دانشگاه صنعتی خواجه نصیرالدین طوسی
دانشکدهٔ مهندسی برق - گروه مهندسی کنترل

درس مبانی سیستم‌های هوشمند

پاسخ مینی پروژه سری دوم

نام و نام خانوادگی	فریما ایران‌خبش
شمارهٔ دانشجویی	۹۸۱۹۸۸۳
تاریخ	آذرماه ۱۴۰۲

فهرست مطالب

فهرست تصاویر

فهرست جداول

فهرست برنامه‌ها

۱ مجموعه دادهٔ 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.

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

ابتدا با دستور gdown این دیتا فریم را فراخوانی می‌کنیم و سپس ستون اول و دوم آن را به ویژگی‌ها و ستون سوم را به کلاس داده‌ها اختصاص می‌دهیم و قبل از آن با استفاده از دستور np.where کلاس‌هایی که ۱- هستند را به ۰ تبدیل می‌کنیم و سپس داده‌ها را با نسبت ۸۰ و ۲۰ به داده‌های آموزش و داده‌های تست تقسیم می‌کنیم و سپس آن‌ها را به آرایه تبدیل می‌کنیم:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1h1W7R-sTNoGeXsOJ_Fm8pwRHK4Qrw4Aw

df = pd.read_csv('Perceptron.csv')
df.head()

X = df.iloc[:,0:2]
y = df.iloc[:,-1]
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=np.asarray(x_train)
y_train=np.array(y_train)
y_train=y_train.reshape(-1,1)
y_train.shape,x_train.shape
```

Code 1: My Caption (Python)

توابع activation و loss و میزان دقت را تعریف می‌کنیم:

```
#activation functions
def relu(x):
    return np.maximum(0, x)

def sigmoid(x):
    return 1/(1+np.exp(-x))
```

```
7
8  def tanh(x):
9      pass
10
11 #loss functions
12 def bce(y, y_hat):
13     return np.mean(-(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)))
14
15 def mse(y, y_hat):
16     return np.mean((y - y_hat)**2)
17
18 def accuracy(y, y_hat, t=0.5):
19     y_hat = np.where(y_hat<t, 0, 1)
20     acc = np.sum(y == y_hat) / len(y)
21     return acc
```

Code 2: My Caption (Python)

برای ساخت نرون از دستور class استفاده می‌کنیم. برای این نرون تعداد ویژگی‌ها را و function activation و تعداد انجام آموزش را با تعداد پیش‌فرض ۱۰۰ و ضریب یادگیری را با مقدار پیش‌فرض ۰/۱ طراحی می‌کنیم و اگر verbose که به صورت پیش‌فرض true در نظر گرفته شده است، روشن باشد به ازای هر ۱۰ ایتر مقادیر بدست آمده را نشان می‌دهد. تمامی پارامترهای موجود در ساخت این نرون را به عنوان پارامتر خود نرون معرفی می‌کنیم، بایاس را در ابتدا ۰/۵ در نظر می‌گیریم و قسمت predict و fit و descent gradient را به عنوان توابع این نرون با دستورات همانند پروژه قبل تعریف می‌کنیم:

```
1  class Neuron:
2
3      def __init__(self, in_features, af=None, loss_fn=mse, n_iter=100, eta=0.1,
       verbose=True):
4          self.in_features = in_features
5          # weight & bias
6          self.w = np.random.randn(in_features, 1)
7          self.b = 0.5
8          self.af = af
9          self.loss_fn = loss_fn
10         self.loss_hist = []
11         self.w_grad, self.b_grad = None, None
12         self.n_iter = n_iter
13         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):
            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.mean():.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}
```

Code 3: My Caption (Python)

برای آموزش داده‌ها روی این نرون، با استفاده از function activation سیگموید از دستور زیر استفاده می‌کنیم و پارامترهای بدست آمده را چاپ می‌کنیم:

```
neuron = Neuron(in_features=2, af=sigmoid, loss_fn=bce, n_iter=100, eta=0.1,
```

```
    verbose=True)
neuron.fit(x_train, y_train)
neuron.parameters()

Iter=0, Loss=1.6179
Iter=10, Loss=0.7447
Iter=20, Loss=0.4076
Iter=30, Loss=0.2694
Iter=40, Loss=0.2000
Iter=50, Loss=0.1592
Iter=60, Loss=0.1325
Iter=70, Loss=0.1138
Iter=80, Loss=0.1000
Iter=90, Loss=0.0893
{'w': array([[-0.79808152],
        [-2.0276887 ]]),
 'b': 0.1598341503434096}
```

Code 4: My Caption (Python)

۲.۱ نتیجه را روی داده های مجموعهٔ آزمون نشان دهید و دقت را به دست آورید. برای داده های تست دوخط موازی جداکنندهٔ به دست آمده از قاعدهٔ پرسپترون را نمایش دهید و داده های تفکیک شدهٔ دو کلاس را با رنگ مجزا در Scatter Plot مشخص کنید.

برای داده‌های تست y-hat را مطابق دستور موجود در نرون محاسبه می‌کنیم و دقت را اندازه می‌گیریم:

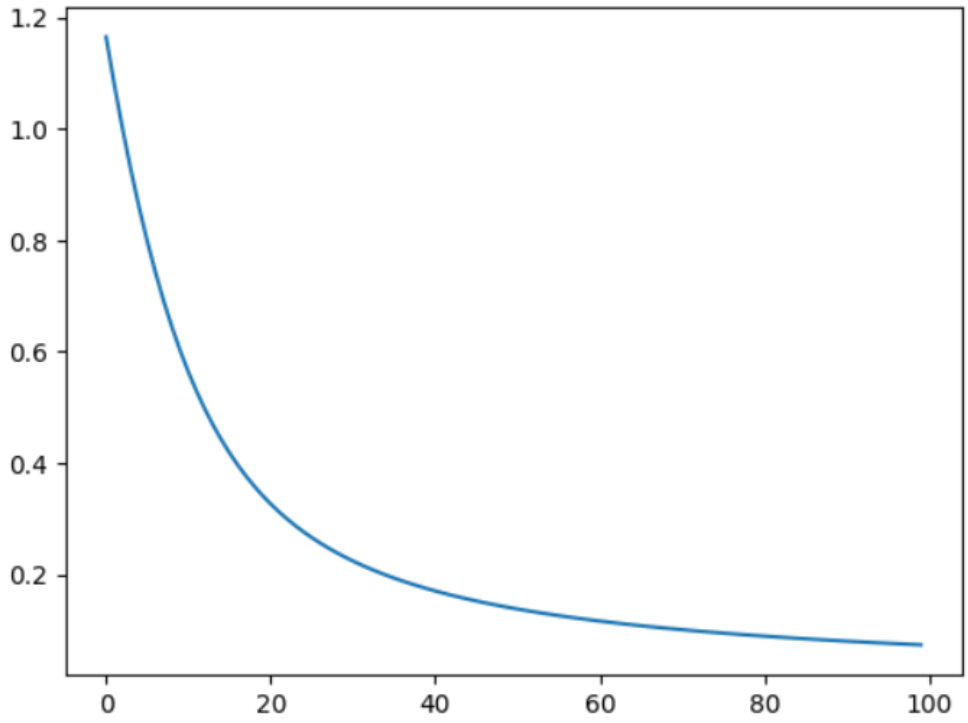
```
y_hat = neuron.predict(x_test)
accuracy(y_test[:, None], y_hat, t=0.5)

#1.0
```

Code 5: My Caption (Python)

تابع اتلاف ما به شکل زیر است:

```
plot_decision_regions(x_train, y_train, clf=neuron)
```

Code 6: My Caption (Python)

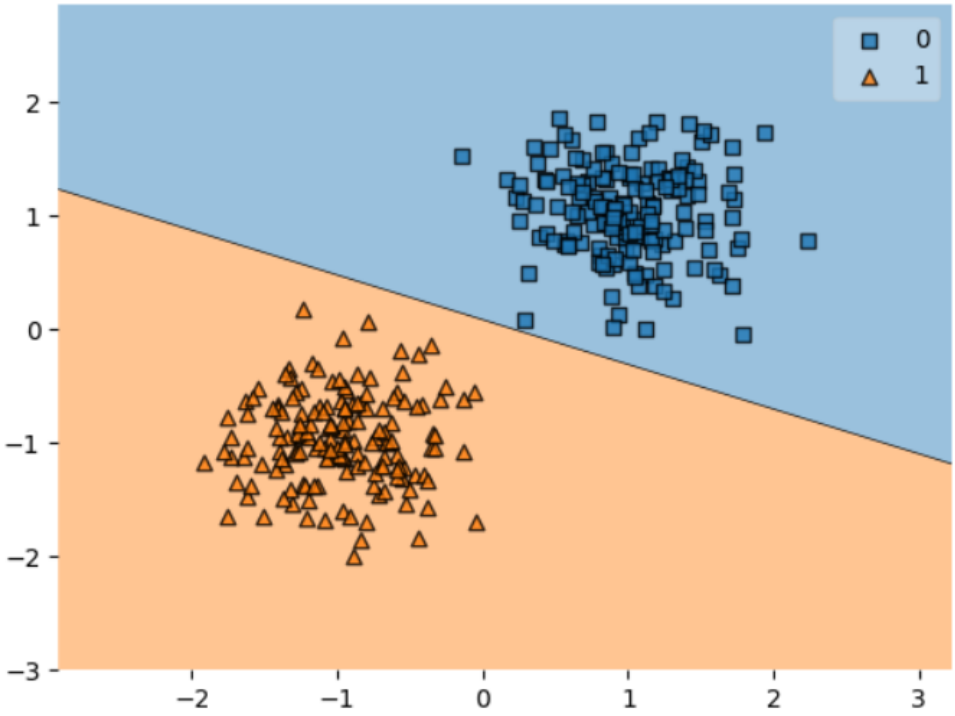


شکل ۱: شکل شماره ۱

۳.۱ قسمت های «۱» و «۲» را با آستانهٔ 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.

برای این قسمت بایاس را عدد دیگری مانند ۶ در نظر می‌گیریم و مراحل قبل را تکرار می‌کنیم:

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

Iter=0, Loss=2.6764
Iter=10, Loss=1.9056
Iter=20, Loss=1.2141
Iter=30, Loss=0.7251
Iter=40, Loss=0.4540
Iter=50, Loss=0.3106
Iter=60, Loss=0.2296
Iter=70, Loss=0.1798
```



شکل ۲: شکل شماره ۲

```
Iter=80, Loss=0.1467
Iter=90, Loss=0.1236
{'w': array([[-3.0686377 ],
        [-2.50747621]]),
 'b': 3.4136654571664766}
```

Code 7: My Caption (Python)

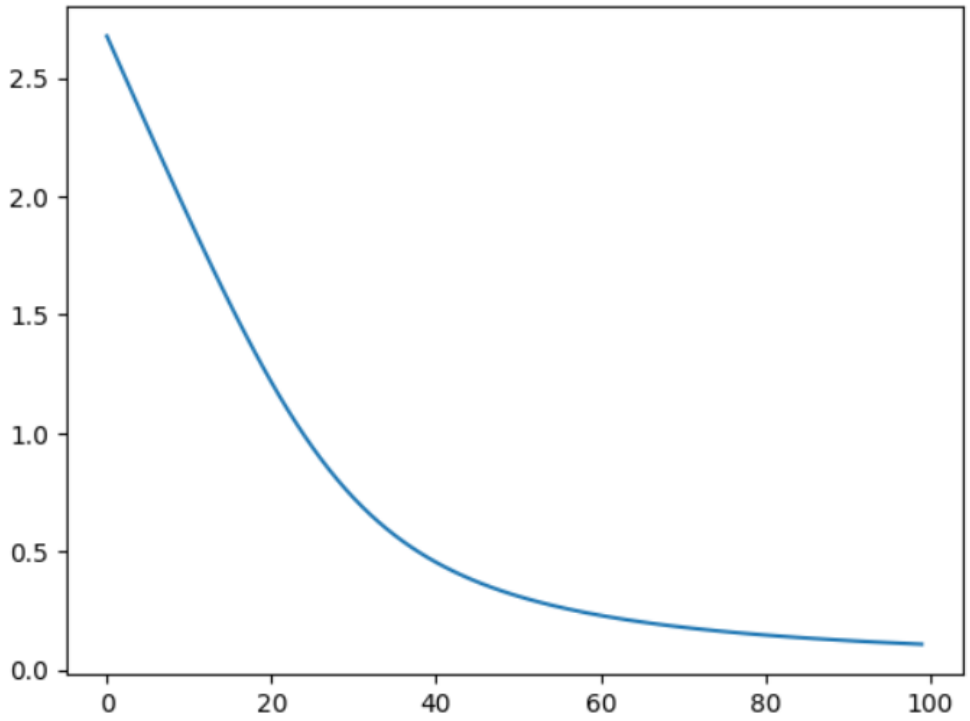
دقت ما کمتر از حالت قبل شده و برابر ۰/۹۷۵ می‌شود:

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

#0.975
```

Code 8: My Caption (Python)

نمودار تابع اتلاف: و نمودار plot: scatter

همانطور که در شکل نیز مشاهده می‌شود با تغییر دادن آستانه، دقت نورون ما تغییر می‌کند و همچنین ناحیه تصمیم گیری نرون



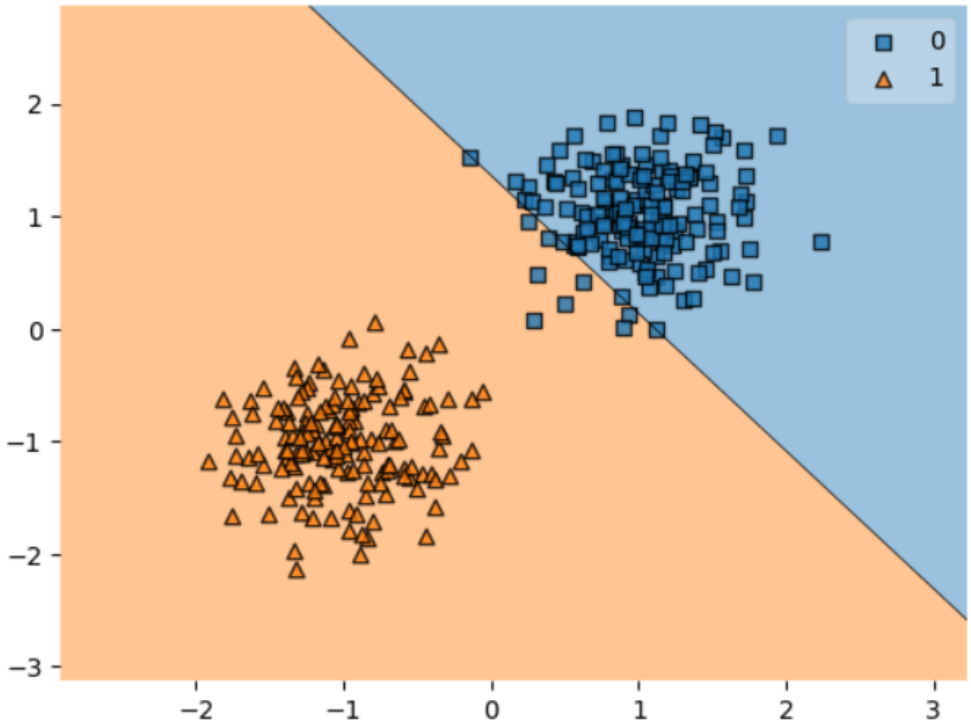
شکل ۳: شکل شماره ۳

که باعث کلاس بندی با استفاده از یک حداقل مقدار می‌شود دسخوش تغییر می‌شود و ممکن است اشتباهاتی صورت گیرد. البته تعیین بهینه‌ی آستانه معمولاً به صورت تجربی و با آزمایش‌های متعدد روی داده‌ها صورت می‌گیرد و بسته به مسئله‌ی خاص و داده‌های مورد استفاده، تأثیرات آن می‌تواند متفاوت باشد.

آستانه مقداری است که در مدل پرسپترون به عنوان یک مقدار ثابت برای تصمیم‌گیری در مورد فعال‌سازی نورون‌ها و استخراج خروجی مورد استفاده قرار می‌گیرد. آستانه تعیین می‌کند که خروجی نورون بعد از تابع فعال‌سازی (مثلاً تابع پله) چگونه باید باشد. اگر خروجی تابع فعال‌سازی از آستانه کمتر باشد، خروجی صفر خواهد بود، در حالی که اگر بیشتر یا مساوی آستانه باشد، ممکن است مقداری دیگر باشد. تغییرات در آستانه می‌تواند تأثیر زیادی بر فرآیند یادگیری پرسپترون داشته باشد. آستانه می‌تواند تعیین‌کننده باشد که یک وزن خاص چقدر باید تغییر کند. تغییرات در آستانه می‌تواند منجر به تغییر در روند یادگیری و سرعت همگرایی شبکه شود. مقدار آستانه ممکن است تأثیر مستقیمی بر قدرت پرسپترون در جداسازی الگوها داشته باشد. آستانه می‌تواند به عنوان یک عامل تعیین‌کننده در تفکیک الگوهای مختلف باشد. آستانه می‌تواند نقشه تصمیم‌گیری مدل را تغییر دهد. این موضوع ممکن است منجر به افزایش یا کاهش دقت یا توانایی پیش‌بینی مدل شود.

برای خنثی کردن اثر بایاس آن را برابر صفر قرار می‌دهیم و مراحل بالا را تکرار می‌کنیم:

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

