ignore")
```

۱. خواندن فایل csv:

```
!pip install --no-cache-dir gdown
!gdown 13XaS5G7bp7niH1hFaEIxoiJPzcTryJ7h
df = pd.read_csv('archive/data.csv')
```

فراخوانی تابع info:

```
df.info()
```

```
<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
```

نمایش داده‌های دارای جای خالی در هر ستون:

```
df.isnull().sum()
```

```
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
```

در اینجا جای خالی نداریم ولی اگر داشتیم میتوانستیم با دستور زیر سطرهای دارای جای خالی را حذف کنیم.

```
#df.dropna(inplace=True)
```

۲. رسم ماتریس هم‌بستگی:

ابتدا باید مقداری پیش پردازش انجام شود.

در دیتا مشاهده می کنیم که ۴ ستون زیر به صورت عدد نیستند. لذا باید کاری در مورد آن‌ها انجام داد.

street	city	statezip	country
18810 Densmore Ave N	Shoreline	WA 98133	USA
709 W Blaine St	Seattle	WA 98119	USA
26206-26214 143rd Ave SE	Kent	WA 98042	USA
857 170th Pl NE	Bellevue	WA 98008	USA
9105 170th Ave NE	Redmond	WA 98052	USA

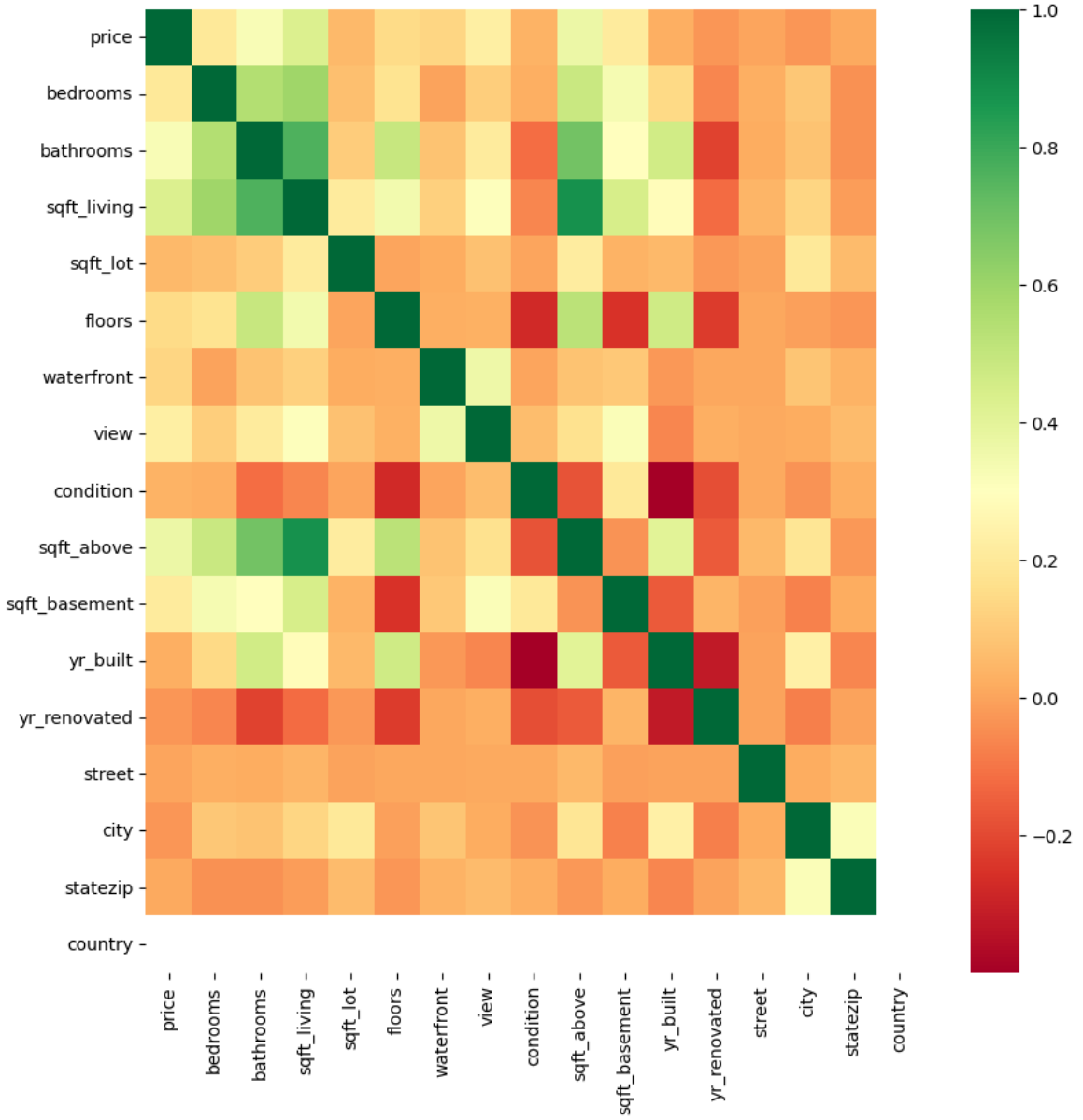
بهترین کار این است که برای مثال به هر شهر یک عدد اختصاص دهیم.

```
df['city'] = pd.factorize(df['city'])[0] + 1
df['street'] = pd.factorize(df['street'])[0] + 1
df['country'] = pd.factorize(df['country'])[0] + 1
df['statezip'] = pd.factorize(df['statezip'])[0] + 1
```

در اخر ماتریس هم بستگی را رسم می‌کنیم:

