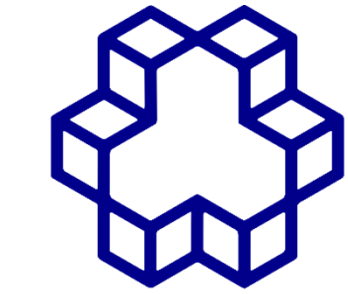
بسمه تعالی



دانشگاه صنعتی خواجه نصیرالدین طوسی

نام و شماره دانشجویی :

علی عبدی

9728463

عنوان :

مینی پروژه دوم

تشریح مساله و کدها

سوال اول

بخش اول

❖ ابتدا کتابخانه های مورد نیاز خود را بارگذاری میکنیم:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
```

❖ حال دیتاست خود را با دستور gdown وارد میکنیم:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1r4RiNjXAlTfbNX3xGsOF1szm1VeTc0BG
```

❖ حال داده‌های خود را به فرمت دیتافریم درمی‌آوریم:

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

	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

400 rows × 3 columns

❖ حال داده های خود را به دو گروه ویژگی ها و هدف تقسیم کرده و سپس در خروجی داده‌های 1 و 1- را به 1 و 0 تبدیل میکنیم و در آخر به نسبت 80 به 20 داده‌های خود را به دو دسته آموزش و ارزیابی تقسیم میکنیم:

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

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

# Splitting the dataset into the Training set and Test set (80/20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
y
```

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

❖ حال دو نوع تابع اتلاف bce و mse تعریف کرده، همچنین برای معیار accuracy نیز تابع مورد نظر را تعریف میکنیم:

Loss

```
[ ]  def bce(y, y_hat):
         eps = np.finfo(float).eps
         y_hat = np.clip(y_hat, eps, 1 - eps)
         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)
```

Accuracy

```
[ ]  def accuracy(y, y_hat):
         acc = np.sum(y == y_hat) / len(y)
         return acc
```

❖ حال کلاس نرون مورد نظر خود را که بر مبنای پرسپترون نوشته باید شود مینویسیم و باید توجه داشته باشیم که تابع فعالسز در این نرون بر مبنای مقدار آستانه‌ای است که ما تعریف میکنیم و اگر عدد مورد نظر از آستانه بزرگتر یا مساوی باشد عدد یک و در غیر این صورت عدد صفر میدهد؛ علاوه بر این در این کلاس ما ورودی ها را رجیستر کرده و متدها را تعریف میکنیم از جمله تابع فعالساز و متد fit و predict ؛ این رون وزن‌ها و بایاس را براساس گرادیان نزولی بروز میکند و کدهای ما بصورت زیر است:

```
class Neuron:

    def __init__(self, in_features, threshold=-18, loss_fn=mse,
n_iter=100, eta=0.1, verbose=True):
        self.in_features = in_features
        # weight & bias
        self.w = np.random.randn(in_features, 1)
        self.b = np.random.randn()
        self.threshold = threshold
        self.loss_fn = loss_fn
        self.loss_hist = []
        self.w_grad, self.b_grad = None, None
        self.n_iter = n_iter
        self.eta = eta
        self.verbose = verbose

    def activation_function(self, z):
        return np.where(z > self.threshold, 1, 0)

    def predict(self, x):
        # x: [n_samples, in_features]
        y_hat = x @ self.w + self.b
        y_hat = self.activation_function(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 % 1 == 0):
                print(f'Iter={i}, Loss={loss:.4}')

    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.threshold})'

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

❖ حال یک آبجکت از مدل خود ساخته و تعداد ایپاک را برابر 100 گرفته و تابع اتلاف را از نوع bce تنظیم کرده و سپس داده‌های خود را با آن آموزش میدهیم و سپس با متد parameters مقدار وزن‌ها و بایاس به ما داده میشود؛ همچنین زمانی که verbose برابر True باشد مقدار اتلاف در هر مرحله را میدهد:

```
neuron = Neuron(in_features=2, loss_fn=bce, n_iter=100, eta=0.1, verbose=True)
perceptron_model=neuron.fit(X_train, y_train[:, None])
neuron.parameters()
```

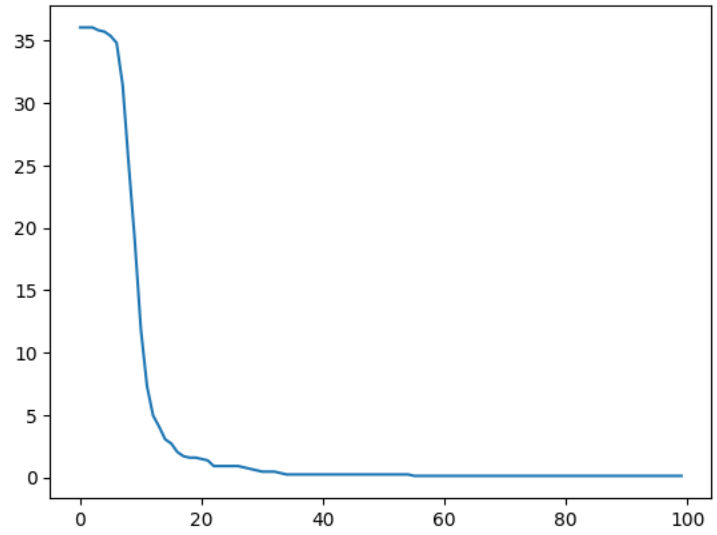
❖ و خروجی کد:

```
Iter=68, Loss=17.57
Iter=69, Loss=17.57
Iter=70, Loss=17.57
Iter=71, Loss=17.57
Iter=72, Loss=17.57
Iter=73, Loss=17.57
Iter=74, Loss=17.57
Iter=75, Loss=17.57
Iter=76, Loss=17.57
Iter=77, Loss=17.57
Iter=78, Loss=17.57
Iter=79, Loss=17.57
Iter=80, Loss=17.57
Iter=81, Loss=17.57
Iter=82, Loss=17.46
Iter=83, Loss=17.46
Iter=84, Loss=17.46
Iter=85, Loss=17.46
Iter=86, Loss=17.46
Iter=87, Loss=17.46
Iter=88, Loss=17.46
Iter=89, Loss=17.23
Iter=90, Loss=17.23
Iter=91, Loss=17.23
Iter=92, Loss=17.23
Iter=93, Loss=17.12
Iter=94, Loss=17.01
Iter=95, Loss=16.9
Iter=96, Loss=16.9
Iter=97, Loss=16.78
Iter=98, Loss=16.22
Iter=99, Loss=16.22
{'w': array([[-4.5459396 ],
       [-4.72529269]]),
 'b': -5.1853210692644796}
```