```

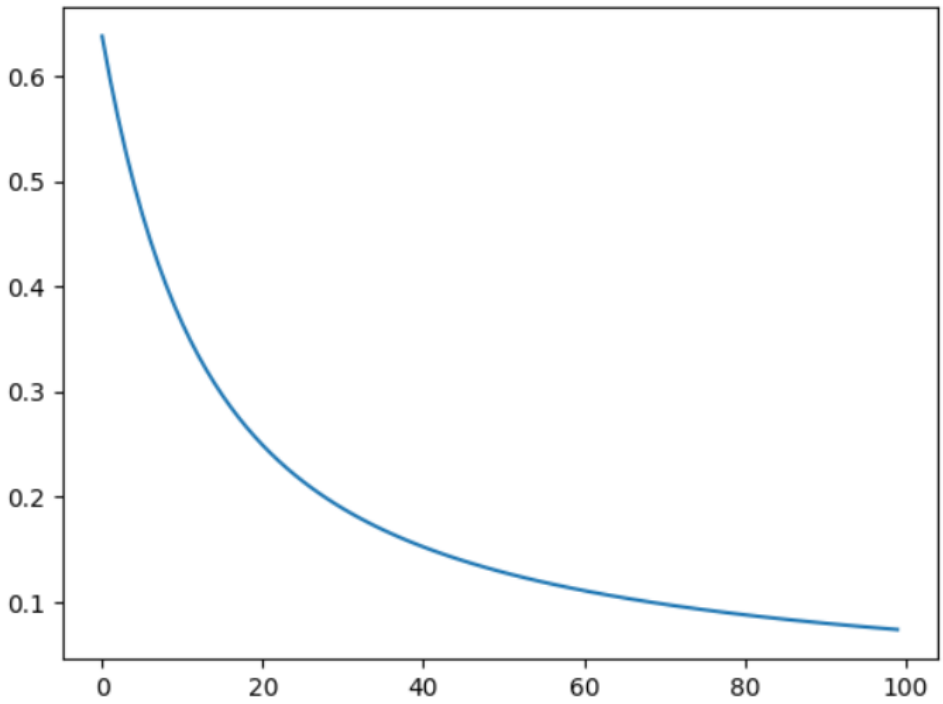


شکل ۴: شکل شماره ۴

```
Iter=0, Loss=0.6378
Iter=10, Loss=0.3643
Iter=20, Loss=0.2489
Iter=30, Loss=0.1886
Iter=40, Loss=0.1522
Iter=50, Loss=0.1279
Iter=60, Loss=0.1107
Iter=70, Loss=0.0977
Iter=80, Loss=0.0877
Iter=90, Loss=0.0796
{'w': array([[-0.83111323],
        [-2.12384767]]),
 'b': -0.0026983614755450697}

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

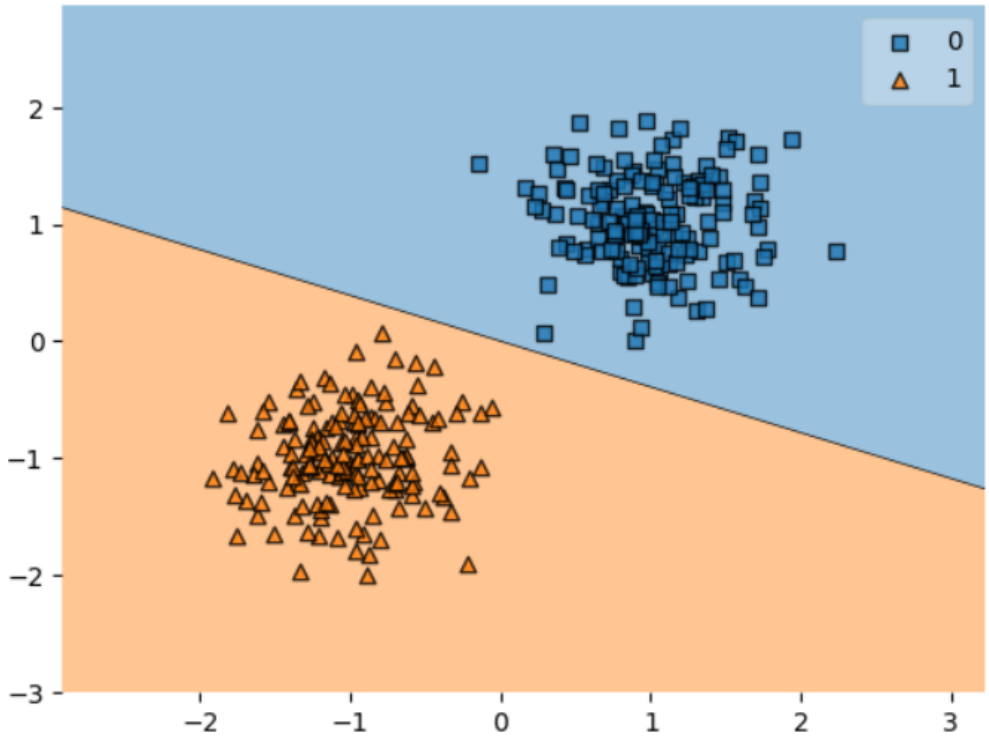
```
#1.0
```

Code 9: My Caption (Python)



شکل ۵: شکل شماره ۵

حذف بایاس ممکن است منجر به کاهش توانایی مدل در توجیه و یادگیری الگوهای پیچیده شود. بایاس به شبکه کمک می‌کند الگوها و ویژگی‌های پیچیده‌تر را در داده‌ها تصویرسازی کند. بایاس به شبکه کمک می‌کند تطبیق بهتری با داده‌های آموزشی داشته باشد. حذف آن ممکن است باعث از دست رفتن قدرت تطبیق و تعمیم به داده‌های جدید شود. در برخی موارد خاص، ممکن است حذف بایاس موثر باشد؛ اما این بستگی به نوع مسئله و داده‌های مورد استفاده دارد. حذف بایاس می‌تواند نقشه تصمیم‌گیری شبکه را تغییر دهد. ممکن است باعث شود که مدل در کلاس‌های مختلف دقت کمتری داشته باشد یا خطای بیشتری را نمایش دهد.



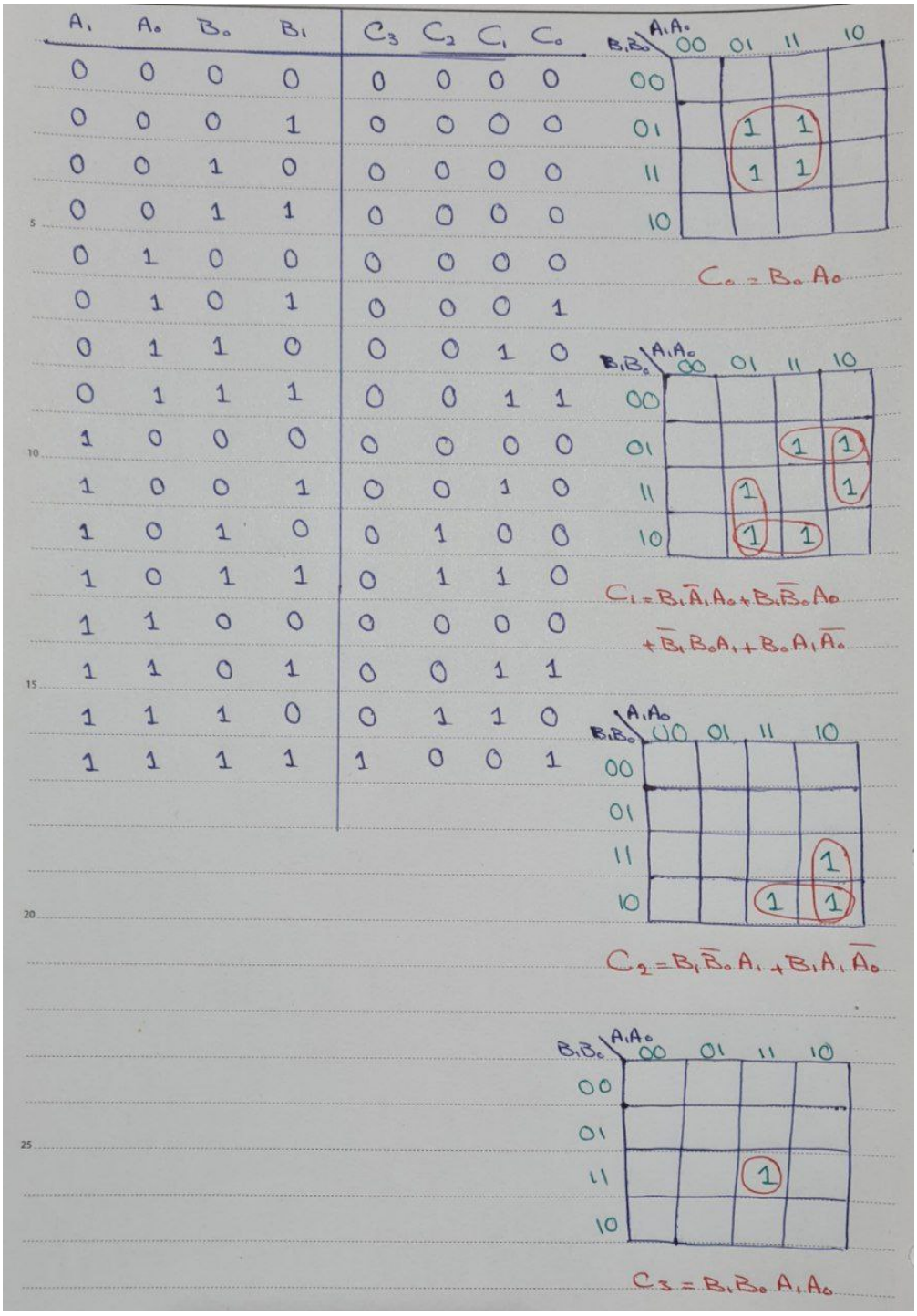
شکل ۶: شکل شماره ۶

۲ سوال دوم

۱.۲ به کمک نورون Pitts-McCulloch توسعه یافته، یک ضرب کنندهٔ باینری بسازید که دو ورودی دوبیتی را گرفته و ۱. به کمک نورون Pitts-McCulloch توسعه یافته آن ها را ضرب کند. برای این کار به دو ورودی دوبیتی (در واقع چهار نورون برای همهٔ ورودی ها) نیاز داریم. هم چنین چهار بیت خروجی (چهار نورون) مورد نیاز است. توجه شود که تمامی نورون های وروی 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.

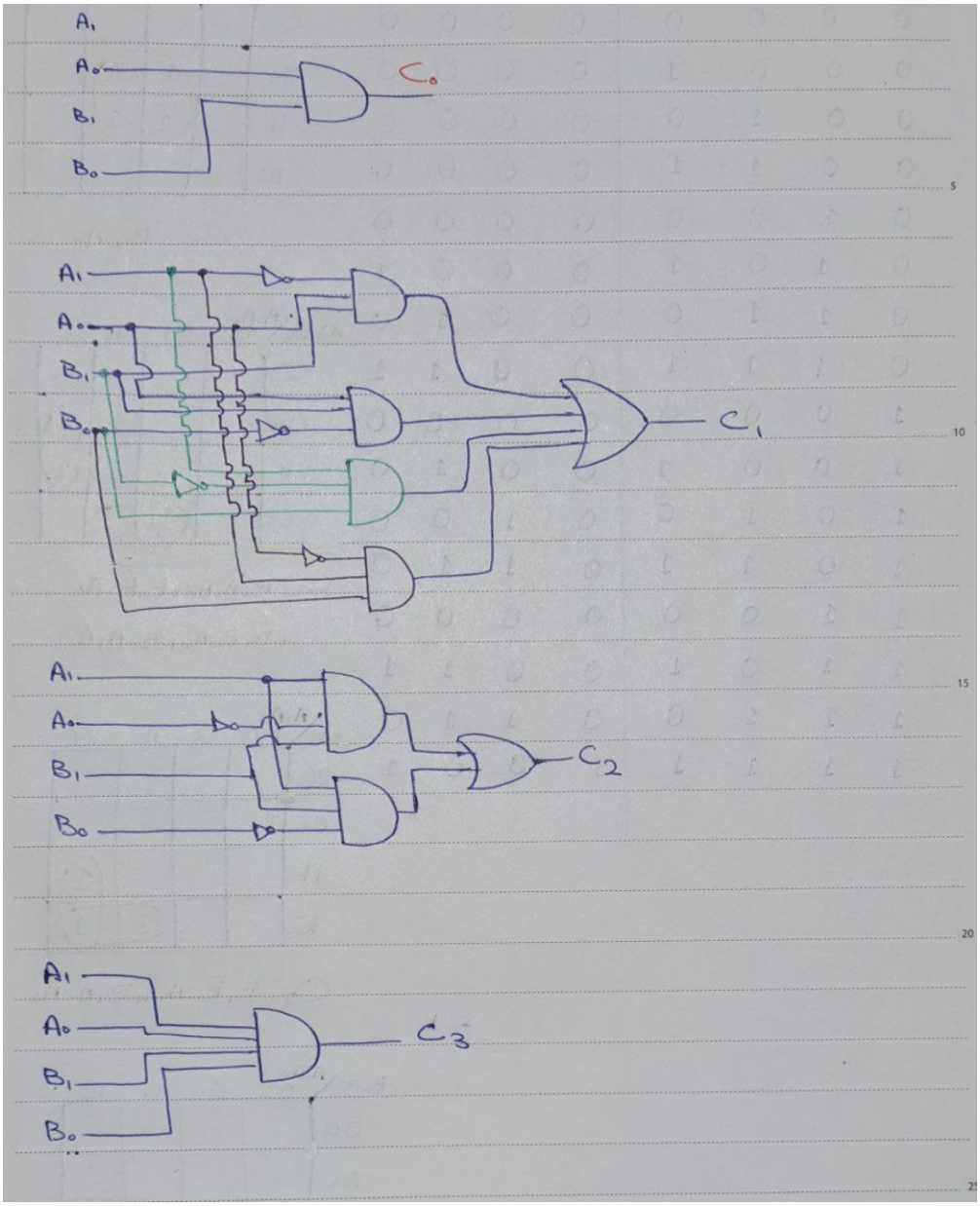
ضمن رسم جدول ورودی خروجی، شبکهٔ هر خروجی را به همراه توضیحات مختصری رسم کنید (نیازی به کدنویسی در این قسمت نیست). دقت داشته باشید که شبکه ای که برای هر خروجی رسم می کنید تا حد ممکن دارای کم ترین تعداد نورون و کم ترین آستانه باشد (تعداد نورون کم تر دارای اهمیت بالاتری نسبت به آستانهٔ کوچک تر است). هم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

ابتدا جدول درستی ضرب دو عدد باینری A و B دو بیتی را که به شکل زیر است می‌کشیم. سپس جدول کارنو هر ۴ بیت خروجی را جدا جدا رسم می‌کنیم. علاوه بر روش دستی، می‌توانیم از سایت 32x8.com نیز استفاده کنیم.



شکل ۷: جدول درستی ضرب باینری

سپس با توجه به ضرب و جمع بین ورودی‌ها، برای هر خروجی گیت منطقی آن را رسم می‌کنیم.



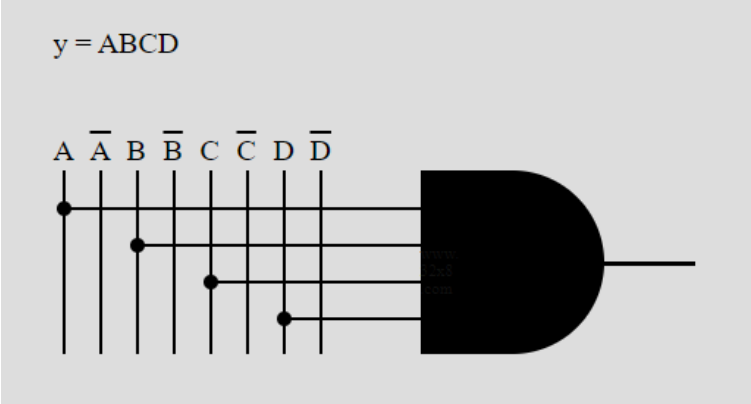
شکل ۸: گیت‌های خروجی

برای بیت اول خروجی:

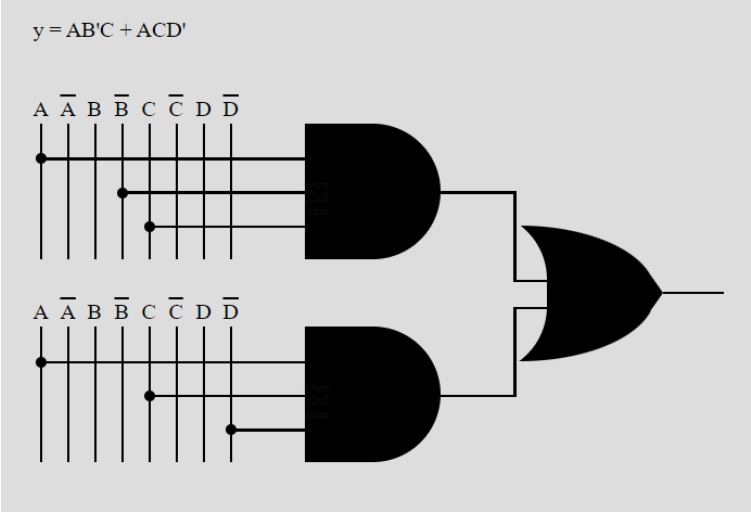
برای بیت دوم خروجی:

برای بیت سوم خروجی:

برای بیت چهارم خروجی:



شکل ۹: بیت اول خروجی

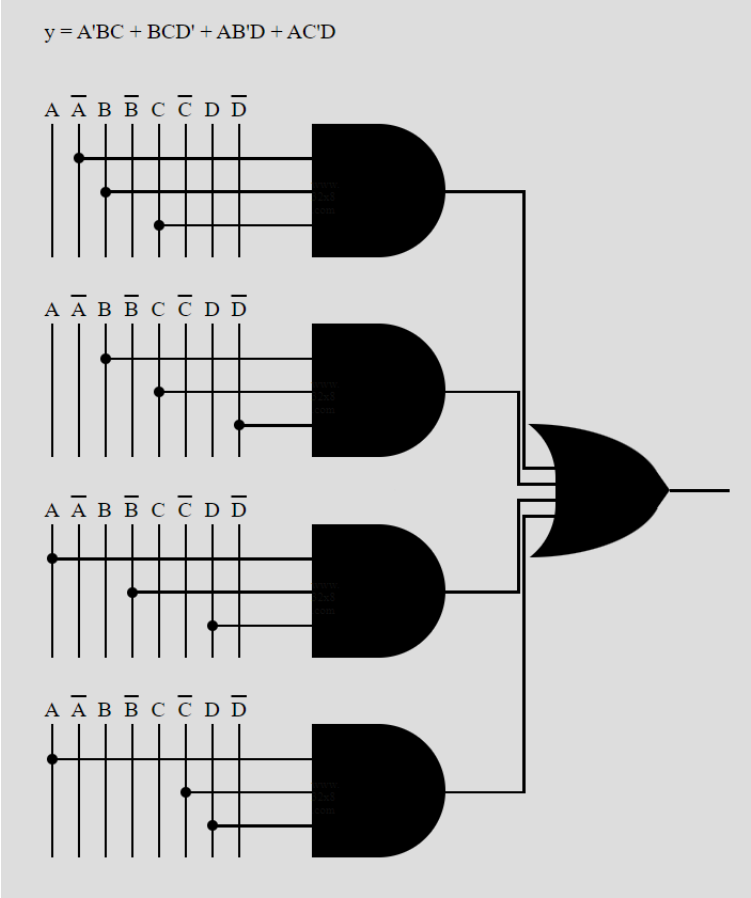


شکل ۱۰: بیت دوم خروجی

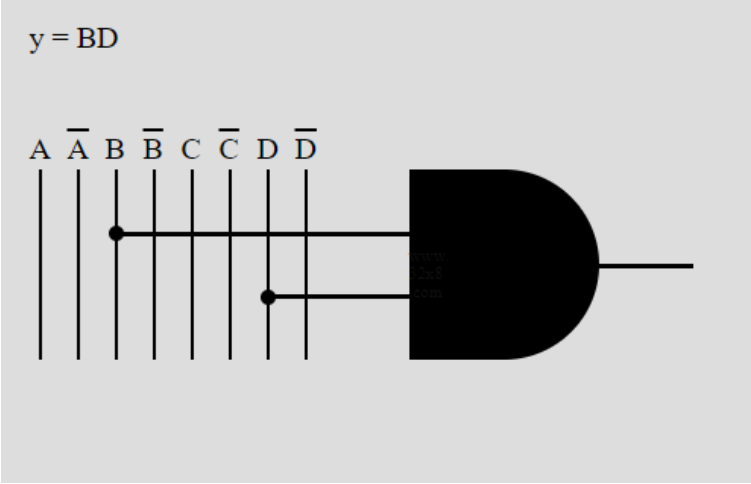
۲.۲ با استفاده از زبان پایتون شبکه های طراحی شده در قسمت «۱» را پیاده سازی کرده و تمامی حالات ممکن را به صورت مناسبی نشان دهید.

ابتدا نرون muculloch pitts را معرفی می‌کنیم. این نرون دو ویژگی اصلی دارد. وزن و آستانه. دو ورودی weights و threshold را می‌گیرد که وزن‌های مورد استفاده در نورون و آستانه‌ی آن را تعیین می‌کنند. weights مجموعه وزن‌های ورودی است که بر روی آن‌ها عملیات ضرب داخلی انجام می‌شود. threshold یک مقدار آستانه است که ورودی‌هایی که مجموع ضرب وزن‌ها در ورودی‌ها بیشتر از آن است، را فعال می‌کند. متد model وظیفه‌ی پیش‌بینی خروجی نورون بر اساس مدل مک‌کالوک-پیتس را دارد. در این مدل، ابتدا عملیات ضرب داخلی بین وزن‌ها و ورودی‌ها با استفاده از x @ self.weights انجام می‌شود. سپس مقدار حاصل از ضرب داخلی با آستانه self.threshold مقایسه می‌شود. اگر مقدار ضرب داخلی بیشتر یا مساوی با آستانه باشد، نورون خروجی ۱ (یا فعال) را تولید می‌کند و در غیر این صورت خروجی ۰ (یا غیرفعال) را باز می‌گرداند.

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



شکل ۱۱: بیت سوم خروجی



شکل ۱۲: بیت چهارم خروجی