```
plt.figure(figsize=(10, 10))
sns.heatmap(df.corr(), cmap="RdYlGn")
plt.show()
```

در شکل مشاهده می شود که هرچه به سمت سبزرنگ برویم، هم بستگی بیشتر میشود. پس sqft_living بیشترین عدد است که درواقع مساحت خانه را نشان می دهد.



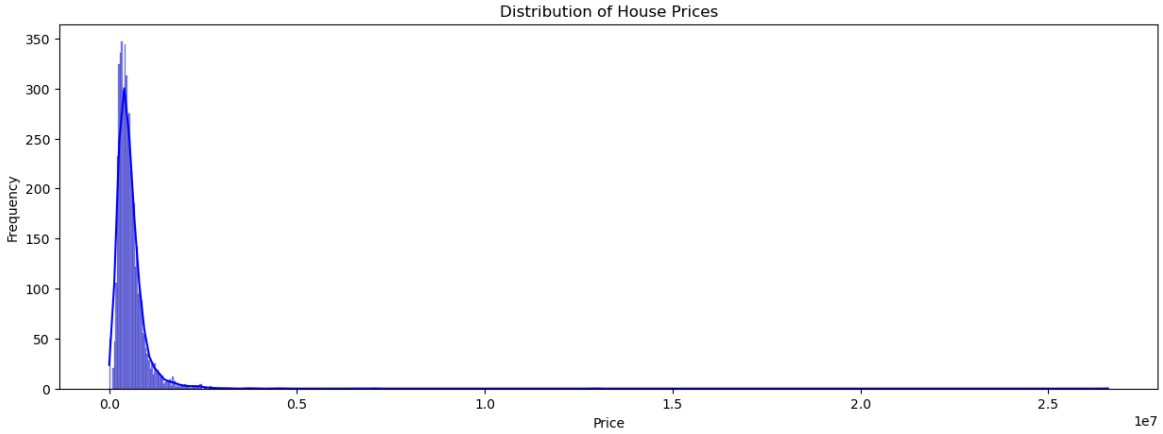
با کد زیر نیز می توان مقادیر هم بستگی را پیدا کرد:

```
df.corr()['price'].sort_values(ascending=False)
```

```
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
statezip         0.014182
street           0.003030
yr_renovated    -0.028774
city            -0.033270
country               NaN
Name: price, dtype: float64
```

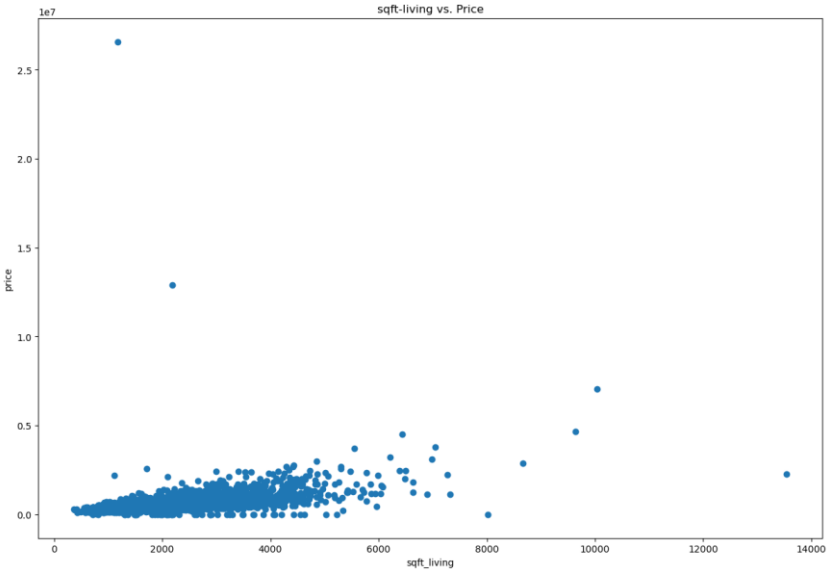
۳. رسم نمودار توزیع قیمت:

```
plt.figure(figsize=(15, 5))
sns.histplot(df['price'], kde=True, color='blue')
plt.title('Distribution of House Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



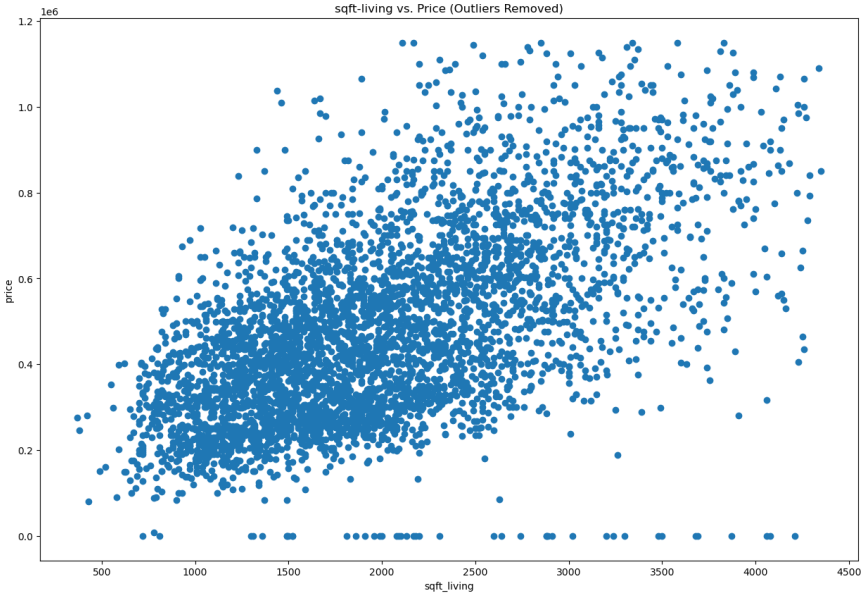
رسم نمودار قیمت و مساحت:

```
plt.figure(figsize=(15, 10))
plt.scatter(x='sqft_living', y='price', data=df)
plt.xlabel('sqft_living')
plt.title('sqft-living vs. Price')
plt.ylabel('price')
plt.show()
```



رسم نمودار با حذف داده های پرت:

```
Q1_sqft = df['sqft_living'].quantile(0.25)
Q3_sqft = df['sqft_living'].quantile(0.75)
IQR_sqft = Q3_sqft - Q1_sqft
Q1_price = df['price'].quantile(0.25)
Q3_price = df['price'].quantile(0.75)
IQR_price = Q3_price - Q1_price
lower_bound_sqft = Q1_sqft - 1.5 * IQR_sqft
upper_bound_sqft = Q3_sqft + 1.5 * IQR_sqft
lower_bound_price = Q1_price - 1.5 * IQR_price
upper_bound_price = Q3_price + 1.5 * IQR_price
filtered_df = df[(df['sqft_living'] >= lower_bound_sqft) & (df['sqft_living'] <= upper_bound_sqft) &
                 (df['price'] >= lower_bound_price) & (df['price'] <= upper_bound_price)]
plt.figure(figsize=(15, 10))
plt.scatter(x='sqft_living', y='price', data=filtered_df)
plt.xlabel('sqft_living')
plt.title('sqft-living vs. Price (Outliers Removed)')
plt.ylabel('price')
plt.show()
```