❖ حال نمودار تابع اتلاف خود را رسم میکنیم:

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

```
[<matplotlib.lines.Line2D at 0x7d9c817f7b50>]
```



بخش دوم

❖ حال مقدار خروجی پیشبینی شده را روی داده‌های تست بدست آورده و شاخص accuracy را محاسبه میکنیم؛ میبینیم که عدد مناسبی بدست آمده:

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

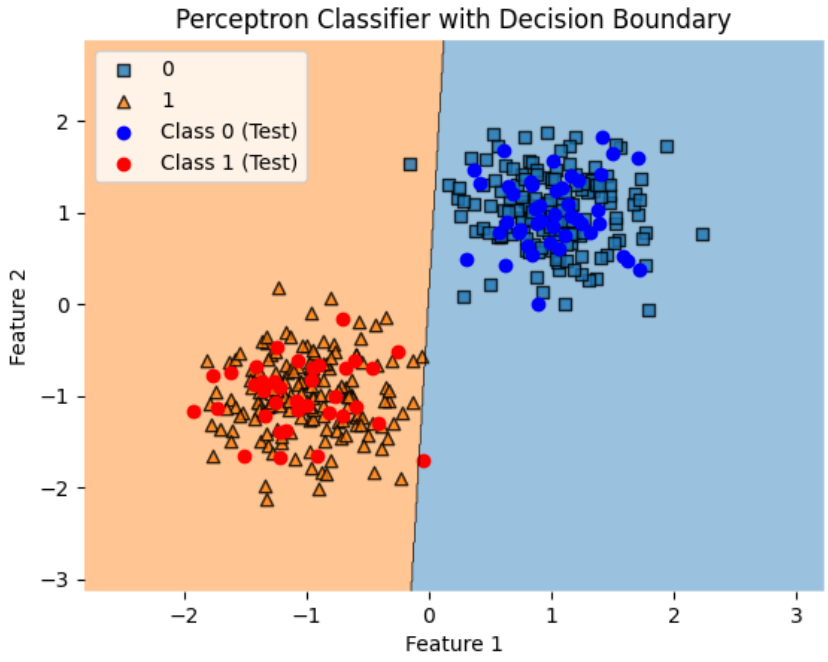
```
0.9875
```

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

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

❖ حال خطی که کار کلاس‌بندی را انجام میدهد با دستور زیر رسم کرده و خروجی را مشاهده میکنیم:

```
] plot_decision_regions(X_train, y_train, clf=neuron, legend=2)

  # Scatter plot for test data
  plt.scatter(X_test[y_test == 0][:, 0], X_test[y_test == 0][:, 1], c='blue', marker='o', label='Class 0 (Test)')
  plt.scatter(X_test[y_test == 1][:, 0], X_test[y_test == 1][:, 1], c='red', marker='o', label='Class 1 (Test)')

  plt.xlabel('Feature 1')
  plt.ylabel('Feature 2')
  plt.title('Perceptron Classifier with Decision Boundary')
  plt.legend()
  plt.show()
```



بخش سوم

در این بخش کدها همان کدهای بالا میباشد و فقط مقدار آستانه را تغییر میدهیم؛ حال مقدار خروجی پیشبینی شده را روی داده‌های تست بدست آورده و شاخص accuracy را محاسبه میکنیم:

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

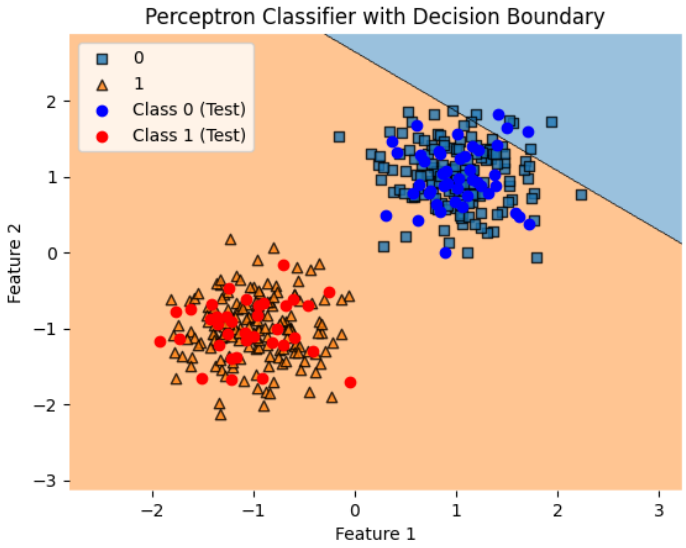
```
0.4875
```

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

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

❖ متاسفانه مشاهده میشود که دقت مدل ما براساس آستانه‌ای که انتخاب میکنیم میتواند بسیار پایین بیاید؛ همچنین خط کلاس‌بندی را رسم میکنیم:

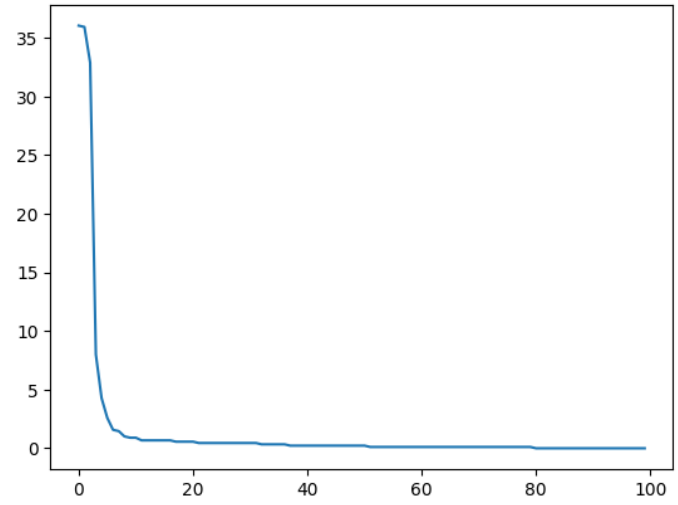
```
plot_decision_regions(X_train, y_train, clf=neuron, legend=2)

# Scatter plot for test data
plt.scatter(X_test[y_test == 0][:, 0], X_test[y_test == 0][:, 1], c='blue', marker='o', label='Class 0 (Test)')
plt.scatter(X_test[y_test == 1][:, 0], X_test[y_test == 1][:, 1], c='red', marker='o', label='Class 1 (Test)')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Perceptron Classifier with Decision Boundary')
plt.legend()
plt.show()
```