```
3
4  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
```

Code 10: My Caption (Python)

برای بیت خروجی اول که یک گیت and است، نرون and را تعریف می‌کنیم. این نرون دو ورودی با وزن ۱ را می‌گیرد و بایاس این نرون ۱/۵ می‌باشد:

```
def y0(input):

  neur5 = McCulloch_Pitts_neuron([1,1],1.5)

  z5 = neur5.model(np.array([input[1],input[3]]))

  return list([z5])
```

Code 11: My Caption (Python)

سپس با استفاده از کتابخانه itertools حالات مختلف ورودی را برای ۴ ورودی به ازای ۰ و ۱ می‌سازیم و به عنوان ورودی به نرون y_0 می‌دهیم و خروجی را بدست می‌آوریم:

```
import itertools
# inputs

input = [0,1]

X = list(itertools.product(input, input, input, input))

for i in X:
    res = y0(i)
    print("y0 with input as", str(i[0]) + str(" ")+str(i[1])+ str(" ")+str(i
    [2])+ str(" ")+str(i[3]), "goes to output ", str(res[0]))
```

Code 12: My Caption (Python)

نرون بعدی نرونی با ۴ and و یک or هست. برای ساخت and به بایاس ۱/۵ و برای or به بایاس ۰/۵ نیاز داریم. ورودی and ها را

```
y1 with input as 0 0 0 0 goes to output  0
y1 with input as 0 0 0 1 goes to output  0
y1 with input as 0 0 1 0 goes to output  0
y1 with input as 0 0 1 1 goes to output  0
y1 with input as 0 1 0 0 goes to output  0
y1 with input as 0 1 0 1 goes to output  0
y1 with input as 0 1 1 0 goes to output  1
y1 with input as 0 1 1 1 goes to output  1
y1 with input as 1 0 0 0 goes to output  0
y1 with input as 1 0 0 1 goes to output  1
y1 with input as 1 0 1 0 goes to output  0
y1 with input as 1 0 1 1 goes to output  1
y1 with input as 1 1 0 0 goes to output  0
y1 with input as 1 1 0 1 goes to output  1
y1 with input as 1 1 1 0 goes to output  1
y1 with input as 1 1 1 1 goes to output  0
```

شکل ۱۳: وضعیت خروجی

با توجه به وزن هر کدام از ورودی‌ها مشخص می‌کنیم و سپس تمامی هاand را با یکدیگر or می‌کنیم.

```python
def y1(input):

  neur1 = McCulloch_Pitts_neuron([1,-1, 0,1], 1.5)
  neur2 = McCulloch_Pitts_neuron([1,0, -1,1], 1.5)
  neur3 = McCulloch_Pitts_neuron([0,1,1,-1], 1.5)
  neur4 = McCulloch_Pitts_neuron([-1,1,1,0], 1.5)
  neur5 = McCulloch_Pitts_neuron([1,1,1,1], 0.5)

  z1 = neur1.model(np.array([input[0], input[1],input[2],input[3]]))
  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([z1,z2,z3,z4]))
  # 3 bit output
  # return str(z1) + str(z2)
  return list([z5])

import itertools
```

```
# inputs

input = [0,1]

X = list(itertools.product(input, input, input, input))

for i in X:
    res = y1(i)
    print("y0 with input as", str(i[0]) + str(" ")+str(i[1])+ str(" ")+str(i
    [2])+ str(" ")+str(i[3]), "goes to output ", str(res[0]))
```

Code 13: My Caption (Python)

خروجی آن به شکل زیر می‌شود: برای خروجی بعدی دو گیت and با ۳ ورودی و یک گیت or برای خروجی and ها داریم. و مانند

```
y0 with input as 0 0 0 0 goes to output  0
y0 with input as 0 0 0 1 goes to output  0
y0 with input as 0 0 1 0 goes to output  0
y0 with input as 0 0 1 1 goes to output  0
y0 with input as 0 1 0 0 goes to output  0
y0 with input as 0 1 0 1 goes to output  0
y0 with input as 0 1 1 0 goes to output  1
y0 with input as 0 1 1 1 goes to output  1
y0 with input as 1 0 0 0 goes to output  0
y0 with input as 1 0 0 1 goes to output  1
y0 with input as 1 0 1 0 goes to output  0
y0 with input as 1 0 1 1 goes to output  1
y0 with input as 1 1 0 0 goes to output  0
y0 with input as 1 1 0 1 goes to output  1
y0 with input as 1 1 1 0 goes to output  1
y0 with input as 1 1 1 1 goes to output  0
```

شکل ۱۴: وضعیت خروجی

قسمت‌های بالا ضرایب ورودی‌ها را مشخص می‌کنیم:

```
def y2(input):

  neur1 = McCulloch_Pitts_neuron([1,0,1,-1], 1.5)
  neur2 = McCulloch_Pitts_neuron([1,-1,1,0], 1.5)
  neur3 = McCulloch_Pitts_neuron([1,1], 0.5)

```

```
  z1 = neur1.model(np.array([input[0], input[1],input[2],input[3]]))
  z2 = neur2.model(np.array([input[0], input[1],input[2],input[3]]))
  z3 = neur3.model(np.array([z1,z2]))

  # 3 bit output
  # return str(z1) + str(z2)
  return list([z3])

import itertools
# inputs

input = [0,1]

X = list(itertools.product(input, input, input, input))

for i in X:
    res = y2(i)
    print("y2 with input as", str(i[0]) + str(" ")+str(i[1])+ str(" ")+str(i
    [2])+ str(" ")+str(i[3]), "goes to output ", str(res[0]))
```

Code 14: My Caption (Python)

خروجی آن به شکل زیر می‌شود: برای خروجی بعدی یک گیت and ۴ ورودی داریم. بنابراین بایاس آن برابر ۳/۵ می‌شود و ضرایب تمام ورودی‌ها ۱ می‌شود:

```
def y3(input):

  neur1 = McCulloch_Pitts_neuron([1,1,1,1], 3.5)

  z1 = neur1.model(np.array([input[0], input[1],input[2],input[3]]))

  return list([z1])

import itertools
# inputs

input = [0,1]

```

```
y2 with input as 0 0 0 0 goes to output  0
y2 with input as 0 0 0 1 goes to output  0
y2 with input as 0 0 1 0 goes to output  0
y2 with input as 0 0 1 1 goes to output  0
y2 with input as 0 1 0 0 goes to output  0
y2 with input as 0 1 0 1 goes to output  0
y2 with input as 0 1 1 0 goes to output  0
y2 with input as 0 1 1 1 goes to output  0
y2 with input as 1 0 0 0 goes to output  0
y2 with input as 1 0 0 1 goes to output  0
y2 with input as 1 0 1 0 goes to output  1
y2 with input as 1 0 1 1 goes to output  1
y2 with input as 1 1 0 0 goes to output  0
y2 with input as 1 1 0 1 goes to output  0
y2 with input as 1 1 1 0 goes to output  1
y2 with input as 1 1 1 1 goes to output  0
```

شکل ۱۵: وضعیت خروجی

```
14 X = list(itertools.product(input, input, input, input))
15
16 for i in X:
17     res = y3(i)
18     print("y3 with input as", str(i[0]) + str(" ")+str(i[1])+ str(" ")+str(i
       [2])+ str(" ")+str(i[3]), "goes to output ", str(res[0]))
```

Code 15: My Caption (Python)

خروجی آن به شکل زیر می‌شود:

```
y3 with input as 0 0 0 0 goes to output  0
y3 with input as 0 0 0 1 goes to output  0
y3 with input as 0 0 1 0 goes to output  0
y3 with input as 0 0 1 1 goes to output  0
y3 with input as 0 1 0 0 goes to output  0
y3 with input as 0 1 0 1 goes to output  0
y3 with input as 0 1 1 0 goes to output  0
y3 with input as 0 1 1 1 goes to output  0
y3 with input as 1 0 0 0 goes to output  0
y3 with input as 1 0 0 1 goes to output  0
y3 with input as 1 0 1 0 goes to output  0
y3 with input as 1 0 1 1 goes to output  0
y3 with input as 1 1 0 0 goes to output  0
y3 with input as 1 1 0 1 goes to output  0
y3 with input as 1 1 1 0 goes to output  0
y3 with input as 1 1 1 1 goes to output  1
```

شکل ۱۶: وضعیت خروجی

۳ به این دفترچه کد مراجعه کنید و با اجرای سلول اول، ۵ دادهٔ 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 ارائهٔ توضیحات متنی و دیداری مناسب لازم است. مثلا می توانید ورودی نویزی و خروجی پیش بینی شده را در یک تصویر در کنار هم قرار دهید.

۱.۳ دو تابع 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 نحوهٔ عملکرد هریک از این توابع توضیح دهید. هم چنین، می توانید این دستورات را به صورتی بهتر و کارآمدتر بازنویسی کنید.

در قسمت ابتدایی کد توابع مربوط برای کار کردن با تصاویر را با استفاده از کتابخانه PIL فراخوانی می‌کنیم. این تابع آدرس تصویر را دریافت می‌کند و کد باینری تصویر را به ما می‌دهد. خط اول کد زیر تصویر را باز می‌کند و خط دوم draw را برای تغییر شکل ایجاد می‌کند. در خط بعدی عرض و طول تصویر را مشخص می‌کنیم و بعد مقادیر پیکسل را برای تصویر بارگزاری می‌کنیم. factor یک مقدار شدت آستانه است که برای تصمیم گیری در مورد سیاه یا سفید بودن یک پیکسل استفاده می شود. و خط آخر پیکسل‌های تصویر را ذخیره می‌کند. -۱ برای سفید و ۱ برای مشکی استفاده می‌شود.