۴. تبدیل ستون دیت به دو ستون ماه و سال و حذف این ستون از دیتافریم:

اولین خط تیره ستون تاریخ را به ویرگول تبدیل می کنیم تا از اولین خط تیره قابل تبعیض باشد

```
df['date'] = df['date'].str.replace('-' , ',' , 1)
```

سال هر تاریخ، از شکسته شدن تاریخ به محض رسیدن به ویرگول تشکیل می شود:

```
# year
def a(date):
    if ',' in date:
        return date.split(',')[0].strip()
    else:
        return 'unknown'
```

ماه هر تاریخ، از شکسته شدن تاریخ به محض رسیدن به خط تیره و حد فاصل آن با ویرگول تشکیل می‌شود:

```
# month
def b(date):
    if ',' in date:
        return date.split(',')[1].split('-')[0].strip()
    else:
        return 'unknown'
```

اعمال توابع به ستون تاریخ در دیتا و حذف ستون اصلی:

```
df['year'] = df['date'].apply(a)
df['month'] = df['date'].apply(b)
df.drop(['date'], axis=1, inplace=True)
```

year	month
2014	05
2014	05
2014	05
2014	05
2014	05
...	...
2014	07
2014	07
2014	07
2014	07
2014	07

۵. تقسیم به آموزش و تست:

```
x = df.drop(["price"], axis=1)
Y = df["price"]
x_train, x_test, y_train, y_test = train_test_split(x, Y, test_size=0.2, random_state=7)
```

اسکیل کردن (داده های ترین و تست را جداگانه اسکیل می کنیم تا در داده های اموزش هیچ اطلاعاتی حتی به اندازه مینیمم و ماکزیمم از داده های تست وجود نداشته باشد):

```
scaler = MinMaxScaler()
x_train_scaled=scaler.fit_transform(x_train)
x_test_scaled=scaler.transform(x_test)
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)
```

```
y_train = scaler.fit_transform(y_train)
y_test = scaler.transform(y_test)
```

۶. طراحی یک مدل mlp ساده با ۲ لایه پنهان که لایه اول ۲۰ نورون و لایه دوم ۳۰ نورون دارد.

```
model = Sequential()
model.add(Dense(20, activation='relu', input_shape=(x_train_scaled.shape[1],)))
model.add(Dense(30, activation='relu'))
model.add(Dense(1, activation='linear'))
```

در مرحله بعد، بهینه ساز را آدام و تابع اتلاف را mean square error انتخاب می‌کنیم

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

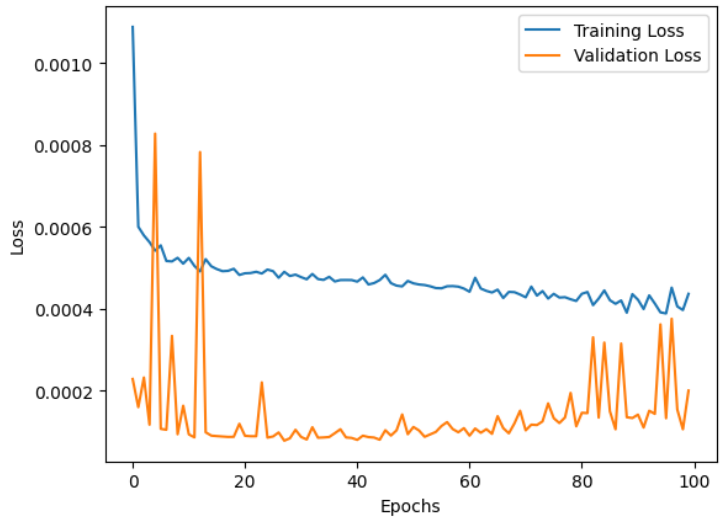
سپس مقداری از داده های اموزش را برای ولید کردن قرار می دهیم

```
history = model.fit(x_train_scaled, y_train, validation_split=0.2, epochs=100 ,batch_size=10, verbose=0)
```

نمودار تابع اتلاف:

```
loss = model.evaluate(x_test_scaled , y_test_scaled)
print(f'R^2 Score: {loss}')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

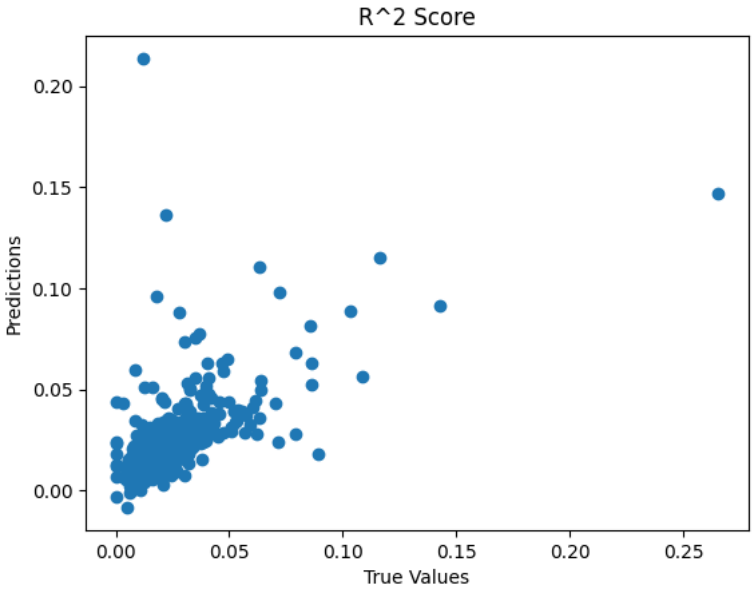
```
29/29 [==============================] - 0s 2ms/step - loss: 1.7426e-04
R^2 Score: 0.0001742627500789 24
```



نمودار r2 score:

```
y_pred = model.predict(x_test_scaled)
rscore = r2_score(y_test_scaled , y_pred)
print(f'R^2 Score: {rscore}')
plt.scatter(y_test_scaled, y_pred)
plt.title('R^2 Score')
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.show()
```