- همانطور که در شکل بالا دیده میشود خط مورد نظر بخوبی داده‌های مربوط به دو کلاس را جدا نمیکند.
- حال بایاس را حذف میکنیم و با تغییرات مقدار آستانه میخواهیم به بیشترین دقت برسیم ابتدا تابع اتلاف در حالت حذف بایاس و انتخاب بهترین آستانه را رسم میکنیم:

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

```
[<matplotlib.lines.Line2D at 0x7d9c8156c6a0>]
```



طبق نمودار بعد از حدود 20 ایپاک خطای ما به صفر میل خواهد کرد؛ حال مقدار خروجی پیشبینی شده را روی داده‌های تست بدست آورده و شاخص accuracy را محاسبه میکنیم:

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

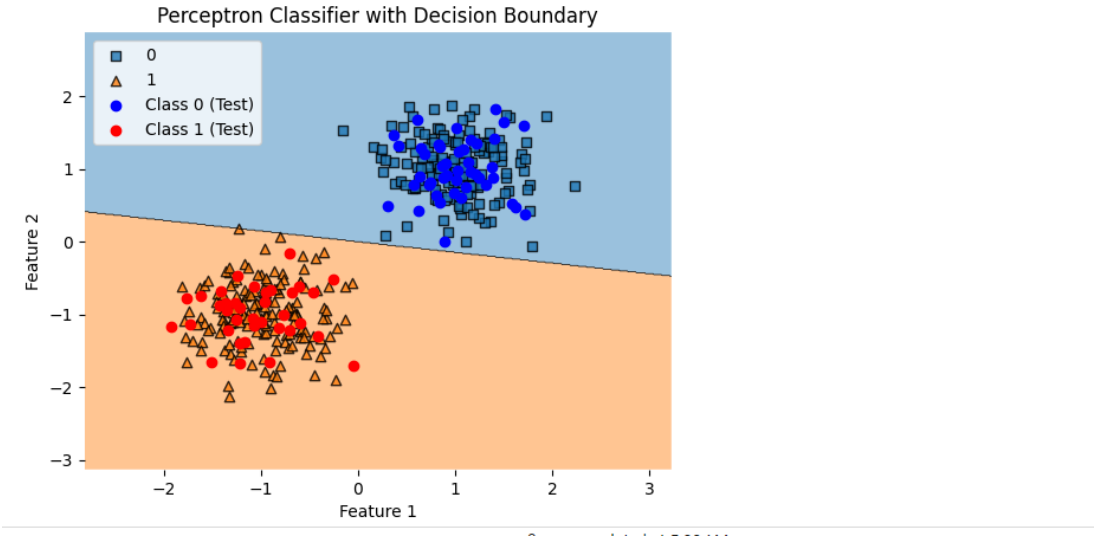
```
1.0
```

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

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

حال خط کلاس‌بندی را رسم میکنیم:

```
plot_decision_regions(X_train, y_train, clf=neuron, legend=2)

# Scatter plot for test data
plt.scatter(X_test[y_test == 0][:, 0], X_test[y_test == 0][:, 1], c='blue', marker='o', label='Class 0 (Test)')
plt.scatter(X_test[y_test == 1][:, 0], X_test[y_test == 1][:, 1], c='red', marker='o', label='Class 1 (Test)')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Perceptron Classifier with Decision Boundary')
plt.legend()
plt.show()
```



- همانطور که میزان دقت مدل را در این حالت بدست آورده‌ایم و شکل بالا نیز نشان میدهد که زمانی که تابع فعالساز به صورت آستانه‌ای تعریف میشود میتواند با تعیین مناسب مقدار آستانه نبود بایاس را جبران کرده و عملیات کلاس‌بندی را بخوبی انجام دهد.

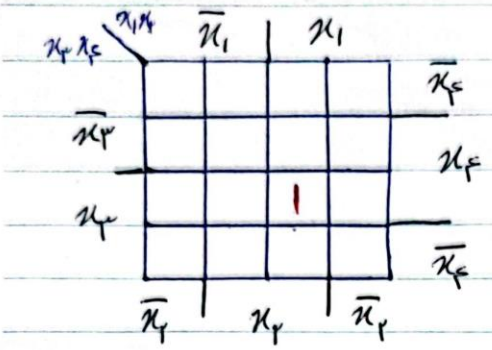
سوال دوم

- در این سوال یک ضرب کننده باینری به کمک نورون مک کلاچ پیتز طراحی خواهیم کرد تا دو عدد دو بیتی دریافت کند و آنها را در هم ضرب کرده و یک خروجی چهار بیتی بدهد توضیحات زیر بصورت کاملتر میباشد:

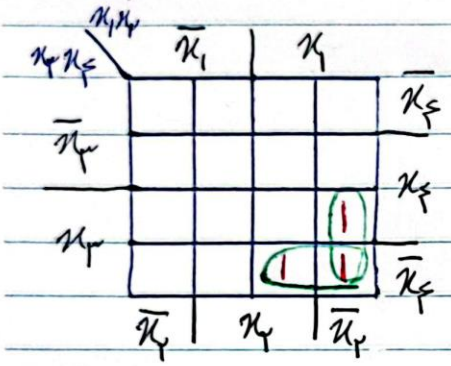
میخواهیم یک ضرب کننده باینری بسازیم که دو ورودی دریافت کند که هرکدام از این عددهای ورودی خودشان دو بیتی هستند بصورت $(\overline{x_1 x_2})_2$ و $(\overline{x_3 x_4})_2$ که در مبنای دو هستند و خروجی بصورت چهار بیتی می باشد بصورت $(\overline{y_1 y_2 y_3 y_4})_2$ که خروجی هم یک عدد در مبنای دو است؛ برای ساخت این ضرب کننده ابتدا باید جدول درستی خود را بصورت زیر بنویسیم: (توجه کنیم که 2^4 حالت داریم)

x_1	x_2	x_3	x_4	y_1	y_2	y_3	y_4
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	1	0	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