```
    image = Image.open(path)
```

```
    draw = ImageDraw.Draw(image)
    width = image.size[0]
    height = image.size[1]
    pix = image.load()
    factor = 100
    binary_representation = []
```

Code 16: My Caption (Python)

در این قسمت کد در پیکسل‌ها می‌چرخیم و مقادیر مشخص شده را به آن‌ها اختصاص می‌دهیم. با استفاده از for در تمامی پیکسل‌ها گردش می‌کنیم. مقادیر RGB هر پیکسل را استخراج می‌کند و شدت کل را محاسبه می‌کند. بر اساس آستانه شدت، تعیین می کند که پیکسل باید سفید یا سیاه باشد: اگر شدت بیشتر از یک آستانه باشد، مقادیر RGB مربوط به سفید که تمامی آن‌ها ۲۵۵ است را می‌گیرد و پیکسل را سفید (-۱) در نظر می‌گیرد و ۱- را به binary-representation آن اضافه می کند. در غیر این صورت آن را مشکی در نظر می‌گیرد و RGB آن را تماما صفر می‌کند و عدد ۱ ر به آن اضافه می‌کند. خط آخر کد هم رنگ پیکسل در تصویر را با توجه به رنگ تعیین شده (سیاه یا سفید) تنظیم می‌کند.

```
    for i in range(width):
        for j in range(height):
            red = pix[i, j][0]
            green = pix[i, j][1]
            blue = pix[i, j][2]

            total_intensity = red + green + blue

            if total_intensity > (((255 + factor) // 2) * 3):
                red, green, blue = 255, 255, 255  # White pixel
                binary_representation.append(-1)  # Append -1 for white pixel
            else:
                red, green, blue = 0, 0, 0  # Black pixel
                binary_representation.append(1)  # Append 1 for black pixel

            draw.point((i, j), (red, green, blue))
```

Code 17: My Caption (Python)

del draw ابزار ترسیم را پاک می‌کند. در آخر به ما binary-representation را که شامل نمایش باینری تصویر بر اساس شدت پیکسل است را بر می‌گرداند. این تابع اساساً هر پیکسل در تصویر را پردازش می‌کند، شدت آن را تعیین می‌کند و آن را به یک نمایش باینری (سیاه یا سفید) بر اساس یک آستانه از پیش تعریف‌شده تبدیل می‌کند و این مقادیر را در یک لیست ذخیره می‌کند.

```
    del draw
```

```
    return binary_representation
```

Code 18: My Caption (Python)

تغییری که من در این کد برای استفاده از آن دادم این بود که بجای آنکه برای مقادیر سفید عدد ۱- را نشان دهد، عدد ۰ را نشان دهد:

```
from PIL import Image, ImageDraw

def convertImageToBinary(path):
    """
    Convert an image to a binary representation based on pixel intensity.

    Args:
        path (str): The file path to the input image.

    Returns:
        list: A binary representation of the image where white is represented
    by 0 and black is represented by 1.
    """
    # Open the image file.
    image = Image.open(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 intensity thresholding.
    factor = 100

    # Initialize an empty list to store the binary representation.
    binary_representation = []

```

```
    # Loop through all pixels in the image.
    for i in range(width):
        for j in range(height):
            # Extract the Red, Green, and Blue (RGB) values of the pixel.
            red = pix[i, j][0]
            green = pix[i, j][1]
            blue = pix[i, j][2]

            # Calculate the total intensity of the pixel.
            total_intensity = red + green + blue

            # Determine whether the pixel should be white or black based on
    the intensity.
            if total_intensity > (((255 + factor) // 2) * 3):
                red, green, blue = 255, 255, 255
                binary_representation.append(0)
            else:
                red, green, blue = 0, 0, 0
                binary_representation.append(1)

    # Clean up the drawing tool.
    del draw

    # Return the binary representation of the image.
    return binary_representation
```

Code 19: My Caption (Python)

کد بعدی که در سلول سوم قرار دارد، تصاویر نویزدار را بر اساس تصاویر ورودی ارائه شده تولید می کند. ابتدا مانند کد قبلی PIL را برای تصویر فراخوانی می‌کنیم و از random برای ایجاد نویز تصادفی استفاده می‌کنیم. تابع getNoisyBinaryImage آدرس تصویر ورودی و آدرس تصویر خروجی برای ذخیره کردن تصویر نویزی را به عنوان ورودی می‌گیرد. خط اول تا چهارم این کد مانند تابع قبلی است. در خط پنجم مقادیر پیکسل تصویر را بارگزاری می‌کند و noise-factor مقداری است که برای کنترل شدت نویز اضافه شده استفاده می شود.

```
    image = Image.open(input_path)
    draw = ImageDraw.Draw(image)
    width = image.size[0]
```

```
    height = image.size[1]
    pix = image.load()
    noise_factor = 5
```

Code 20: My Caption (Python)

این کد در تمامی پیکسل‌ها با استفاده از حلقه for می‌چرخد و مقدار رندوم نویز را برای noise-factor ایجاد می‌کند. نویز ایجاد شده را به مقادیر RGB(red،green،blue) پیکسل‌ها اضافه می‌کند و در نهایت با استفاده از کد خط‌های آخری اطمینان حاصل می‌کند که مقادیر RGB بین ۰ تا ۲۵۵ بماند.

```
    for i in range(width):
        for j in range(height):
            rand = random.randint(-noise_factor, noise_factor)
            red = pix[i, j][0] + rand
            green = pix[i, j][1] + rand
            blue = pix[i, j][2] + rand

            # Ensure RGB values stay within the valid range (0-255).
            red = min(max(red, 0), 255)
            green = min(max(green, 0), 255)
            blue = min(max(blue, 0), 255)

            draw.point((i, j), (red, green, blue))
```

Code 21: My Caption (Python)

و در نهایت تصویر نویزی شده را در آدرس مشخص شده، ذخیره می‌کند.

```
    image.save(output_path, "JPEG")
    del draw
```

Code 22: My Caption (Python)

این تابع بر اساس لیست مسیرهای تصویر ورودی ارائه شده، چندین تصویر نویز تولید می کند. برای هر تصویر ورودی، با استفاده از تابع getNoisyBinaryImage یک تصویر نویزدار متناظر تولید می کند و آن را با نام فایل جدید ذخیره می کند.

```
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"
        getNoisyBinaryImage(image_path, noisy_image_path)
        print(f"Noisy image for {image_path} generated and saved as {
    noisy_image_path}")
```

Code 23: My Caption (Python)

این خط تولید تصاویر نویزدار را بر اساس تصاویر مشخص شده در لیست image-paths آغاز می کند و آنها را در فهرست /content/ ذخیره می کند. پیامی را چاپ می کند که مسیر تصویر اصلی و مسیر مربوطه را که در آن تصویر نویز ذخیره شده است را نشان می دهد. این کد اساساً نویز تصادفی را به پیکسل‌های تصاویر ورودی اضافه می‌کند تا تصاویر جدیدی با نویز اضافه ایجاد کند، و این کار را برای چندین تصویر ورودی مشخص شده در لیست image-paths انجام می‌دهد.

```
generateNoisyImages()
```

Code 24: My Caption (Python)

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

تصاویر اصلی و بدون نویز، و تصاویری را که با تابع generateNoisyImages نویزی کرده‌ایم را به عنولن ورودی به تابع convertImageToBinary می‌دهیم و اعداد باینری تصاویر را ایجاد و 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")
```

Code 25: My Caption (Python)

کد شبکه همینگ به شرح زیر است: کلاس با مجموعه ای از الگوها مقداردهی اولیه می شود و نام آنها به عنوان لیستی از تاپل ها patterns-with-names ارسال می‌شود. self.patterns دیکشنری است که نام الگوها را به عنوان کلید و الگوهای مربوط به آنها را به عنوان مقادیر ذخیره می‌کند. متد train خالی است زیرا شبکه های همینگ نیازی به آموزش ندارند. آنها بر اساس تطبیق الگو با استفاده از فاصله همینگ عمل می کنند. متد recall به منظور یادآوری الگوها است. در این پیاده‌سازی ساده، الگوی ورودی را بدون هیچ پردازشی برمی‌گرداند زیرا شبکه‌های همینگ فراخوانی تکراری را انجام نمی‌دهند. متد hamming-network فاصله هامینگ بین دو الگو را محاسبه می کند. با استفاده از zip برای مقایسه بیت های مربوطه و شمارش عدم تطابق، تعداد بیت های متفاوت بین الگوی ۱ و الگوی ۲ را می شمارد. متد find-closest-match نزدیکترین تطابق الگو با الگوی فراخوان شده را پیدا می کند. از طریق الگوهای ذخیره شده تکرار می شود، فاصله همینگ بین الگوی recalled-pattern و هر الگوی ذخیره شده را محاسبه می کند. فاصله بین الگوی فراخوان شده و هر الگوی ذخیره شده را چاپ می کند. نام الگوی را برمی‌گرداند که حداقل فاصله همینگ (نزدیک‌ترین تطابق) را دارد.

```
class HammingNetwork:
    def __init__(self, patterns_with_names):
        self.patterns = {name: pattern for name, pattern in
   patterns_with_names}

    def train(self):
        pass  # Training not needed as the Hamming distance doesn't require
   training

    def recall(self, input_pattern, max_iterations=100):
        return input_pattern

    def hamming_distance(self, pattern1, pattern2):
        # Calculate Hamming distance between two patterns
        distance = sum(bit1 != bit2 for bit1, bit2 in zip(pattern1, pattern2))
        return distance

    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
```

Code 26: My Caption (Python)

ابتدا pattern ها را که همان باینری‌های تصاویر بدون نویز هستند را تعریف می‌کنیم. و سپس هاpattern را به string تبدیل می‌کنیم. این الگوها را برای مقایسه آسان تر به رشته تبدیل می کند. نمونه ای از کلاس HammingNetwork (hamming-net) با الگوهای ارائه شده ایجاد می شود. یک الگوی نویزدار (noisy-pattern) فرض می شود که وجود داشته باشد و به یک رشته (rescalled-pattern) تبدیل می شود. نزدیکترین تطابق با الگوی فراخوانی شده با استفاده از روش find-closest-match از hamming-net یافت می شود. نام نزدیکترین الگوی منطبق چاپ می شود. این کد یک شبکه همینگ ساده را تعریف می کند که قادر به ذخیره و یادآوری الگوها بر اساس فاصله همینگ است. الگوها و نام‌های آنها را ذخیره می‌کند، امکان یادآوری الگوها را فراهم می‌کند، فاصله همینگ بین الگوهای فراخوان‌شده و الگوهای ذخیره‌شده را محاسبه می‌کند، و نزدیک‌ترین تطابق را در بین الگوهای ذخیره‌شده برای یک الگوی فراخوان‌شده پیدا می‌کند. استفاده از مثال نشان می دهد که چگونه می توان از این شبکه همینگ با یک الگوی نویز استفاده کرد تا نزدیکترین الگوی ذخیره شده را شناسایی کند.

```
# Example usage:
if __name__ == "__main__":
    # Define some patterns with names
    patterns_with_names = [
        ("pattern1", x1),
        ("pattern2", x2),
        ("pattern3", x3),
        ("pattern4", x4),
        ("pattern5", x5)
    ]

    # Convert patterns to strings for easier comparison
    patterns_with_names = [(name, ''.join(map(str, pattern))) for name,
    pattern in patterns_with_names]

    # Create a Hamming network
    hamming_net = HammingNetwork(patterns_with_names)

    # Test recall with a noisy pattern
```

```
    noisy_pattern = p1
    recalled_pattern = ''.join(map(str, noisy_pattern))
    print("Noisy Pattern:", noisy_pattern)
    print("Recalled Pattern:", recalled_pattern)

    # Find the closest match to the recalled pattern and retrieve the pattern
    name
    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.")
```

Code 27: My Caption (Python)

خروجی ما برای داده‌ی نویزی ،p۳ همانند شکل زیر است و درست تشخیص داده شده است:

```
Noisy Pattern: [0, 0, 0, 0, 0, 0,
Recalled Pattern: 000000000000000
Distance from pattern1: 908
Distance from pattern2: 1007
Distance from pattern3: 100
Distance from pattern4: 845
Distance from pattern5: 828
Closest Match Found: pattern3
```

شکل ۱۷: شکل شماره ۱۷

با تنظیم پارامتر noise-factor روی ۲۰۰۰ هنوز هم دقت ۱۰۰ درصد را برای تشخیص الگو داریم و با تنظیم آن روی ۳۰۰۰ دیگر به درستی الگو را تشخیص نمی‌دهد. برای مثال وقتی p۳ را به عنوان ورودی به آن می‌دهیم الگوی اول را به عنوان خروجی به ما باز می‌گرداند. با افزایش مقدار نویز دقت تابع ما کمتر می‌شود و به اشتباه تشخیص می‌دهد.

```
Noisy Pattern: [1, 0, 0, 0, 1, 1,
Recalled Pattern: 10001100011001(
Distance from pattern1: 3949
Distance from pattern2: 4002
Distance from pattern3: 3997
Distance from pattern4: 3954
Distance from pattern5: 3975
Closest Match Found: pattern1
```

شکل ۱۸: شکل شماره ۱۸

۳.۳ با الهام گرفتن از تابع نوشته‌شده برای تولید داده‌های نویزی، یک تابع بنویسید که از داده‌های ورودی، خروجی‌های دارای **missing point** تولید کند. سپس عملکرد شبکه خود را با مقدار مشخصی **missing point** آزمایش و تحلیل کنید. اگر میزان **missing point** از چه حدی بیشتر شود عملکرد شبکه طراحی شده شما دچار اختلال می‌شود؟ راه حل چیست؟

این کد یک شبکه همینگ ابتدایی را برای تشخیص الگو با استفاده از پایتون پیاده سازی می کند. کد دارای چندین عملکرد و عملیات است.

show(matrix): تابع نمایش یک ماتریس به صورت فرمت شده.

change(vector, a, b): تابع تبدیل یک بردار به یک ماتریس با ابعاد مشخص.

product(matrix, vector, T): تابع ضرب یک ماتریس در بردار با پارامتر آستانه.

action(vector, T, Emax): تابع فعال سازی برای پردازش یک بردار بر اساس پارامترهای آستانه.