```
29/29 [==============================] - 0s 2ms/step
R^2 Score: 0.30620705087421807
```



۷. انجام همان فرایند سوال قبل با بهینه ساز و تابع اتلاف جدید:

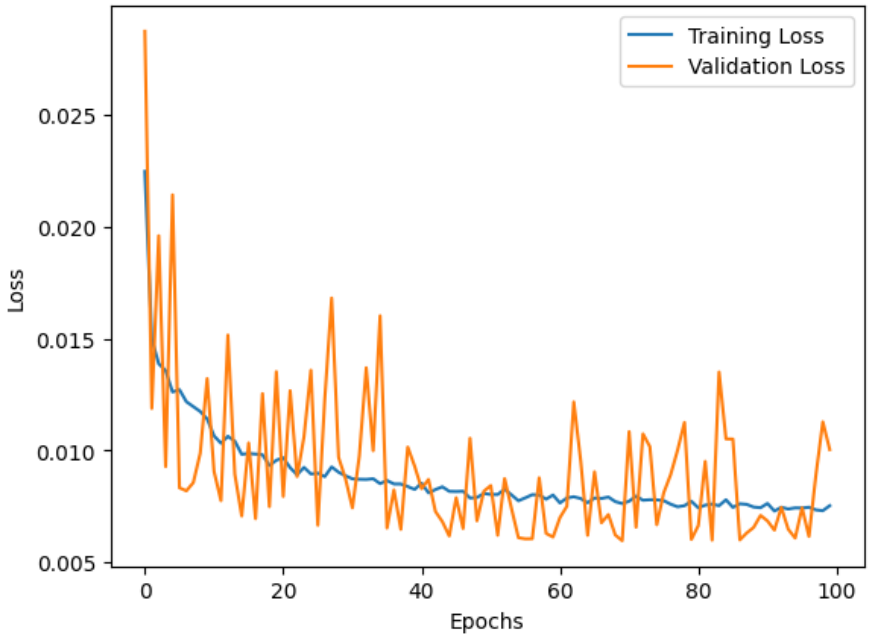
این بار بهینه ساز را sgd و تابع اتلاف را mean absolute error قرار می دهیم:

```
model_2.compile(optimizer = 'sgd',loss = 'mae')
```

کدهای نمودار تابع اتلاف و r2score به همان شکل قبل است با این تفاوت که از model_2 استفاده می شود.

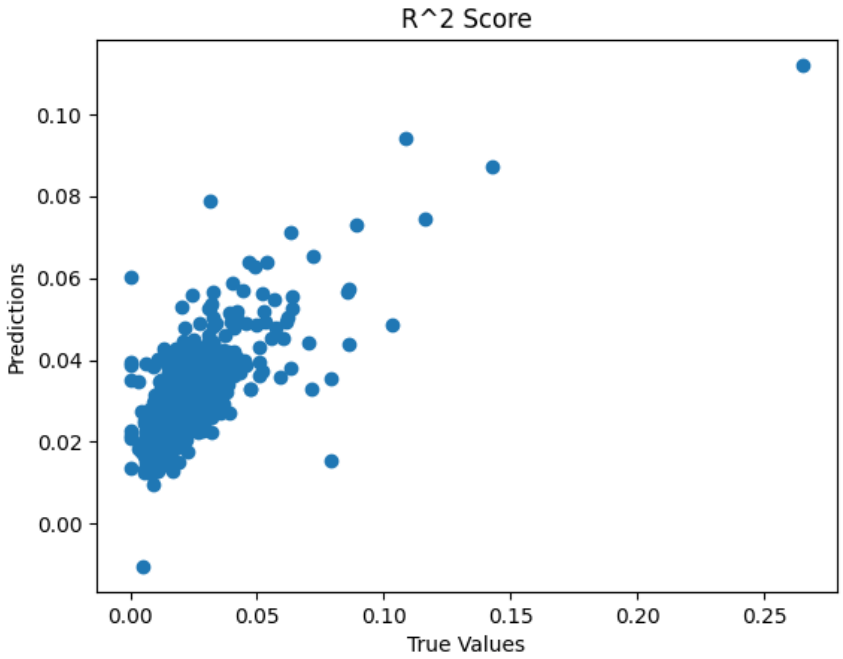
تابع اتلاف: (افزایش پیدا کرده)

```
29/29 [==============================] - 0s 3ms/step - loss: 0.0102
R^2 Score: 0.010246527381241322
```



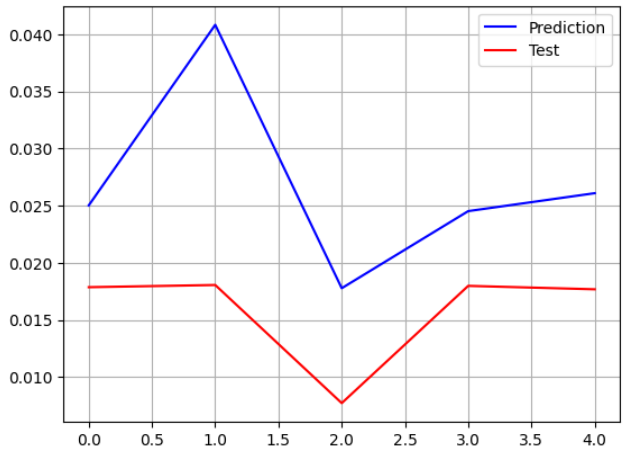
نمودار r2score: (کاهش پیدا کرده)

```
29/29 [==============================] - 0s 2ms/step
R^2 Score: 0.2887858055444549
```



۸. انتخاب ۵ داده به صورت تصادفی از مجموعه ارزیابی و نشان دادن قیمت پیش بینی شده و قیمت واقعی:

```
import random
random_pred = list()
random_test = list()
for i in range:(۵)
    j = random.randint(0, len(y_pred) - 1)  # Generate a random index
    random_pred.append(y_pred[j])  # Append y_pred_2 value at the random index j
    random_test.append(y_test_scaled[j])  # Append y_test value at the same random index j
plt.plot(random_pred, 'b', label='Prediction')  # Blue line for predictions
plt.plot(random_test, 'r', label='Test')        # Red line for actual test outputs
plt.legend()
plt.grid()
plt.show()
```



محاسبه تفاوت قیمتها:

```
for i in range:(۵)
   dis = random_pred[i] - random_test[i]
   print(dis)
```

```
[0.00716125]
[0.02278295]
[0.01006068]
[0.00654471]
[0.00841493]
```

چون داده ها اسکیل شده بودند، آن ها را باید از حالت ترنسفورم دراورد تا تفاوت دقیق قیمت مشخص شود:

```
random_pred2 = scaler.inverse_transform(random_pred)
random_test2 = scaler.inverse_transform(random_test)
```