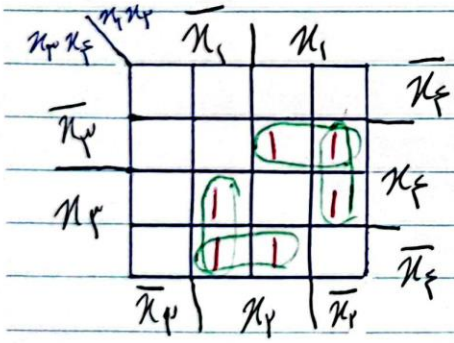
حال برمبنای جدول درستی، جدول کارنو هر بیت خروجی را می نویسیم:



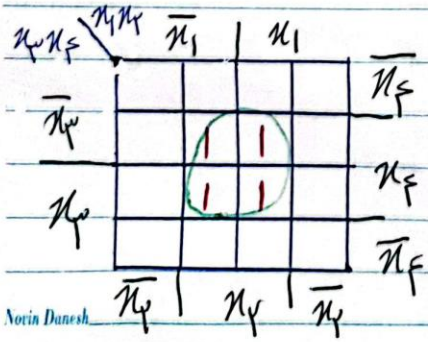
$$\Rightarrow Y_1 = x_1 x_2 x_3 x_4$$



$$\Rightarrow Y_2 = x_1 x_3 \bar{x}_2 + x_1 x_3 \bar{x}_4$$



$$\Rightarrow Y_3 = x_1 x_4 \bar{x}_2 + x_1 x_4 \bar{x}_3 + x_2 x_3 \bar{x}_4 + x_2 x_3 \bar{x}_1$$



$$\Rightarrow Y_4 = x_2 x_4$$

حالا که هر بیت خروجی را بدست آورده ایم میخواهیم آنهارا با گیت های NaNd ، NoR و Not پیاده سازی کنیم ؛ ضرایب مربوط به این گیت ها بصورت زیر از قبل بدست آورده ایم:

Not ⟶ $-۳x = -۱,۵$ ⟵ مقدار آستانه

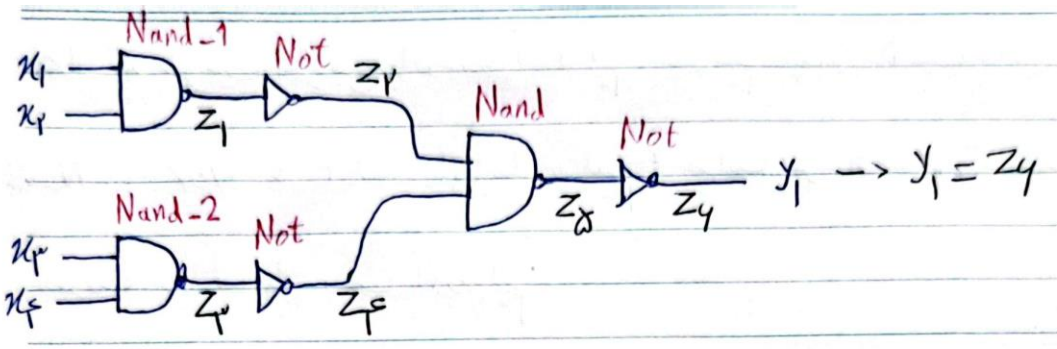
Nand ⟶ $-x_۱ - x_۲ = -۱,۵$ ⟵ ضرایب ورودی

NoR ⟶ $-۳x_۱ - ۳x_۲ = -۱,۵$

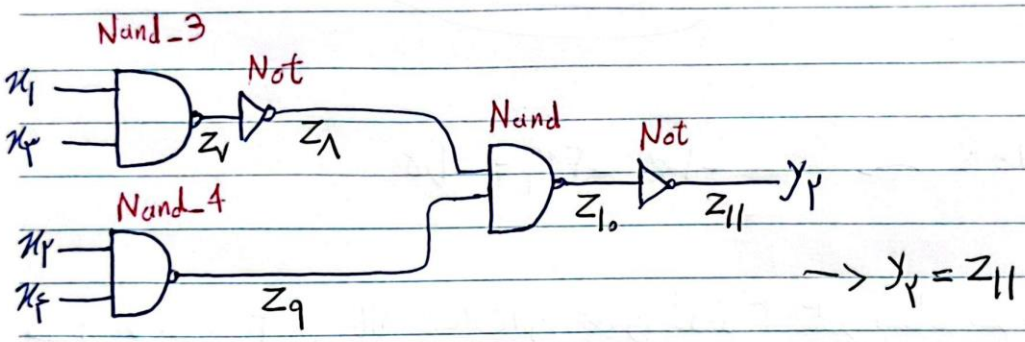
نکته: ضرایب ۳ در بالا به علت این تعیین شدند تا مقادیر آستانه سه گیت برابر شوند.

حال به ترتیب بیت های خروجی را با گیت های بالا می سازیم:

$$Y_۱ = x_۱ x_۲ x_۳ x_۴ = \overline{(x_۱ x_۲)} \cdot \overline{(x_۳ x_۴)}$$

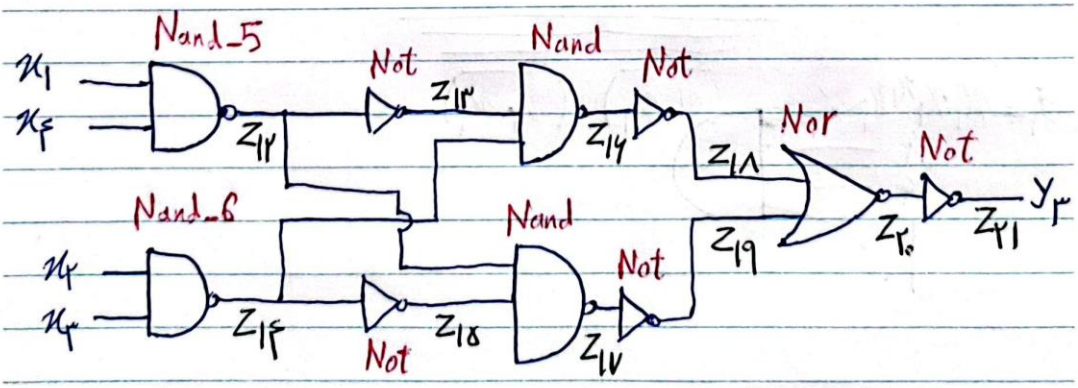


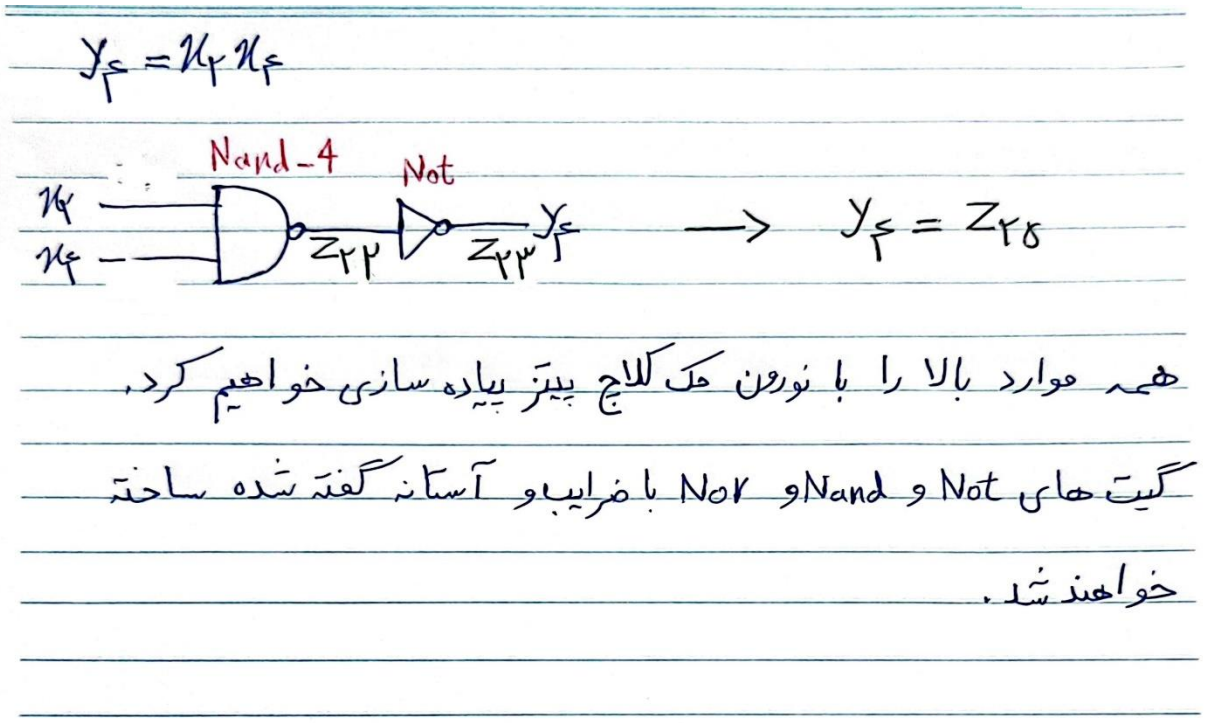
$$y_1 = x_1 x_2 \bar{x}_3 + x_1 x_2 \bar{x}_4 = x_1 x_2 (\bar{x}_3 + \bar{x}_4) = \overline{(x_1 x_2)}\,\overline{(x_2 x_4)}$$



$$y_2 = x_1 x_4 \bar{x}_3 + x_1 x_4 \bar{x}_2 + x_2 x_2 \bar{x}_1 + x_3 x_2 \bar{x}_4$$

$$= \overline{(x_1 x_4)}\,\overline{(x_3 x_2)} + \overline{(x_2 x_2)}\,\overline{(x_1 x_4)}$$





همه موارد بالا را با نورون مک کلاچ پیتز پیاده سازی خواهیم کرد.
گیت های Not و Nand و Nor با ضرایب و آستانه گفته شده ساخته
خواهند شد.

❖ حال تمامی موارد بالا را در پایتون پیاده سازی میکنیم؛ ابتدا کتابخانه‌های موردنیاز خود را وارد کرده و کلاس نورون مک کلاچ پیتز را تعریف میکنیم:

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

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

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

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

❖ حالا تمامی نرون های توضیح داده شده را به کمک نورون مک کلاچ پیتز نوشته و سپس خروجی های میانی را حساب کرده و در نهایت چهار بیت خروجی را با تابعی ضربی که نوشتیم محاسبه خواهد شد:

```
def Zarb(state):
  # تعریف گیت ها
  Not = McCulloch_Pitts_neuron([-1] , -0.5)
  Nor = McCulloch_Pitts_neuron([-1, -1] , -0.5)
  Nand = McCulloch_Pitts_neuron([-1/3, -1/3] , -0.5)
  Nand_1 = McCulloch_Pitts_neuron([-1, -1 , 0, 0] , -1.5)
  Nand_2 = McCulloch_Pitts_neuron([0, 0 , -1, -1] , -1.5)
  Nand_3 = McCulloch_Pitts_neuron([-1, 0 , -1, 0] , -1.5)
  Nand_4 = McCulloch_Pitts_neuron([0, -1 , 0, -1] , -1.5)
  Nand_5 = McCulloch_Pitts_neuron([-1, 0 , 0, -1] , -1.5)
  Nand_6 = McCulloch_Pitts_neuron([0, -1 , -1, 0] , -1.5)

  # y1
  # تعریف خروجی های میانی
  z1 = Nand_1.model(np.array([state[0], state[1], state[2], state[3]]))
  z2 = Not.model(np.array([z1]))
  z3 = Nand_2.model(np.array([state[0], state[1], state[2], state[3]]))
  z4 = Not.model(np.array([z3]))
  z5 = Nand.model(np.array([z2, z4]))
  z6 = Not.model(np.array([z5]))

  y1 = z6


  # y2
  # تعریف خروجی های میانی
  z7 = Nand_3.model(np.array([state[0], state[1], state[2], state[3]]))
  z8 = Not.model(np.array([z7]))
  z9 = Nand_4.model(np.array([state[0], state[1], state[2], state[3]]))
  z10 = Nand.model(np.array([z8, z9]))
  z11 = Not.model(np.array([z10]))

  y2 = z11


  # y3
  # تعریف خروجی های میانی
  z12 = Nand_5.model(np.array([state[0], state[1], state[2], state[3]]))
  z13 = Not.model(np.array([z12]))
  z14 = Nand_6.model(np.array([state[0], state[1], state[2], state[3]]))
  z15 = Not.model(np.array([z14]))
  z16 = Nand.model(np.array([z13, z14]))
```

```
    z17 = Nand.model(np.array([z12, z15]))
    z18 = Not.model(np.array([z16]))
    z19 = Not.model(np.array([z17]))
    z20 = Nor.model(np.array([z18, z19]))
    z21 = Not.model(np.array([z20]))

    y3 = z21


    # y4
    # تعریف خروجی های میانی
    z22 = Nand_4.model(np.array([state[0], state[1], state[2], state[3]]))
    z23 = Not.model(np.array([z22]))

    y4 = z23

    return list([y1,y2,y3,y4])
```