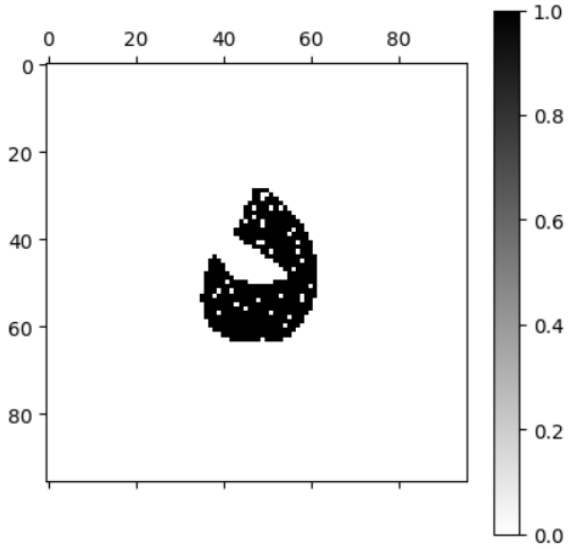
mysum(vector, j): تابعی برای محاسبه مجموع مقادیر برداری به استثنای یک شاخص خاص.

norm(vector, p): تابعی برای محاسبه هنجار اقلیدسی تفاوت بین دو بردار.

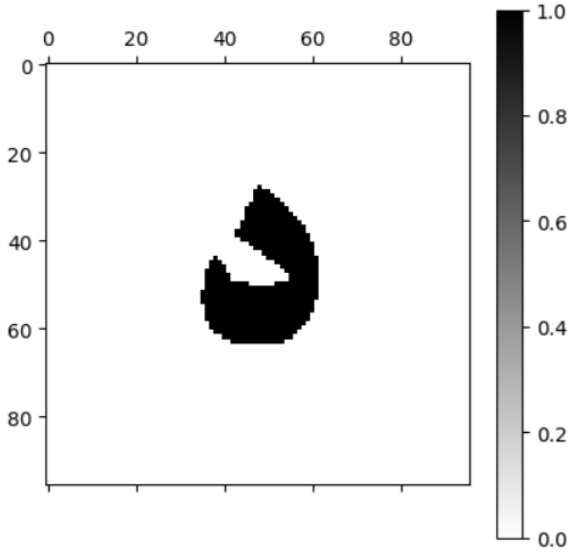
کد مسیر تصویر ورودی (IMAGE-PATH) را تعریف می کند و تصویر را به عنوان یک نمایش باینری بارگذاری می کند. این تصاویر نمونه را به نمایش های باینری تبدیل می کند و آنها را در x ذخیره می کند. این عملیات برای تبدیل و دستکاری داده های تصویر ورودی انجام می دهد: تبدیل تصویر ورودی به ماتریس .(q) تنظیم ماتریس های وزن ،(w) آستانه فعال سازی ،(T) ماتریس اتصال سیناپسی (E) و سایر پارامترها. کد الگوریتم شبکه همینگ را برای تشخیص الگو اجرا می کند: بردارها و ماتریس ها را برای اتصالات خروجی و سیناپسی مقداردهی اولیه می کند. حلقه اصلی تا زمانی تکرار می شود که هنجار اختلاف بین بردارهای خروجی متوالی به زیر یک آستانه (Emax) برسد. بردارهای خروجی (y) را بر اساس قوانین تعریف شده و توابع فعال سازی محاسبه و به روز می کند. در نهایت، بردارهای خروجی را نمایش می دهد و کلاس مرتبط با بالاترین مقدار خروجی مثبت را تعیین می کند. کد، ماتریس حاصل را که با کلاسی که بالاترین مقدار خروجی مثبت را دارد، تجسم می کند.

به طور کلی، این کد یک شبکه Hamming ساده را برای تشخیص الگو پیاده سازی می کند. این تصاویر را به نمایش های باینری تبدیل

می کند، آنها را از طریق تکرارهای شبکه پردازش می کند و کلاسی را با بالاترین مقدار خروجی بر اساس الگوریتم شبکه همینگ تعیین می کند. علاوه بر این، ماتریس حاصل را که با کلاس شناسایی شده مرتبط است، تجسم می کند.
به دلیل طولانی بودن کد در گزارش آورده نشده است. برای تصویر نویزی۴ خروجی به شکل زیر است:

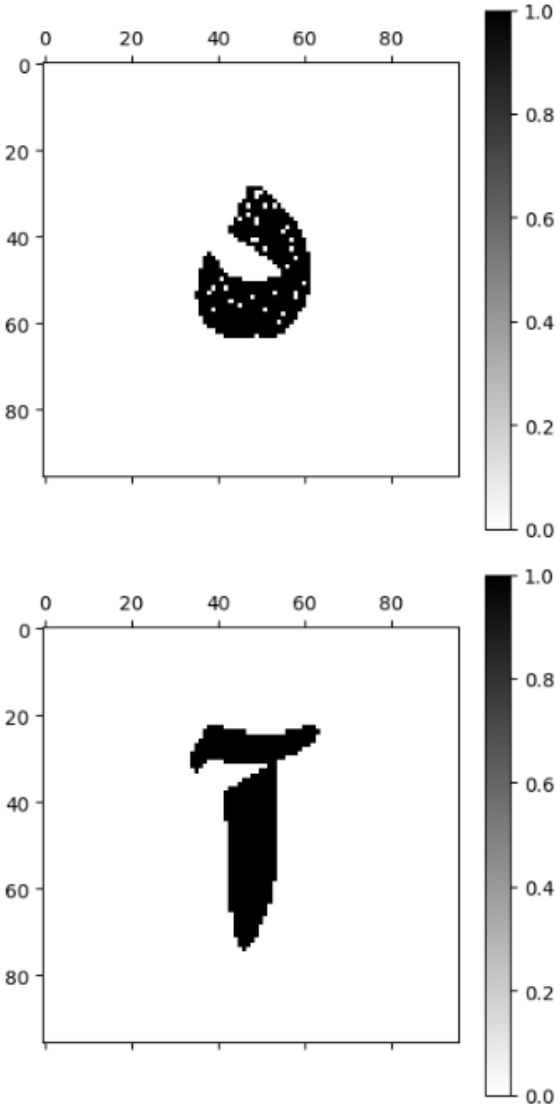


شکل ۱۹: شکل شماره ۱۹



شکل ۲۰: شکل شماره ۲۰

با تغییر تمامی پارامترهای ذکر شده در بالا، مانند a,b,Emax,w,T,e,U می‌توانیم میزان missing point را تغییر بدهیم. اما راحت‌ترین پارامتر که Emax است را تغییر دادیم و از ۰/۰۰۰۰۰۱ به ۰/۰۰۱ تغییر دادیم و مشاهده می‌کنیم که شبکه به اشتباه آن را تشخیص می‌دهد:
برای تغییر میزان missing point داده‌ها می‌توانیم باقی پارامترها را تغییر دهیم تا این مشکل را برطرف کنیم. برای مثال میتوانیم آستانه تابع فعالساز را کم‌تر کنیم تا راحت‌تر تشخیص دهد و یا a و b را کم‌تر کنیم.



شکل ۲۱: شکل شماره ۲۱

۴ یک مجموعه داده برای پیش‌بینی قیمت خانه‌ها را از طریق این پیوند دانلود کنید و مراحل ذکر شده در سوالات بعدی را برای فایل **data.csv** 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.

۱.۴ فایل csv مربوط به این سوال را خوانده و سپس تابع info را از Pandas فراخوانی کنید. تعداد داده‌هایی که Nan هستند را بر حسب هر ستون نمایش دهید و اگر نیاز است دستوراتی برای رفع این مشکل بنویسید.

ابتدا کتابخانه‌های مورد نیاز را فراخوانی می‌کنیم و سپس با تابع gdown دیتافریم خود را لود می‌کنیم و آن را در df می‌ریزیم. با استفاده از تابع info در می‌یابیم که دیتا فریم ما دارای ۴۶۰۰ سطر برای هر ۱۸ ستون هست و نوع دیتای هر ستون را متوجه می‌شویم. برای مثال در ستون ،price داده‌های ما از نوع عدد اعشاری هستند و در ستون yr-built داده‌ها از نوع عدد صحیح هستند و در ستون city داده‌ها از نوع object هستند:

```python
df = pd.read_csv('data.csv')
df.head()

df.info()
```

Code 28: My Caption (Python)

برای مشاهده تعداد داده‌های بر حسب Nan از isnull استفاده می‌کنیم و برای حذف آن می‌توانیم از dropna استفاده کنیم. مشاهده می‌شود که در هیچ ستونی داده‌ی Nan وجود ندارد:

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

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

date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
```

```
<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
```

شکل ۲۲: شکل شماره ۲۲

```
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64
```

Code 29: My Caption (Python)

۲.۴ ماتریس هم‌بستگی را رسم کنید. چه ویژگی‌ای با قیمت هم‌بستگی بیشتری دارد؟

از آنجایی که شهری که خانه در آنجا وجود دارد، نقش مهمی در تعیین میزان قیمت خانه دارد، ابتدا با استفاده از value-counts اسم تمام شهرهای موجود و تعداد تکرار آن‌ها را می‌شماریم. با بررسی خروجی این کار ر می‌یابیم که اشتباه تایپی در نام شهرها وجود ندارد و نیاز به تغییر آن و یکسان سازی نیست. پس برای آنکه بتوانیم هر شهر را با ۰ و ۱ به خانه‌ها اختصاص دهیم از one-hot encoding استفاده می‌کنیم. به دین صورت تمامی شهرها به ستون‌ها اضافه می‌شوند و هر خانه‌ای که در آن شهر باشد، عدد ۱ و اگر در آن شهر نباشد عدد ۰ را می‌گیرد. همچنین چون street و statezip نقش مهمی در تعیین قیمت خانه ندارد، با استفاده از drop این ستون‌ها را از دیتا فریم حذف می‌کنیم. با بررسی دیتافریم در می‌یابیم که مربوط به خانه‌های داخل کشور آمریکا هستند، پس همچنین نیازی به ستون country نداریم و می‌توانیم این ستون را نیز حذف کنیم:

```python
city = df["city"]
city.value_counts()

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

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()
```

Code 30: My Caption (Python)

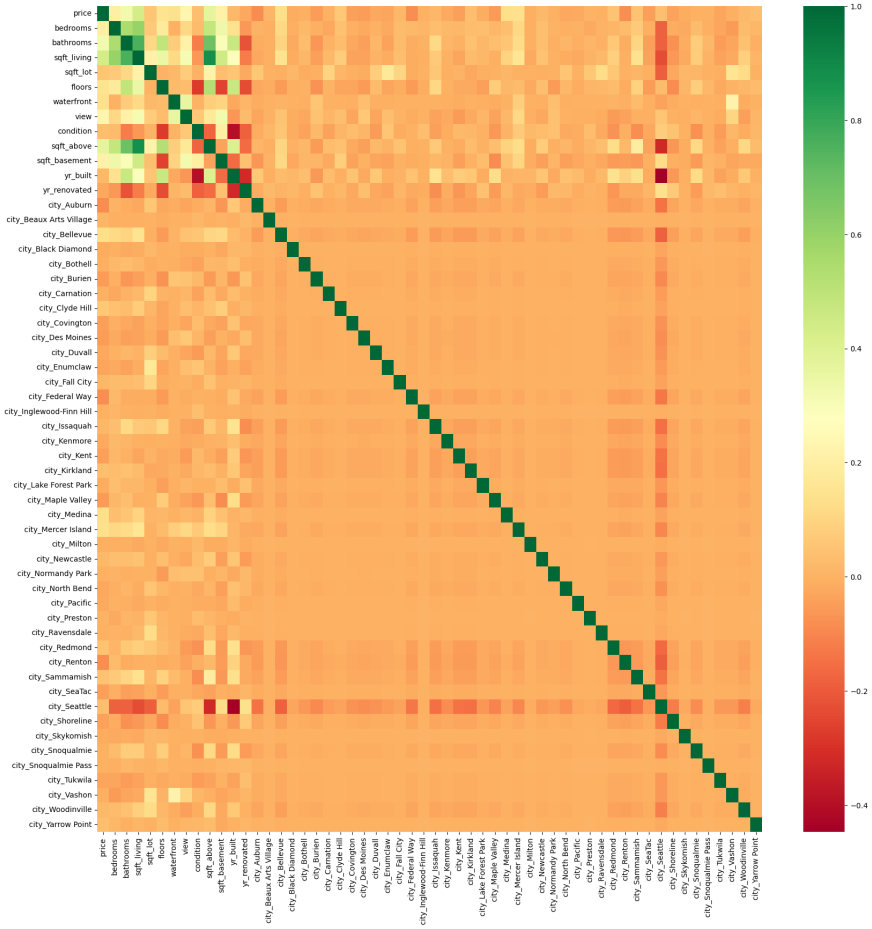
با بررسی ۵ سطر اول دیتافریم می‌بینیم که تعداد ستون‌های فعلی ما به ۵۷ ستون رسیده است و از این به بعد با این دیتافریم کار می‌کنیم.

برای رسم ماتریس هم‌بسنگی به دو صورت می‌توانیم این کار را انجام دهیم. روش اول با استفاده از sns.heatmap است و خروجی آن به شکل زیر می‌شود:

```python
plt.figure(figsize=(20, 20))
sns.heatmap(df2.corr(), cmap="RdYlGn")
plt.show()
```

Code 31: My Caption (Python)

همانطور که در شکل نیز مشاهده می‌شود هرچقدر به رنگ شبز نزدیک‌تر باشیم میزان هم‌بستگی دو پارامتر با یکدیگر بیشتر می‌شود و هز چقدر به رنگ قرمز نزدیک‌تر شویم میزان هم‌بستگی کمتر می‌شود. در قطر اصلی میزان هم‌بستگی دو پارامتر یکسان بررسی می‌شود. پس میزان هم‌بستگی آن ۱ می‌شود. پارامتر بعدی‌ای که با قیمت هم‌بستگی بیشتری دارد sqft-living است. هم‌چنین می‌توانیم با استفاده از کورلیشن، مقدار عددی هم‌بستگی بین ستون قیمت و باقی ستون‌ها را دریابیم:



شکل ۲۳: شکل شماره ۲۳