```
[190417.54821539]
[605798.70305955]
[267513.55642453]
[174023.90804887]
[223753.10454518]
```

این عملکرد مناسب نیست و تفاوت نسبتا زیادی بین قیمت پیش بینی شده و قیمت اصلی وجود دارد.
میتوان برای بهبود عملکرد، در ساخت مدل از gridsearch استفاده کرد. به این صورت که برای پارامترهای مدل چند پارامتر مد نظر قرار داده می شود و با همه انها مدلهای مختلف ساخته شده و نهایتا مدلی که بهترین عملکرد را روی دیتا داشته باشد، انتخاب می شود.

سوال ۵

۱. استفاده از دیتای iris و پیش پردازش آن:

لود کردن دیتا:

```
iris = datasets.load_iris()
```

بررسی سایز دیتا:

```
iris.data.shape
```

```
(150, 4)
```

مشاهده نام ستون ها یا نام ویژگی ها:

```
iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

بررسی نام خروجی یا تارگت ها:

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

نگاه کلی به دیتا:

```
iris.data
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

نگاه کلی به خروجی یا تارگت:

```
iris.target
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

وارد کردن دیتا به دیتافریم و مشاهده ۵ سطر اول:

```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df.head()
```

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

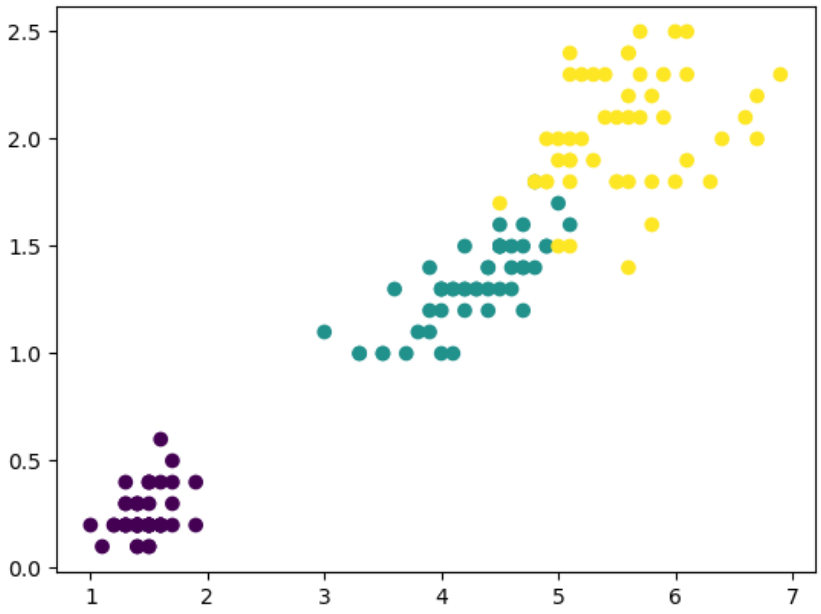
افزودن تارگت به دیتافریم:

```
df['class']= iris.target
df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class
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
...	...	...	...	...	...

نمایش تارگت ها به صورت نمودار طبق دو ویژگی ستون دوم و سوم:

```
x=iris.data[:,[2,3]]
y=iris.target
import matplotlib.pyplot as plt
plt.scatter(x[:,0],x[:,1],c=y)
```



۲. مدلسازی و ارزیابی نتایج با حداقل ۴ شاخص و ماتریس درهم ریختگی

ابتدا ورودی و تارگت را از هم جدا و مشخص می کنیم و سپس با قرار دادن ۲۰٪ داده ها برای تست، مدلسازی را شروع می کنیم.

```
iris = datasets.load_iris()
X = iris.data[:, 0:4]
y = iris.target
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=93)
```

- مدل رگرسیون لجستیک:

```
model = LogisticRegression(C=1e5, solver='lbfgs', multi_class='multinomial')
model.fit(x_train, y_train)
```

پیش بینی خروجی داده های تست:

```
ypred = model.predict(x_test)
```

بررسی درصد تشخیص درستی برای داده های تست و آموزش:

```
a = model.score(x_train, y_train)
b = model.score(x_test, y_test)
[a, b]
```

```
[0.95, 1.0]
```

بررسی خطا:

```
a = met.mean_squared_error(y_test, ypred)     # MSE
b = mean_absolute_error(y_test, ypred)        # MAE
c = sqrt(mean_squared_error(y_test, ypred))   # SMSE
[a, b, c]
```

```
[0.0, 0.0, 0.0]
```

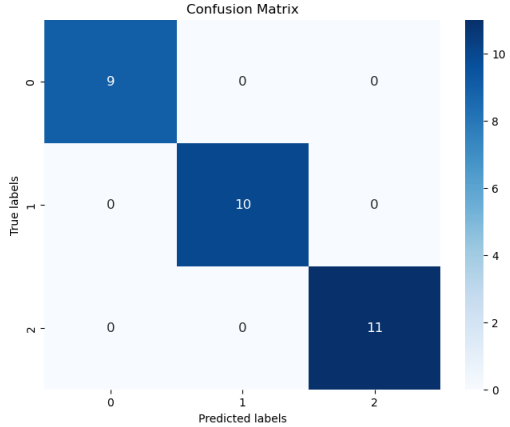
بدست آوردن ماتریس در هم ریختگی:

```
confusion_matrix(y_test, ypred)
```

```
array([[ 9,  0,  0],
       [ 0, 10,  0],
       [ 0,  0, 11]], dtype=int64)
```

رسم ماتریس در هم ریختگی:

```
cf_matrix = confusion_matrix(y_test, ypred)
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 12})
plt.gca().set_ylim(len(np.unique(y_test)), 0)  # Fix for matplotlib 3.1.1 and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
```



بررسی گزارش کلی و محاسبه ۴ معیار بررسی:

```
print(classification_report(y_test, ypred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

مدل به خوبی عمل کرده است.