❖ حالا با دستورات زیر ورودی خود را ساخته و به تابع ضرب خود میدهیم:

```
state_b = [1, 0]
state = list(itertools.product(state_b, state_b, state_b, state_b))
print('state: ', state)
```

```
state:  [(1, 1, 1, 1), (1, 1, 1, 0), (1, 1, 0, 1), (1, 1, 0, 0), (1, 0, 1, 1), (1, 0, 1, 0), (1, 0, 0, 1), (1, 0, 0, 0), (0, 1, 1, 1), (0, 1,
```

```
X = list(itertools.product(state))
for i in X:
    kh = Zarb(i[0])
    print("Zarbe adad aval do biti ", str(i[0][0])+str(" ")+str(i[0][1]), " dar adad dovom do biti ", str(i[0][2])+str(" ")+str(i[0][3]),
          " barabar ast ba adad chahar biti ",str(kh[0]) + str(" ")+str(kh[1])+ str(" ")+str(kh[2])+ str(" ")+str(kh[3]), " ast ")
```

❖ و در نهایت خروجی کد بالا، لیستی بصورت زیر چاپ میشود که نتایج ضرب دو عدد دوبیتی را در حالت های مختلف به ما میدهد:

```
Zarbe adad aval do biti  1 1  dar adad dovom do biti  1 1  barabar ast ba adad chahar biti  1 0 0 1  ast
Zarbe adad aval do biti  1 1  dar adad dovom do biti  1 0  barabar ast ba adad chahar biti  0 1 1 0  ast
Zarbe adad aval do biti  1 1  dar adad dovom do biti  0 1  barabar ast ba adad chahar biti  0 0 1 1  ast
Zarbe adad aval do biti  1 1  dar adad dovom do biti  0 0  barabar ast ba adad chahar biti  0 0 0 0  ast
Zarbe adad aval do biti  1 0  dar adad dovom do biti  1 1  barabar ast ba adad chahar biti  0 1 1 0  ast
Zarbe adad aval do biti  1 0  dar adad dovom do biti  1 0  barabar ast ba adad chahar biti  0 1 0 0  ast
Zarbe adad aval do biti  1 0  dar adad dovom do biti  0 1  barabar ast ba adad chahar biti  0 0 1 0  ast
Zarbe adad aval do biti  1 0  dar adad dovom do biti  0 0  barabar ast ba adad chahar biti  0 0 0 0  ast
Zarbe adad aval do biti  0 1  dar adad dovom do biti  1 1  barabar ast ba adad chahar biti  0 0 1 1  ast
Zarbe adad aval do biti  0 1  dar adad dovom do biti  1 0  barabar ast ba adad chahar biti  0 0 1 0  ast
Zarbe adad aval do biti  0 1  dar adad dovom do biti  0 1  barabar ast ba adad chahar biti  0 0 0 1  ast
Zarbe adad aval do biti  0 1  dar adad dovom do biti  0 0  barabar ast ba adad chahar biti  0 0 0 0  ast
Zarbe adad aval do biti  0 0  dar adad dovom do biti  1 1  barabar ast ba adad chahar biti  0 0 0 0  ast
Zarbe adad aval do biti  0 0  dar adad dovom do biti  1 0  barabar ast ba adad chahar biti  0 0 0 0  ast
Zarbe adad aval do biti  0 0  dar adad dovom do biti  0 1  barabar ast ba adad chahar biti  0 0 0 0  ast
Zarbe adad aval do biti  0 0  dar adad dovom do biti  0 0  barabar ast ba adad chahar biti  0 0 0 0  ast
```

سوال سوم

بخش اول

❖ در ابتدا با دستور gdown داده های خود را که همان تصاویر حروف الفبا هستند بارگذاری میکنیم:

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

❖ حال برای اینکه بتوانیم روی داده های تصویری کار کنیم باید در ابتدا تابعی بنویسیم تا این تصاویر را دریافت کرده و بصورت عددی یعنی ماتریس دربیاورد؛ همچنین باید کتابخانه‌های موردنیاز برای کار با داده های تصویری آورده شود؛ کد تابع موردنظر بصورت زیر است:

```
from PIL import Image, ImageDraw
import random

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 -1 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  # White pixel
                binary_representation.append(-1)
            else:
                red, green, blue = 0, 0, 0  # Black pixel
                binary_representation.append(1)

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

```
    # Clean up the drawing tool.
    del draw

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

- کدهای بالا در واقع عکس را بصورت پیکسلی درآورده و عرض و ارتفاع آن را حساب کرده و حلقه هایی تعریف میکند که بتواند رنگ‌های RGB را تشخیص داده و ترکیب کرده و با تعریف یک شرط نقاط سیاه و سفید مشخص میشود و در نهایت یک ماتریس از تصویر ساخته میشود.
- حال از آنجایی که میخواهیم شبکه عصبی برای شناسایی حروف طراحی کنیم نیازمند این هستیم تا تصاویر حروف را بطور ناقص داشته باشیم تا مدل خود را آزمایش کنیم، لذا باید تابعی تعریف کنیم تا از تصاویر حروف کامل، تصاویر دارای نویز بسازد، به صورت زیر:

```
from PIL import Image, ImageDraw
import random

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

    for i, image_path in enumerate(image_paths, start=1):
        noisy_image_path = f"/content/noisy{i}.jpg"
        getNoisyBinaryImage(image_path, noisy_image_path)
        print(f"Noisy image for {image_path} generated and saved as
{noisy_image_path}")

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

    Args:
        input_path (str): The file path to the input image.
        output_path (str): The file path to save the noisy image.
    """
    # Open the input image.
```

```
    image = Image.open(input_path)

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

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

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

    # Define a factor for introducing noise.
    noise_factor = 1500
    # Loop through all pixels in the image.
    for i in range(width):
        for j in range(height):
            # Generate a random noise value within the specified factor.
            rand = random.randint(-noise_factor, noise_factor)

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

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

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

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