```python
# Calculate the correlation between columns and 'price', then sort them in
    descending order
correlation_matrix = df2.corr()['price'].sort_values(ascending=False)
correlation_matrix

price                 1.000000
sqft_living           0.430410
sqft_above            0.367570
bathrooms             0.327110
view                  0.228504
```

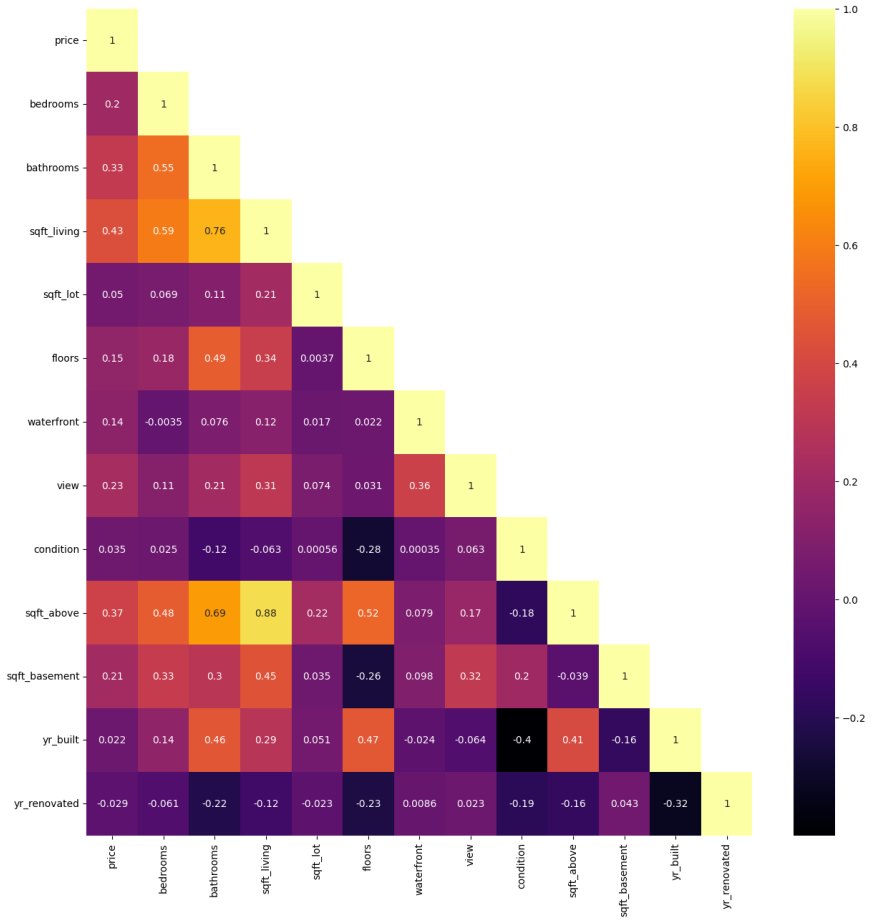
Code 32: My Caption (Python)

روش دوم با استفاده از sns.heatmap است. برای جلوگیری از توهم رفتگی اعداد، با همان df ابتدایی این‌کار را انجام می‌دهیم. برای این‌کار ابتدا ستون‌هایی را که داده‌های آن‌ها object هستند را در num می‌ریزیم و سپس ماتریس هم‌بستگی را به ازای باقی ستون‌ها

رسم می‌کنیم:
در اینجا هر چی به رنگ زرد نزدیک‌تر می‌شویم هم‌بستگی دو ستون بیشتر است و با رنگ زرد به ۱ می‌رسد و هر چه به رنگ مشکی نزدیک‌تر می‌شویم هم‌بستگی کمتر می‌شود و به سمت ۰ می‌رود. و مقدار هم‌بستگی نیز در مربع‌ها نوشته شده است و نتایج بدست آمده مانند قبل است. و مشخص است که قیمت با ستون sqft-living هم‌بستگی بیشتری دارد و مقدار آن ۰/۴۳ است.

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

plt.figure(figsize=(15, 15))
sns.heatmap(df[num].corr(), annot=True, cmap='inferno', mask=np.triu(df[num].
    corr(), k=1))
```

Code 33: My Caption (Python)



شکل ۲۴: شکل شماره ۲۴

۳.۴ نمودار توزیع قیمت و نمودار قیمت و ویژگی‌ای که هم‌بستگی زیادی با قیمت دارد را رسم کنید.

برای رسم این نمودار از plt.scatter استفاده می‌کنیم و محور x آن را ستون sqft-living و محور y آن را ستون price قرار می‌دهیم و خروجی آن به شکل زیر می‌شود:

```
# Create a scatter plot of enginesize against price
plt.figure(figsize=(5, 5))
plt.scatter(x='sqft_living', y='price', data=df2)
plt.xlabel('sqft_living')
plt.title('sqft-living vs. Price')
plt.ylabel('price')
plt.show()
```

Code 34: My Caption (Python)

شکل ۲۵: شکل شماره ۲۵

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

ابتدا با استفاده از pd.to-datetime ساعت را از این ستون حذف می‌کنیم و در ستون جدیدی به نام datetime-column قرار می‌دهیم. سپس ماه و سال را با دستورات زیر در ستون‌های جدید قرار می‌دهیم

```python
# Convert the 'datetime_column' to datetime format
df2['datetime_column'] = pd.to_datetime(df2['date'])

# Extract year and month into separate columns
df2['year'] = df2['datetime_column'].dt.year
df2['month'] = df2['datetime_column'].dt.month

# Display the updated DataFrame
print(df2)
```

Code 35: My Caption (Python)

سپس داده‌های مربوط به دو ستون date و datetime-column را حذف می‌کنیم:

```python
df2 = df2.drop(['datetime_column','date'], axis=1)
```

Code 36: My Caption (Python)

۵.۴ داده‌ها را با نسبت ۸۰ به ۲۰ درصد به مجموعه‌های آموزش و آزمون تقسیم کنیدو داده‌های آموزشی و آزمون را با استفاده از MinMaxScaler مقیاس کنید.

ابتدا با استفاده از labelencoding ستون‌هایی را که تایپ داده‌ی آن string باشد را به مقادیر عددی تبدیل می‌کنیم. سپس همه‌ی ستون‌ها را به جز ستون price به X یا همان ویژگی‌ها اختصاص می‌دهیم و ستون price را به Y. و سپس با نسبت ۰/۲، داده‌ها را به دو بخش آموزش و تست تقسیم می‌کنیم:

```python
# Initialize LabelEncoder
l1 = LabelEncoder()

# Convert object-type columns to numerical using Label Encoding
for i in df2.columns:
    if df2[i].dtype == 'object':
        df2[i] = l1.fit_transform(df2[i])

df2
```

```
10
11 # Separate input (X) and output (Y) data
12 X = df2.drop(["price"], axis=1)  # Input data
13 Y = df2["price"]                  # Output data
14
15 # Perform train-test split
16 x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
       random_state=7)
```

Code 37: My Caption (Python)

سپس با استفاده از ()MinMaxScaler داده‌های آموزش و آزمون را مقیاس بندی می‌کنیم:

```
1 # Initialize Min-Max Scaler
2 scaler_1 = MinMaxScaler()
3
4 # Normalize the training input data
5 x_train = scaler_1.fit_transform(x_train)
6
7 # Normalize the test input data
8 x_test = scaler_1.transform(x_test)
9
10 # Convert y_train and y_test type to DataFrame
11 y_train = pd.DataFrame(y_train)
12 y_test = pd.DataFrame(y_test)
13
14 scaler_2 = MinMaxScaler()
15
16 # Normalize outputs
17 y_train = scaler_2.fit_transform(y_train)
18 y_test = scaler_2.transform(y_test)
```

Code 38: My Caption (Python)

۶.۴ یک مدل **Multi-Layer Perceptron (MLP)** ساده با ۲ لایه پنهان یا بیشتر بسازید. بخشی از داده‌های آموزش را برای اعتبار سنجی کنار بگذارید و با انتخاب بهینه‌ساز و تابع اتلاف مناسب، مدل را آموزش دهید. نمودارهای اتلاف و **R2 Score** را رسم و نتیجه را تحلیل کنید.

ابتدا شبکه عصبی‌ای با ۲ لایه پنهان می‌سازیم. لایه اول با ۵۰ نرون و تابع فعالساز relu و لایه دوم با ۳۰ نرون و تابع فعالساز relu و لایه آخر با یک نرون و تابع فعالساز linear.

```python
model_2 = Sequential()

# Add the first hidden layer with 50 neurons and relu activation function
model_2.add(Dense(50, activation='relu', input_shape=(x_train.shape[1],)))

# Add the second hidden layer with 30 neurons and relu activation function
model_2.add(Dense(30, activation='relu'))

# Add an output layer with 1 neuron and linear activation function
model_2.add(Dense(1, activation='linear'))

model_2.summary()
```

Code 39: My Caption (Python)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                2900

 dense_1 (Dense)             (None, 30)                1530

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 4461 (17.43 KB)
Trainable params: 4461 (17.43 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

شکل ۲۶: شکل شماره ۲۶

سپس با استفاده از شبکه عصبی ایجاد شده مدل را آموزش می‌دهیم. با استفاده از بهینه ساز adam و تابع اتلاف mse. برای آموزش شبکه‌مون x-train و y-train را به عنوان ورودی می‌دهیم و ۰/۲ داده‌ها را برای اعتبارسنجی قرار می‌دهیم. تعداد epoch ها را برابر ۱۰۰ می‌گذاریم و batch-size را ۱۰ قرار می‌دهیم. از batch-size برای آن استفاده می‌کنیم که تمام داده‌ها را برای آموزش با یکدیگر ندهیم و به دسته‌های کوچکتر تقسیم کنیم.

```
model_2.compile(optimizer='adam', loss='mse')
history = model_2.fit(x_train, y_train, validation_split=0.2, epochs=100 ,
    batch_size=10, verbose=0)
```

Code 40: My Caption (Python)

برای محاسبه اتلاف، با استفاده از evaluate مقادیر y-test را پیش‌بینی می‌کنیم و سپس اتلاف را محاسبه می‌کنیم. اتلاف برابر 8.1478e-05 می‌شود. برای محاسبه R۲ از تابع r2-score استفاده می‌کنیم و y-test و y پیش بینی شده‌ی شبکه عصبی خود را مقایسه می‌کنیم و نتیجه برابر ۰/۵۷۰ می‌شود.

```
#Evaluate the model
loss = model_2.evaluate(x_test , y_test)
#loss: 8.1478e-05

y_pred_2 = model_2.predict(x_test)
rscore_2 = r2_score(y_test , y_pred_2)
rscore_2
#0.5704281737140428
```

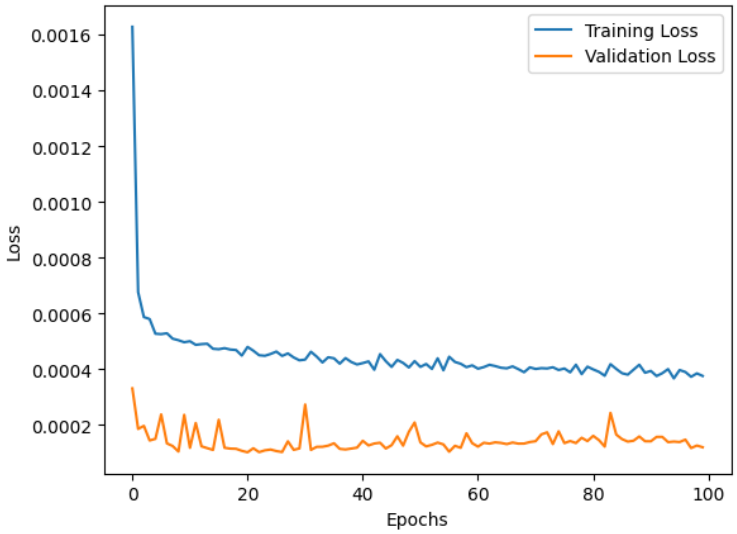
Code 41: My Caption (Python)

برای رسم نمودار اتلاف آموزش و اعتبار سنجی به شکل زیر عمل می‌کنیم و نتیجه به شکل زیر می‌شود:

```
# 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.show()
```

Code 42: My Caption (Python)

طبق نتایج بدست آمده، هر چه R۲score به یک نزدیک‌تر باشد بهتر است و در اینجا عدد ۰/۵۷ بدست آمده است و نشان دهنده‌ی آن است که عملکرد شبکه عصبی ما خوب نمی‌باشد. همچنین با بررسی نمودار اتلاف در می‌یابیم که به دلیل آنکه اتلاف به سمت پایدار شدن نمی‌رود و در عین نزولی بودن همچنان ناپایدار است، شبکه عصبی ما خوب آموزش ندیده است و در مورد داده‌های اعتبار سنجی نیز این موضوع کاملا مشهود است. همچنین با افزایش تعداد epoch ها به نتیجه‌ي مطلوبی نرسیدیم. برای رفع این مشکل می‌توانیم از چندین راه حل استفاده کنیم. برای مثال می‌توانیم لایه‌های پنهان شبکه عصبی را زیاد تر کنیم و تعداد نرون لایه‌ها را افزایش دهیم. همچنین می‌توانیم با تغییر تابع فعالساز و تابع اتلاف و بهینه ساز و آزمون خطا نتیجه بهتری را کسب کنیم.



شکل ۲۷: شکل شماره ۲۷

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

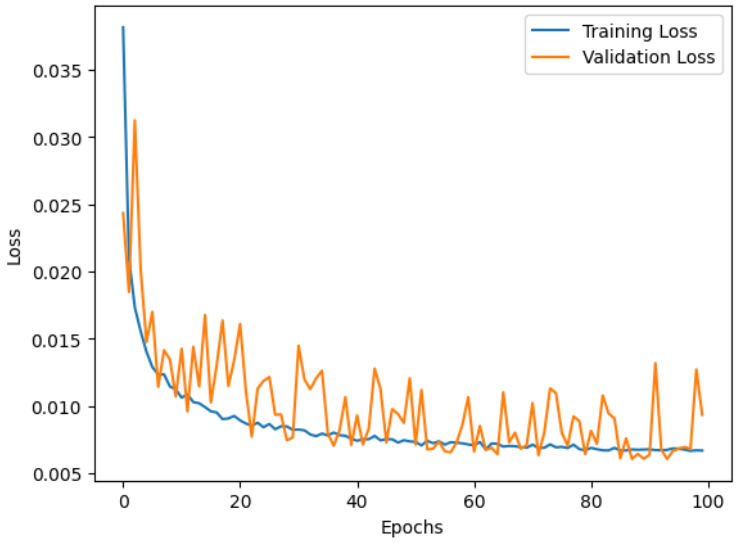
برای این سوال از تابع اتلاف Mean Absolute Error (MAE) و بهینه ساز stochastic gradient descent(sgd) استفاده کردیم و نتایج به شرح زیر شد:

```
# Compile model with stochastic gradient descent optimizer and mean absolute
    error loss
model_2.compile(optimizer = 'sgd',loss = 'mae')

history = model_2.fit(x_train ,y_train ,validation_split=0.2 ,epochs = 100,
    batch_size = 10,verbose = 0)
loss = model_2.evaluate(x_test , y_test)
#loss: 0.0086

y_pred_2 = model_2.predict(x_test)
rscore_2 = r2_score(y_test , y_pred_2)
rscore_2
#0.1630696235488699

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

```

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

Code 43: My Caption (Python)

در این حالت اتلاف افزایش پیدا کرده است و R۲ نیز به قدر زیادی کاهش پیدا کرده اس و عملکرد شبکه عصبی به شدت افت داشته



شکل ۲۸: شکل شماره ۲۸

است. با اینکه نمودار اتلاف داده‌های آموزش در حال کاهش است اما اعتبار سنجی وضعیت خوبی ندارد و همواره در حال افزایش و کاهش است و ناپایدار می‌باشد. این حالت به دلایل زیادی رخ می‌دهد و یکی از علت‌های آن می‌تواند overfitting باشد. که علت آن آن است که مدل بیش از حد پیچیده است و نویز داده‌های آموزش را یاد می‌گیرد و قادر به تعمیم آن الگوها به خوبی نمی‌باشد. علت دیگر آن می‌تواند پیچیدگی مدل باشد. تعداد پارامترهای بیش از حد یا لایه‌های زیادی نسبت به اندازه مجموعه داده، باعث می‌شود که مدل به جای یادگیری الگوهای کلی، به خوبی داده‌های آموزشی را یاد بگیرد. و یا آنکه به مقدار کافی داده نداریم. زیرا داده‌های محدود ممکن است باعث شود که مدل بیش از حد به داده‌های آموزشی پیچیده شود و قادر به تعمیم به خوبی نباشد.

راه حل‌های حل این مشکل شامل:

- تکنیک Regularization: حذف L۱،L۲
- افزایش داده‌ها یا جمع آوری داده‌های متنوع تر
- استپ زودهنگام: برای متوقف کردن آموزش هنگامی که اتلاف اعتبارسنجی بهبود نمی‌یابد.