- مدل MLP:

```
model = MLPClassifier(hidden_layer_sizes=(8) , max_iter=110 , alpha=1e-4, solver='sgd', random_state=93
                      ,verbose=True , learning_rate_init=.1 )
```

داده ها روی مدل جدید فیت شده، کدها به همان صورت قبلیست و فقط خروجی نمایش داده می شود:

بررسی درصد تشخیص درستی برای داده های تست و آموزش:

```
[a, b]
 [0.875, 0.8]
```

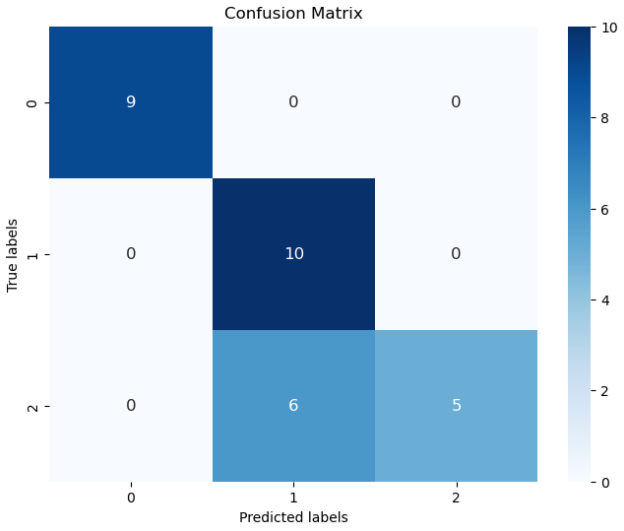
بررسی خطا:

```
[a, b, c]
 [0.2, 0.2, 0.4472135954999579]
```

بدست آوردن ماتریس در هم ریختگی:

```
array([[ 9,  0,  0],
       [ 0, 10,  0],
       [ 0,  6,  5]], dtype=int64)
```

رسم ماتریس در هم ریختگی:



بررسی گزارش کلی و محاسبه ۴ معیار بررسی:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9
1	0.62	1.00	0.77	10
2	1.00	0.45	0.62	11
accuracy			0.80	30
macro avg	0.88	0.82	0.80	30
weighted avg	0.88	0.80	0.79	30

خطا به نسبت مدل قبل بیشتر شده ولی این مدل هم تقریبا به خوبی عمل کرده است

- مدل RBF:

```
model = svm.SVC(kernel='rbf', gamma=0.7, C=1)
```

داده ها روی مدل جدید فیت شده، کدها به همان صورت قبلیست و فقط خروجی نمایش داده می شود:

بررسی درصد تشخیص درستی برای داده های تست و آموزش:

[a, b]

```
[0.95, 1.0]
```

بررسی خطا:

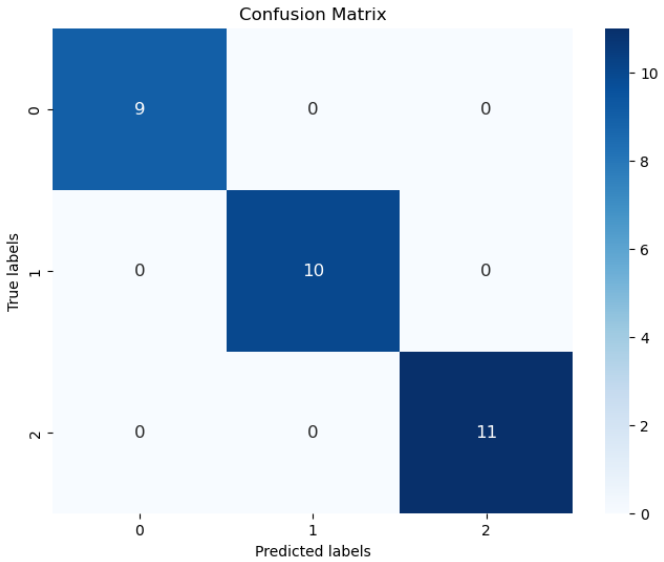
[a, b, c]

```
[0.0, 0.0, 0.0]
```

بدست آوردن ماتریس در هم ریختگی:

```
array([[ 9,  0,  0],
       [ 0, 10,  0],
       [ 0,  0, 11]], dtype=int64)
```

رسم ماتریس در هم ریختگی:



بررسی گزارش کلی و محاسبه ۴ معیار بررسی:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9
1	0.62	1.00	0.77	10
2	1.00	0.45	0.62	11
accuracy			0.80	30
macro avg	0.88	0.82	0.80	30
weighted avg	0.88	0.80	0.79	30

این مدل دقیقا به اندازه مدل اول به خوبی عمل می‌کند.

- حل سوال بدون استفاده از کتابخانه‌ها و مدل‌های آماده پایتون:
- مدل Logistic Regression:

این کد یک مثال از الگوریتم رگرسیون لجستیک را برای مسئله‌ی دسته‌بندی دو کلاسه اجرا می‌کند. در اینجا، دیتاست Iris برای دسته‌بندی گل‌های زنبق به دو گونه مختلف (Iris-Versicolor و دیگران) استفاده شده است. ما هدف داریم تا با استفاده از رگرسیون لجستیک، احتمال تعلق هر نمونه به گونه‌ی Iris-Versicolor را پیش‌بینی کنیم.

بارگذاری دیتاست: ابتدا دیتاست Iris بارگذاری شده و برای دسته‌بندی به یک مسئله دو کلاسه تبدیل می‌شود.

```
iris = load_iris()
X = iris.data
```

افزودن intercept به ماتریس ویژگی‌ها (X): یک ستون متغیر برای عبور از اندازه‌ی بایاس (intercept) به ماتریس ویژگی‌ها اضافه می‌شود.

```
y = (iris.target != 0).astype(int)  # Convert to binary classification problem (1 for Iris-Versicolor, 0 for others)
X = np.c_[np.ones((X.shape[0], 1)), X]
```

تعریف توابع مورد نیاز: توابع سیگموید، فرضیه رگرسیون لجستیک، تابع هزینه و تابع گرادیان نوشته شده‌اند.