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

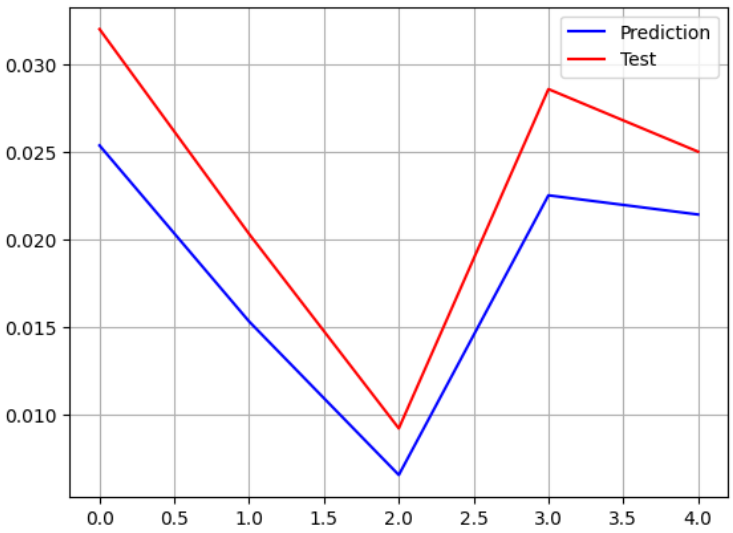
دو لیست خالی برای ذخیره ۵ مقدار پیش‌بینی شده و واقعی ایجاد می‌کنیم. ۵ تا از داده‌ها را در طول دیتافریم به صورت تصادفی انتخاب می‌کنیم و مقادیر رندوم امتخاب شده را به لیست خالی اضافه می‌کنیم و سپس نمودار آن را رسم می‌کنیم:

```
import random

random_pred = list()
random_test = list()

for i in range(5):
    j = random.randint(0, len(y_pred_2) - 1)  # Generate a random index
    random_pred.append(y_pred_2[j])  # Append y_pred_2 value at the random index j
    random_test.append(y_test[j])  # Append y_test value at the same random index j

# Plot the random predictions and actual test outputs
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()
```

Code 44: My Caption (Python)

تفاوت قیمت پیش‌بینی شده با قیمت واقعی:

```
for i in range(5):
    tafavot = random_pred[i] - random_test[i]
    print(tafavot)

[-0.00663961]
[-0.00499544]
```



شکل ۲۹: شکل شماره ۲۹

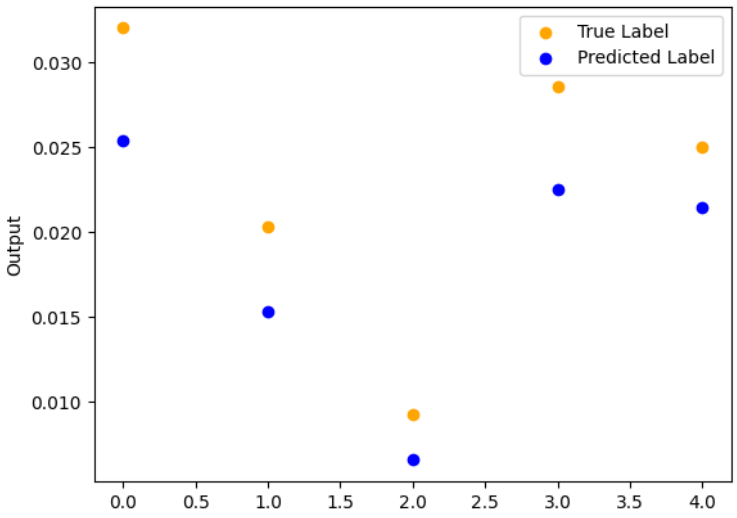
```
[-0.00265931]
[-0.0060627]
[-0.00358737]
```

Code 45: My Caption (Python)

با توجه به این داده‌ها و مقدار تفاوتی که پیش‌بینی شبکه با مقدار واقعی دارد در می‌یابیم که عملکرد شبکه عصبی خوب است و تمامی تفاوت‌ها در حد هزارم اعشار هستند.

```
# Create a scatter plot for true and predicted outputs
plt.scatter(range(len(random_test)), random_test, color="orange")  # True
    labels in orange
plt.scatter(range(len(random_test)), random_pred, color="blue")  # Predicted
    labels in blue

plt.legend(['True Label', 'Predicted Label'])
plt.ylabel("Output")
plt.show()
```

Code 46: My Caption (Python)



شکل ۳۰: شکل شماره ۳۰

برای بهبود عملکرد شبکه می‌توانیم اطمینان حاصل کنیم که ویژگی‌های ورودی مورد استفاده برای آموزش MLP مناسب و اطلاعاتی هستند. با تغییر تعداد لایه‌ها، نورون‌ها در هر لایه، توابع فعال‌سازی یا ساختارهای مختلف مانند شبکه‌های کم عمق و عمیق، به دنبال یافتن ساختار بهینه برای داده‌های خود باشیم. اعمال تکنیک‌های مانند رگولاریزیشن L۱/L۲ یا dropout برای جلوگیری از اورفیت و بهبود تعمیم‌پذیری مدل. همچنین با تنظیم هایپرپارامترهایی مانند نرخ یادگیری، اندازه دسته، بهینه‌ساز، تعداد اپوک و ... با استفاده از تکنیک‌هایی مانند جستجوی شبکه، جستجوی تصادفی یا بهینه‌سازی. پیاده‌سازی توقف زودهنگام برای متوقف کردن آموزش زمانی که عملکرد بر روی مجموعه اعتبارسنجی بهبود نمی‌یابد و جلوگیری از اورفیت. استفاده از روش‌های اعتبارسنجی متقاطع برای ارزیابی عملکرد مدل بر روی زیرمجموعه‌های مختلف داده و اطمینان از تعمیم‌پذیری مدل.

۵ سوال پنجم

۱.۵ مجموعه داده Iris را فراخوانی کنید و روش‌های تحلیل داده‌ای که آموخته‌اید را روی آن ببندید. داده‌ها را با نسبتی دلخواه و مناسب به مجموعه‌های آموزش و ارزیابی تقسیم کنید.

ابتدا دیتاست Iris را لود می‌کنیم و سپس آن را تبدیل به دیتا فریم می‌کنیم و ستون target را که شامل اسم سه گونه‌ی مختلف گل برای این دیتافریم است را به آن اضافه می‌کنیم. تعداد تکرار هر گونه‌ی گل را می‌شماریم و می‌بینیم برای هر گونه ۵۰ داده وجود دارد و تعداد آن‌ها با هم برابر است و نیازی به تغییر تعداد داده‌ها برای همسان سازی نیست. در نهایت با استفاده از one-hot encoding نام‌های موجود در ستون species را به عدد تبدیل می‌کنیم. برای همین تعداد ستون‌های target ما از ۱ به ۳ تغییر می‌کند و نوع آن از string به عدد تبدیل می‌شود.

```
import pandas as pd
from sklearn.datasets import load_iris

```

```python
# Load Iris dataset from scikit-learn
iris_sklearn = load_iris()

# Create a DataFrame from the iris dataset
iris = pd.DataFrame(data=iris_sklearn.data, columns=iris_sklearn.feature_names
    )

# Add target (species) column to DataFrame
iris['species'] = iris_sklearn.target_names[iris_sklearn.target]

# Display the count of each species before encoding
print(iris['species'].value_counts())

# Convert categorical columns to numerical using one-hot encoding
df = pd.get_dummies(iris, columns=['species'])

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

Code 47: My Caption (Python)

در نهایت چهار ستون ابتدایی را به ویژگی‌ها یا X و سه ستون نهایی را به y اختصاص می‌دهیم و داده‌ها را با نسبه ۲۰ به ۸۰ به مجموعه داده‌های آموزش و ارزیابی تقسیم می‌کنیم:

```python
X = df.iloc[:,0:4]
y = df.iloc[:,4:]

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Code 48: My Caption (Python)

یا می‌توانیم از روش زیر استفاده کنیم که در این صورت ستون target های ما بر اساس اعداد هست و به یه کلاس طبقه بندی شده اند. از اینجا به بعد با این روش کار می‌کنیم:

```python
# Load Iris dataset from scikit-learn
iris_sklearn = load_iris()
X = iris_sklearn.data
y = iris_sklearn.target

# Splitting the data into training and testing sets
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=83)
```

Code 49: My Caption (Python)

۲.۵ با استفاده از روش‌های آماده پایتون، سه مدل بر مبنای رگرسیون لجستیک، MLP و شبکه‌های عصبی پایه شعاعی (RBF) را تعریف کرده و روی داده‌ها آموزش دهید. نتایج روی داده‌های ارزیابی را حداقل با چهار شاخص و ماتریس درهم‌ریختگی نشان داده و تحلیل کنید. در انتخاب فراپارامترها آزاد هستید؛ اما لازم هست که نتایج را به صورت کامل مقایسه و تحلیل کنید.

ابتدا مدل خود را با روش mlp می‌سازیم. مدل ساخته شده‌ی ما برای این قسمت یک شبکه عصبی با یک لایه و ۱۰۰ نرون است و تابع فعالساز آن ،relu بهینه‌ساز آن ،adam ضریب یادگیری آن ۰/۰۰۱ در نظر گرفته شده است و باقی پارامترهای آن به شرح زیر است. همچنین برای این داده‌ها، با تعداد لایه‌های بیشتری نیز داده‌ها را فیت کردیم، اما score آن کاهش یافت. برای همین یک لایه را انتخاب کردیم.

```
model = MLPClassifier(hidden_layer_sizes=(200), activation='relu', solver='
    adam',
                      alpha=0.0001, batch_size='auto', learning_rate='constant
    ', learning_rate_init=0.001,
                      power_t=0.5, max_iter=200, shuffle=True, random_state
    =83, tol=0.0001, verbose=False,
                      warm_start=False, momentum=0.9, nesterovs_momentum=True,
     early_stopping=False, validation_fraction=0.1,
                      beta_1=0.9, beta_2=0.999, epsilon=1e-08,
    n_iter_no_change=10, max_fun=15000)

model.fit(x_train, y_train)
model.score(x_test, y_test)

#1.0
```

Code 50: My Caption (Python)

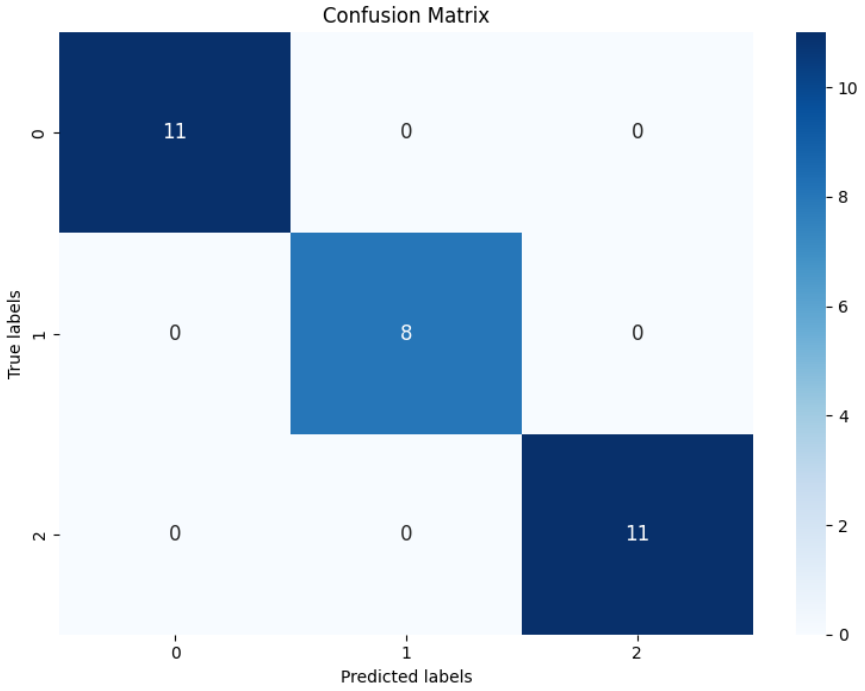
برای روش MLP ابتدا روی داده‌های ارزیابی، پیش‌بینی می‌کنیم و سپس ماتریس هم‌بستگی را بر اساس پیش‌بینی شبکه و مقادیر ارزیابی محاسبه می‌کنیم. سپس با استفاده از sns ماتریس درهم‌ریختگی را با رنگ آبی نمایش می‌دهیم. و در نهایت تصویر آن را به عنوان png ذخیره می‌کنیم و نمایش می‌دهیم. با استفاده از classification-report با چند شاخص مختلف نتایج ارزیابی را نمایش می‌دهیم:

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report
```

```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Making predictions on the test set
y_pred = model.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_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')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

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

Code 51: My Caption (Python)

با مشاهده ماتریس در هم ریختگی در می‌یابیم که به ازای ۱۱ داده‌ای که باید آن را کلاس ۰ تشخیص می‌داد، درست عمل کرده است و آن‌ها را کلاس ۰ تشخیص داده است. همچنین به ازای ۸ داده‌ای که متعلق به کلاس ۱ بوده‌اند نیز درست عمل کرده و درست تشخیص داده است. همچنین برای کلاس سوم نیز درست تشخیص داده است و خطایی وجود نداشته است.



شکل ۳۱: شکل شماره ۳۱

Classification Report:	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	1.00	1.00	8
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

شکل ۳۲: شکل شماره ۳۲

اطلاعاتی که classification-report به ما می‌دهد، به شرح زیر است:

- Precision: دقت پیش‌بینی‌های مثبت را اندازه‌گیری می‌کند. این نسبت مشاهدات مثبت پیش بینی شده صحیح به کل مثبت های پیش بینی شده است.
- Recall (Sensitivity/True Positive Rate): نسبت مثبت های واقعی را که به درستی پیش بینی شده بودند محاسبه می کند. این نسبت مشاهدات مثبت پیش بینی شده درست به همه مثبت های واقعی است.
- F1-Score: میانگین هارمونیک دقت و یادآوری است. تعادلی بین دقت و یادآوری ایجاد می کند. امتیاز F۱ به بهترین مقدار خود در ۱ (دقت و یادآوری کامل) و بدترین مقدار در ۰ می رسد.

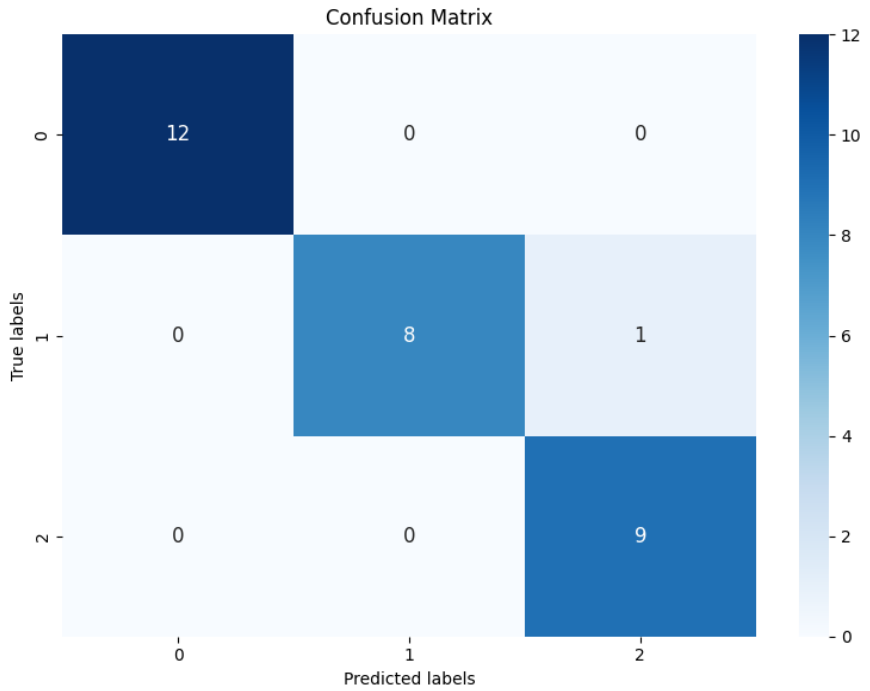
- Support: تعداد داده‌هایی که در هر کلاس پیش‌بینی شده‌اند.
- accuracy: نشان دهنده دقت مدل در کل مجموعه تست است.
- macro avg: میانگین وزن نشده Precision, Recall, F1-Score در کلاس‌ها است.
- weighted avg: میانگین وزن شده Precision, Recall, F1-Score در کلاس‌ها است که با support وزن دار شده است.

برای روش LogisticRegression تعداد iter ها را برابر ۲۰۰ و randomstate را روی ۸۳ تنظیم می‌کنیم و مانند روش بالا ماتریس در هم‌ریختگی را نمایش می‌دهیم:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Assuming you've trained your model already
model = LogisticRegression(max_iter=200, random_state=83)
model.fit(x_train, y_train)

# Making predictions on the test set
y_pred = model.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_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')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

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

Code 52: My Caption (Python)



شکل ۳۳: شکل شماره ۳۳

در اینجا به ازای ۱۲ داده‌ای که برای تست کردن کلاس ۰ استفاده شده است، همه‌ی ۱۲ تا داده درست طبقه بندی شده‌اند و برای ۹ داده‌ای که برای کلاس ۱ استفاده شده‌اند، ۸ تای آن‌ها درست پیش‌بینی شده بودند اما یکی از آن‌ها به اشتباه کلاس ۲ پیش‌بینی شده است. برای کلاس ۲، ۹ داده که مربوط به کلاس۲ بوده‌اند درست تشخیص داده‌شده اند و همان داده‌ی مربوط به کلاس۱ اشتباه شده است.

برای حالت rbf نتایج به صورت زیر است:

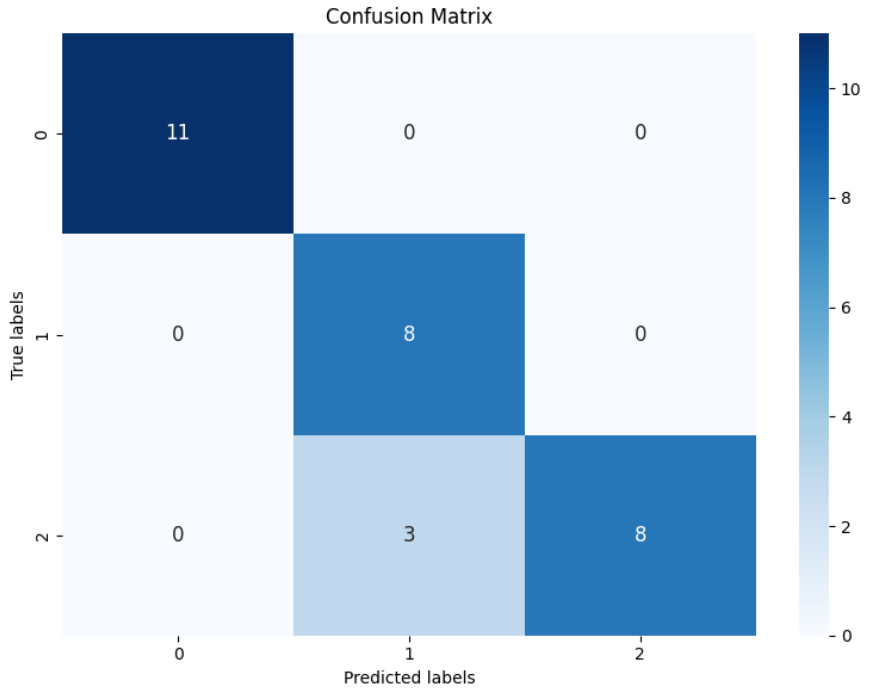
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      0.89      0.94         9
           2       0.90      1.00      0.95         9

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.96        30
weighted avg       0.97      0.97      0.97        30
```

شکل ۳۴: شکل شماره ۳۴

```python
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split


# Assuming you've trained your model already
model = SVC(kernel='rbf', random_state=83)
model.fit(x_train, y_train)

# Making predictions on the test set
y_pred = model.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})

```

```python
# Get the axis to modify layout
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')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()

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

Code 53: My Caption (Python)



شکل ۳۵: شکل شماره ۳۵

در این حالت به ازای ۱۱ داده‌ای که برای ارزیابی کلاس ۰ استفاده شده‌اند، تمامی ۱۱ مقدار آن درست پیش‌بینی شده‌اند و برای کلاس

۱، ۸ داده وجود داشته‌اند که هر ۸ تای آن درست پیش‌بینی شده‌اند. اما برای کلاس ۲، ۳ داده وجود دارند که به اشتباه به عنوان کلاس ۱ طبقه‌بندی شده‌اند. به همین دلیل مقدار persicion برای کلاس ۱ به مقدار ۰/۷۳ رسیده است و شاخص recall برای کلاس۲ به مقدار ۰/۷۳ رسیده است.

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	0.73	1.00	0.84	8
2	1.00	0.73	0.84	11
accuracy			0.90	30
macro avg	0.91	0.91	0.89	30
weighted avg	0.93	0.90	0.90	30

شکل ۳۶: شکل شماره ۳۶

همانطور که از نتایج مشخص است، روش MLP به نسبت دو روش دیگر نتایج بهتری را به دنبال داشته است.

۳.۵ به دانشجویانی که این سوال را بدون استفاده از کتابخانه‌ها و مدل‌های آماده پایتونی انجام دهند، تا ۲۰ درصد نمره امتیازی تعلق خواهد گرفت.

برای MLPclassifier به شکل زیر عمل می‌کنیم. ابتدا پارامترهای مورد نیاز را با self برای کلاس خود تعریف می‌کنیم و سپس توابعی را که نیاز داریم می‌سازیم. قسمت forward-pass ورودی و خروجی لایه پنهان را با استفاده از تابع فعال سازی سیگموئید و خروجی لایه خروجی را با استفاده از تابع فعال سازی softmax محاسبه می کند. قسمت backward-pass گرادیان را محاسبه می کند و وزن ها و بایاس ها را برای لایه های مخفی و خروجی بر اساس خطای بین مقادیر پیش بینی شده و واقعی به روز می کند. قسمت fit برچسب‌های هدف را یک‌بار کدگذاری می‌کند، forward-pass را انجام می‌دهد، و سپس backward-pass را برای تعداد معینی از دوره‌ها اجرا می‌کند. و در نهایت قسمت predict برای داده‌های ورودی‌اش لیبل کلاس‌ها را پیش‌بینی می‌کند.

کلاس MLPClassifier یک پیاده‌سازی ابتدایی از یک شبکه عصبی پیشرو (MLP) برای وظایف طبقه‌بندی استفاده می‌کند، با استفاده از توابع فعال‌سازی sigmoid و softmax، فرآیندهای پیشرو و عقب‌انتشاری برای آموزش و پیش‌بینی. تغییرات و بهبودهای می‌تواند برای مسائل پیچیده‌تر یا بهبود عملکرد، مانند اضافه کردن لایه‌های بیشتر، تغییر توابع فعال‌سازی یا استفاده از تکنیک‌های بهینه‌سازی متفاوت، اعمال شود. که به دلیل طولانی بودن کد آن در گزارش کار آن را نیاورده‌ام.

```
mlp = MLPClassifier(input_size=X.shape[1], hidden_size=64, output_size=3,
    learning_rate=0.01, num_epochs=1000)

mlp.fit(x_train, y_train)

predictions = mlp.predict(x_test)

```

```
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
print(classification_report(y_test, predictions))

cf_matrix = confusion_matrix(y_test, predictions)

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)
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()
```

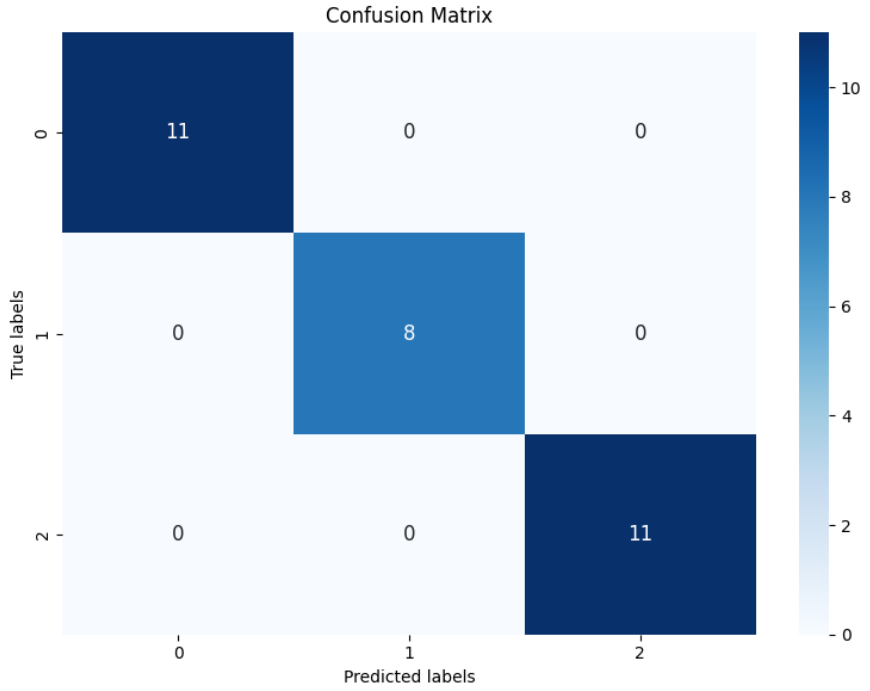
Code 54: My Caption (Python)

خروجی شبکه:

در روش دوم از روش LogisticRegression استفاده می‌کنیم. در این کد ابتدا پارامترهای خود را انتخاب می‌کنیم و سپس تابع مورد نیاز خود را تعریف می‌کنیم. با استفاده از متد fit شبکه خود را آموزش می‌دهیم. وزن ها و بایاس ها را مقدار دهی اولیه می کند، سپس گرادیان نزول را برای تعداد مشخصی از تکرارها انجام می دهد. در هر تکرار، مدل خطی (linear-model) را با ضرب ورودی X در وزن ها و اضافه کردن بایاس محاسبه می کند. با استفاده از تابع سیگموئید احتمالات پیش بینی شده را محاسبه می کند. گرادیان نزول برای به روز رسانی وزن ها (self.weights) و bias (self.bias) بر اساس گرادیان تابع اتلاف استفاده می شود. در نهایت متد predict مدل خطی را محاسبه می کند، تابع سیگموید را برای بدست آوردن احتمالات اعمال می کند و سپس این احتمالات را با آستانه گذاری در ۰.۵ به برچسب های کلاس تبدیل می کند.

این کد یک پیاده‌سازی ابتدایی از رگرسیون لجستیک برای وظایف دسته‌بندی دودویی فراهم می‌کند. این شامل متدهایی برای مقدمه‌سازی مدل، آموزش از طریق نزول گرادیان، و انجام پیش‌بینی‌ها بر اساس وزن‌ها و بایاس‌های یادگرفته شده است. تنظیمات و بهبودهایی می‌توان برای دیتاست‌های مختلف یا بهبود عملکرد مدل اعمال کرد، مانند مدیریت رگولارسیون، ادغام واژه‌های اعمالی، یا گسترش آن برای مدیریت دسته‌بندی چندکلاسه.

```
import numpy as np

class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
```



شکل ۳۷: شکل شماره ۳۷

Accuracy: 1.0

	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	1.00	1.00	8
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

شکل ۳۸: شکل شماره ۳۸

```
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
        self.bias = None

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

```

```
    def fit(self, X, y):
        num_samples, num_features = X.shape
        self.weights = np.zeros(num_features)
        self.bias = 0

        # Gradient Descent
        for _ in range(self.num_iterations):
            linear_model = np.dot(X, self.weights) + self.bias
            y_predicted = self.sigmoid(linear_model)

            # Gradient calculation
            dw = (1 / num_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / num_samples) * np.sum(y_predicted - y)

            # Update weights and bias
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self.sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        return y_predicted_cls
```

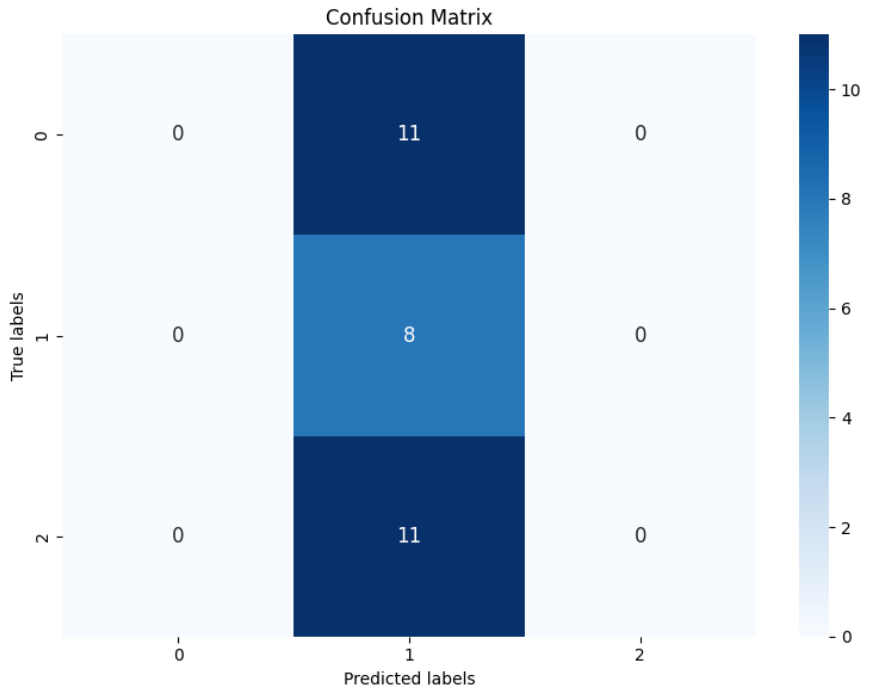
Code 55: My Caption (Python)

محاسبه ماتریس درهم‌ریختگی:

```
model = LogisticRegression(learning_rate=0.1, num_iterations=1000)
model.fit(x_train, y_train)

predictions = model.predict(x_test)

from sklearn.metrics import accuracy_score, classification_report

print("Accuracy:", accuracy_score(y_test, predictions))
print("Classification Report:\n", classification_report(y_test, predictions))

```

```python
cf_matrix = confusion_matrix(y_test, predictions)

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)
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()
```

Code 56: My Caption (Python)



شکل ۳۹: شکل شماره ۳۹

همانطور که از نتایج مشخص است عملکرد این شبکه به شدت ضعیف است و هر ۱۱ داده‌ی کلاس ۰ و کلاس ۲ به اشتباه برای کلاس ۱ پیش‌بینی شده‌اند.

```
Accuracy: 0.26666666666666666
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        11
           1       0.27      1.00      0.42         8
           2       0.00      0.00      0.00        11

    accuracy                           0.27        30
   macro avg       0.09      0.33      0.14        30
weighted avg       0.07      0.27      0.11        30
```

شکل ۴۰: شکل شماره ۴۰

روش بعدی RBF است. برای این روش متغیرها و پارامترهای مدل اولیه‌سازی می‌شوند. این متد محاسبه تابع پایه RBF را انجام می‌دهد. این متد شامل فرآیند آموزش مدل با استفاده از داده‌های ورودی X و برچسب‌ها y است. این فرایند شامل انتخاب تصادفی مراکز برای RBF، محاسبه پارامتر پراکندگی beta، محاسبه فعال‌سازی‌های RBF برای داده‌های آموزشی و اضافه کردن ترم bias به داده‌های ورودی می‌شود. این فرآیند شامل محاسبه فعال‌سازی‌های RBF برای داده‌های تست و انجام پیش‌بینی با استفاده از وزن‌های یادگرفته شده می‌باشد.

این کد یک شبکه عصبی ساده با توابع پایه RBF را پیاده‌سازی می‌کند که برای مسائل رگرسیون به کار می‌رود. فرآیند آموزش شبکه شامل انتخاب مراکز RBF، محاسبه پارامتر پراکندگی، و آموزش با استفاده از نزول گرادیان است. این مدل می‌تواند برای پیش‌بینی‌ها در داده‌های تست استفاده شود.

```python
rbf = RBFNetwork(num_centers=10, learning_rate=0.01, num_epochs=1000)
rbf.fit(x_train, y_train)

predictions = rbf.predict(x_test)

mse = np.mean((predictions - y_test) ** 2)
print("Mean Squared Error:", mse)

#Mean Squared Error: 0.035046537910832905
```

Code 57: My Caption (Python)