تابع سیگموئید: $\frac{1}{1+e^{-z}}\sigma(z)$

فرضیه رگرسیون لجستیک: $\sigma(\theta^T X)h_\theta(X)$

تابع هزینه: $J(\theta) = -\frac{1}{m}\sum[y\log(h) + (1-y)\log(1-h)]$

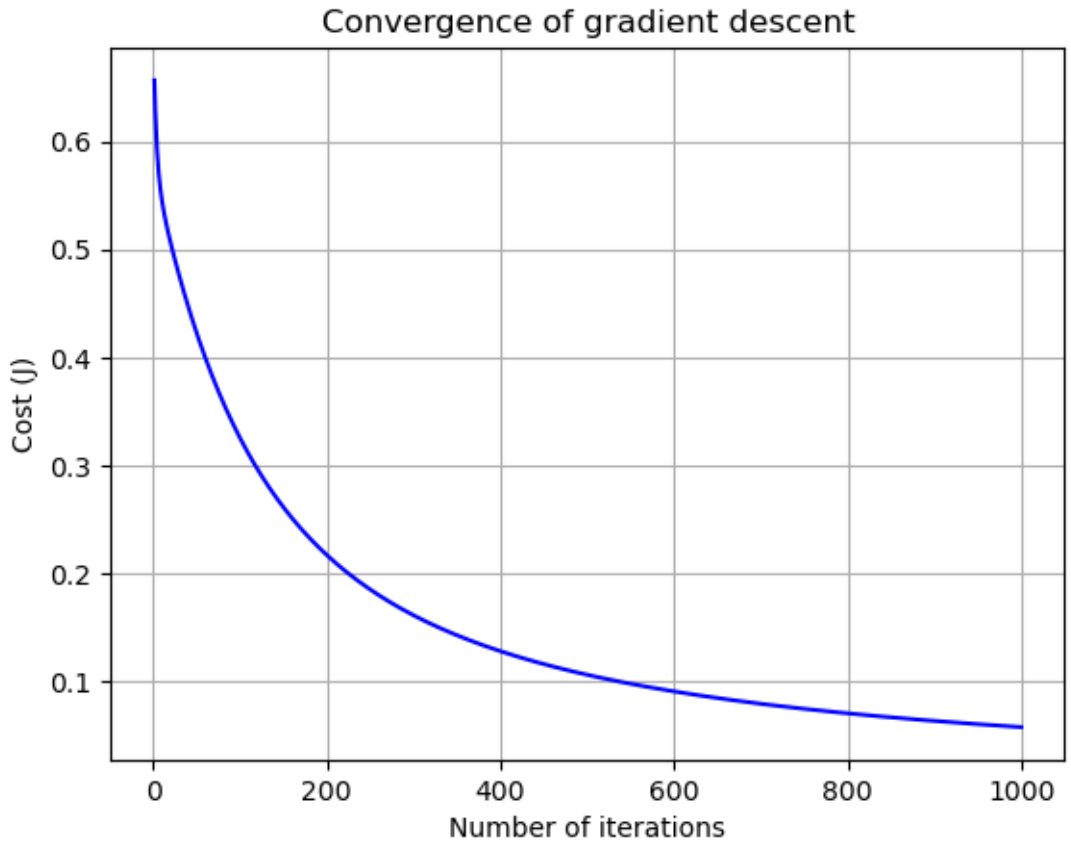
تابع گرادیان: $\frac{1}{m}X^T(h - \nabla J(\theta)$

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
def hypothesis(theta, X):
    return sigmoid(np.dot(X, theta))
def cost_function(theta, X, y):
    m = len(y)
    h = hypothesis(theta, X)
    return (-1/m) * np.sum(y * np.log(h) + (1 - y) * np.log(1 - h))
def gradient_descent(theta, X, y, learning_rate, iterations):
    m = len(y)
    cost_history = np.zeros(iterations)
    for i in range(iterations):
        h = hypothesis(theta, X)
        gradient = (1/m) * np.dot(X.T, (h - y))
        theta -= learning_rate * gradient
        cost_history[i] = cost_function(theta, X, y)
    return theta, cost_history
```

اجرای گرادیان کاهشی: الگوریتم گرادیان کاهشی برای بهینه‌سازی پارامترها اجرا می‌شود. همچنین، تاریخچه هزینه نیز ذخیره می‌شود تا بتوانیم پیشرفت هزینه را در طول زمان ببینیم.

```
theta = np.zeros(X.shape[1])
learning_rate = 0.01
iterations = 1000
theta, cost_history = gradient_descent(theta, X, y, learning_rate, iterations)
```

نمودار کاهش هزینه: یک نمودار از تاریخچه هزینه رسم می‌شود تا نشان دهد که الگوریتم بهینه‌سازی به درستی کار می‌کند و هزینه کاهش پیدا کرده است.



```
plt.plot(range(1, iterations + 1), cost_history, color='blue')
plt.rcParams["figure.figsize"] = (10,6)
plt.grid()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent')
plt.show()
```

نمایش پارامترهای نهایی و دقت: پارامترهای نهایی مدل چاپ شده و دقت مدل بر اساس پیش‌بینی‌ها محاسبه و نمایش داده می‌شود.

```
print('Theta:', theta)
```

```
Theta: [-0.17413711 -0.26714577 -0.91860382  1.46965364  0.66477688]
```

پیش‌بینی: با استفاده از پارامترهای بهینه‌سازی شده، احتمال تعلق هر نمونه به گونه‌ی Iris-Versicolor محاسبه و دسته‌بندی می‌شود.

```
predictions = hypothesis(theta, X)
predicted_labels = (predictions >= 0.5).astype(int)
print('Accuracy:', np.mean(predicted_labels == y))
```

```
Accuracy: 1.0
```

- مدل MLP:

این کد یک شبکه عصبی چند لایه با ورودی‌ها، یک لایه مخفی و یک لایه خروجی است که برای مسئله دسته‌بندی گل‌های Iris آموزش داده شده است. در زیر توضیحاتی در مورد هر بخش از کد آورده شده است:

- از دیتاست Iris استفاده می‌شود که درون ماژول `load_iris` اسکایت‌لرن قرار دارد.

```
iris = load_iris()
X, y = iris.data, iris.target
```

- داده‌ها به دو قسمت آموزش و آزمون تقسیم می‌شوند.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- ویژگی‌ها با استفاده از `StandardScaler` به شکل استاندارد (میانگین صفر و واریانس یک) تغییر می‌کنند.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- برچسب‌ها به شکل یک-هات انکد می‌شوند تا بتوانند به عنوان خروجی شبکه عصبی استفاده شوند.

```
y_train_one_hot = np.eye(3)[y_train]  # 3 classes in the Iris dataset
y_test_one_hot = np.eye(3)[y_test]
```

تعریف معماری شبکه عصبی

- یک لایه ورودی با تعداد ویژگی‌های ورودی.
- یک لایه مخفی با تابع فعال‌سازی sigmoid.
- یک لایه خروجی با تابع فعال‌سازی softmax.

```
input_size = X_train.shape[1]
hidden_size = 10
output_size = 3
learning_rate = 0.01
epochs = 1000
```

- وزن‌ها و اساسنامه‌ها به شکل تصادفی مقداردهی اولیه می‌شوند.

```
np.random.seed(42)
weights_input_hidden = np.random.randn(input_size, hidden_size)
biases_input_hidden = np.zeros((1, hidden_size))
weights_hidden_output = np.random.randn(hidden_size, output_size)
biases_hidden_output = np.zeros((1, output_size))
```

- توابع sigmoid و softmax تعریف شده‌اند.

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)
```

- از روش ارتباط پیش به عقب (backpropagation) برای به‌روزرسانی وزن‌ها و اساسنامه‌ها با استفاده از گرادیان کاهشی استفاده می‌شود.

- تابع خطا به عنوان خطاهای آموزشی انتروپی متقاطع (cross-entropy) محاسبه می‌شود.

- مدل به مدت تعداد دوره‌های آموزش (epochs) آموزش داده می‌شود.

```
for epoch in range(epochs):
    hidden_output = sigmoid(np.dot(X_train, weights_input_hidden) + biases_input_hidden)
    output = softmax(np.dot(hidden_output, weights_hidden_output) + biases_hidden_output)
    loss = -np.sum(y_train_one_hot * np.log(output)) / len(X_train)
    output_error = output - y_train_one_hot
    hidden_error = np.dot(output_error, weights_hidden_output.T) * hidden_output * (1 - hidden_output)
    weights_hidden_output -= learning_rate * np.dot(hidden_output.T, output_error) / len(X_train)
    biases_hidden_output -= learning_rate * np.sum(output_error, axis=0, keepdims=True) / len(X_train)
    weights_input_hidden -= learning_rate * np.dot(X_train.T, hidden_error) / len(X_train)
    biases_input_hidden -= learning_rate * np.sum(hidden_error, axis=0, keepdims=True) / len(X_train)
    if epoch % 100 == 0:
        print(f'Epoch {epoch}, Loss: {loss}')
```

```
Epoch 0, Loss: 1.22717408717471
Epoch 100, Loss: 0.7881132458546621
Epoch 200, Loss: 0.6379310024528893
Epoch 300, Loss: 0.5665679279175809
Epoch 400, Loss: 0.5205483894576426
Epoch 500, Loss: 0.48709691497281954
Epoch 600, Loss: 0.4612454753675664
Epoch 700, Loss: 0.44042080676271206
Epoch 800, Loss: 0.4231163632962177
Epoch 900, Loss: 0.4083843376874954
```

- مدل روی داده‌های آزمون ارزیابی می‌شود و دقت آن محاسبه می‌شود.

```
hidden_output_test = sigmoid(np.dot(X_test, weights_input_hidden) + biases_input_hidden)
output_test = softmax(np.dot(hidden_output_test, weights_hidden_output) + biases_hidden_output)
predicted_labels = np.argmax(output_test, axis=1)
```

توجه داشته باشید که این کد در حالت عملیاتی قرار دارد و فرض می‌شود که توابعی مانند ``load_iris`، `train_test_split`، ``StandardScaler`، و `accuracy_score` در دسترس هستند. اگر این توابع در کد شما وجود ندارند، شما باید این توابع را از ماژول‌های مربوطه وارد کنید یا پیاده‌سازی کنید.

```
accuracy = accuracy_score(y_test, predicted_labels)
print(f'Test Accuracy: {accuracy}')
```

```
Test Accuracy: 0.9333333333333333
```

- مدل RBF:

این کد یک مدل شبکه عصبی با تابع هسته Radial Basis Function (RBF) برای دسته‌بندی داده‌های مجموعه داده Iris پیاده‌سازی کرده است.

- از مجموعه داده Iris استفاده شده است.

- داده‌ها به دو بخش آموزش و آزمون تقسیم شده‌اند.

```
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- تابع `rbf_kernel` یک تابع هسته RBF با پارامترهای دو بردار و پارامتر sigma را اجرا می‌کند

```
def rbf_kernel(x1, x2, sigma=1.0):
    return np.exp(-np.linalg.norm(x1 - x2)**2 / (2 * sigma**2))
```

- کلاس `RBFModel` یک مدل با تعداد مشخصی از مراکز RBF و پارامتر sigma را تعریف می‌کند.

```
class RBFModel:
    def __init__(self, num_centers, sigma=1.0):
        self.num_centers = num_centers
        self.sigma = sigma
        self.centers = None
        self.weights = None
```

- متد `fit` برای آموزش مدل با استفاده از ماتریس طراحی (design matrix) اجرا می‌شود

```
    def fit(self, X, y):
        self.centers = X[np.random.choice(X.shape[0], self.num_centers, replace=False)]
        design_matrix = np.zeros((X.shape[0], self.num_centers))
        for i in range(X.shape[0]):
            for j in range(self.num_centers):
                design_matrix[i, j] = rbf_kernel(X[i], self.centers[j], self.sigma)
        self.weights = np.linalg.pinv(design_matrix).dot(y)
```

- متد `predict` برای پیش‌بینی خروجی با استفاده از وزن‌های آموزش دیده اجرا می‌شود

```
    def predict(self, X):
        predictions = np.zeros(X.shape[0])
        for i in range(X.shape[0]):
            for j in range(self.num_centers):
                predictions[i] += self.weights[j] * rbf_kernel(X[i], self.centers[j], self.sigma)
        return predictions
```

- یک مدل با تعداد مشخصی از مراکز RBF آموزش داده شده است.

```
num_centers = 10  # You can adjust the number of RBF centers
sigma = 1.0       # You can adjust the width of the RBF kernel
rbf_model = RBFModel(num_centers=num_centers, sigma=sigma)
rbf_model.fit(X_train, y_train)
```

- دقت مدل بر روی داده‌های آزمون اندازه‌گیری شده و چاپ می‌شود.

```
y_pred = rbf_model.predict(X_test)
accuracy = np.mean(y_pred == y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.0
```

- در نهایت، مرزهای تصمیم مدل بر روی داده‌های آموزش و آزمون تصویرسازی شده‌اند.

```
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis', marker='o', label='Training Data')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis', marker='x', label='True Test Labels')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', marker='s', label='Predicted Test Labels')
plt.title('RBF Model Decision Boundaries')
plt.legend()
```

```
plt.show()
```

توجه داشته باشید که تصویرسازی مرزهای تصمیم فقط برای دو ویژگی اول داده‌ها انجام شده است (X_train[:, ۰] و X_train[:, 1]) و این تصویرسازی به سه دسته اصلی از مجموعه داده Iris اشاره دارد.