```
    # Clean up the drawing tool.
    del draw

# Generate noisy images and save them
generateNoisyImages()
```

- تابعی که در بالا نوشته شده در واقع مقادیر نویز را به مقادیر داده‌های عددی که تابع اولی برای ما تولید کرد اضافه میکند و سپس بصورت عکسی بعنوان یک تصویر دارای نویز ذخیره میکند تا در ادامه اگر خواستیم شبکه عصبی خود را تست کنیم، با این تصاویر این کار را انجام دهیم.

بخش دوم

- در این بخش یک شبکه عصبی همینگ طراحی میکنیم که براساس تصاویر کامل حروف الفبایی که به آن دادیم آموزش میبیند و در ادامه اگر بخواهیم آن را تست کنیم، شباهت بردار ورودی تست را با بردارهایی که از تصاویر حروف الفبا آموزش دیده مقایسه کرده و بیشترین شباهت را تشخیص داده و معین میکند که این عکس ورودی ما مربوط به کدام حرف الفباست؛ کد تمامی موارد گفته شده به صورت زیر است:

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

# Define the path to the input image
IMAGE_PATH = "/content/missingpoint_3.jpg"

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

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

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

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

    Returns:
        list of lists: The transformed matrix.
    """
    matrix = [[0 for j in range(a)] for i in range(b)]
    k = 0
    j = 0
    while k < b:
        i = 0
        while i < a:
            matrix[k][i] = vector[j]
            j += 1
            i += 1
        k += 1
    return matrix

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

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

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

def action(vector, T, Emax):
```

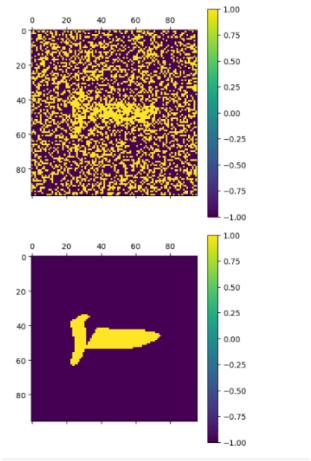
```
    """
    Activation function to process a vector.

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

    Returns:
        list: The output vector after activation.
    """
    result_vector = []
    for value in vector:
        if value <= 0:
            result_vector.append(0)
        elif 0 < value <= T:
            result_vector.append(Emax * value)
        elif value > T:
            result_vector.append(T)
    return result_vector

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

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

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

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

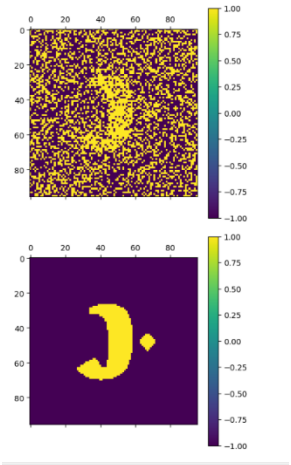
```
    Calculate the difference between two vectors and compute the norm of
the resulting vector.

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

    Returns:
        float: The Euclidean norm of the difference between the two
vectors.
    """
    difference = []
    for i in range(len(vector)):
        difference.append(vector[i] - p[i])
    sum = 0
    for element in difference:
        sum += element * element
    return sqrt(sum)

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

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

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

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

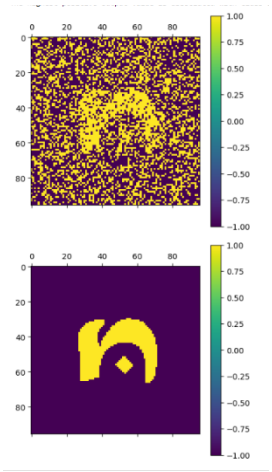
```
m = len(x[0])
w = [[(x[i][j]) / 2 for j in range(m)] for i in range(k)]  # Weight matrix
T = m / 2  # Activation function threshold parameter
e = round(1 / len(x), 1)
E = [[0 for j in range(k)] for i in range(k)]  # Synaptic connection
matrix
Emax = 0.0001  # Maximum allowable difference norm between output vectors
in consecutive iterations
U = 1 / Emax

# Set values for the synaptic connection matrix
for i in range(k):
    for j in range(k):
        if j == i:
            E[i][j] = 1.0
        else:
            E[i][j] = -e

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

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

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

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

if max(y[len(y) - 1]) == 0:
    print("The Hamming network cannot make a preference between classes.")
```

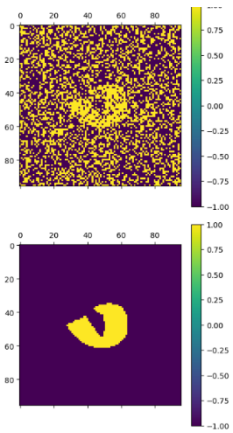
```
    print("In the case of a small number of input characteristics, the
network may not be able to classify the image.")
    plt.show()
    exit()
else:
    q = change(x[result_index - 1], a, b)
    print('The highest positive output value is associated with class',
result_index)
    plt.matshow(q)
    plt.colorbar()
    plt.show()
```

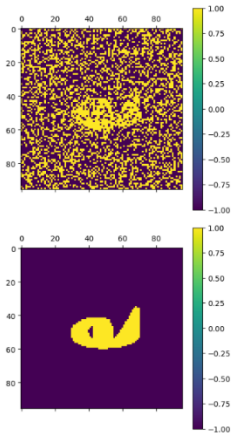
❖ حال که شبکه عصبی را طراحی کردیم تصاویر حروف مختلف را در ابتدا با noise_factor برابر با 300 به شبکه خود میدهیم و حروف الفبا را بصورت زیر تشخیص میدهد:



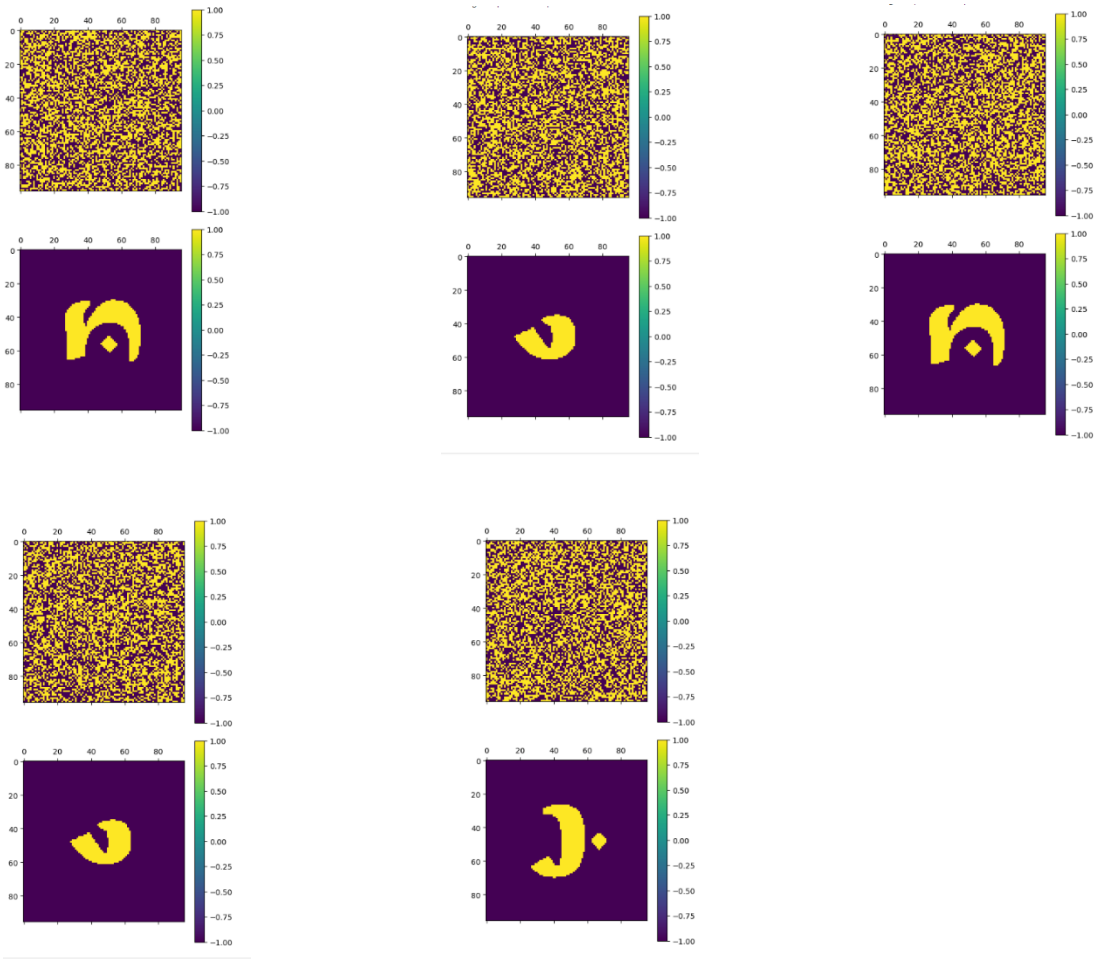








❖ حال مقدار noise_factor بسیار زیاد میکنیم و عدد 10000 قرار میدهیم و تصاویر را همانطور که در بالا چیده شده اند به ترتیب در پایین هم همانطور میچینیم:



❖ همانطور که از دو دسته تصاویر بالا قابل مشاهده است با افزایش بیش از حد نویز مدل ما در تشخیص برخی از حروف دچار مشکل میشود مثلا در حالتی که نویز بالاست تنها حروف **ج** و **د** را به درستی تشخیص داد و بقیه را غلط تشخیص داد؛ لذا باید توجه کرد که میزان نویز نباید از یک حدی بیشتر شود و یا برای حل این مشکل مدل ما باید با تعداد تصاویر بیشتر که شامل حالت های مختلف نویزی نیز میباشد آموزش ببیند تا تصاویر دارای نویز بالا را نیز بتواند تشخیص دهد.

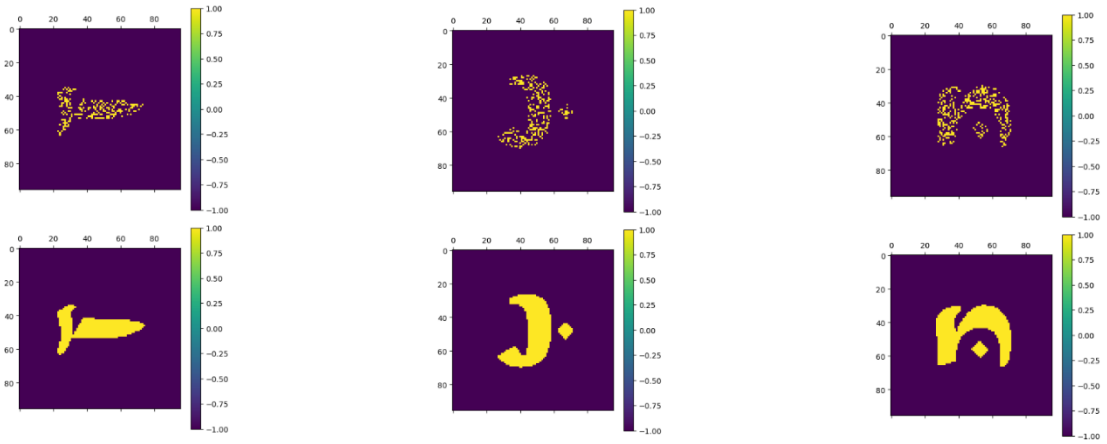
بخش سوم

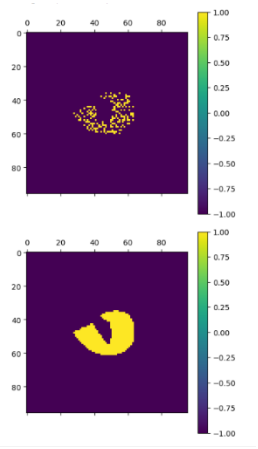
❖ حال در این بخش با الهام از تابع تولید تصاویر نویزی، تابعی مینویسیم که تصاویر درای میزان مشخصی Missing Point تولید کند تا بتوانیم مدل خود را براساس این تصاویر تست کنیم؛ کد تابع مورد نظر بصورت زیر است:

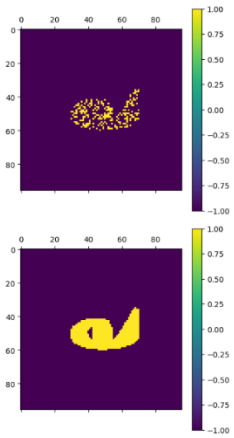
```
from PIL import Image, ImageDraw
import random

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

    for i, image_path in enumerate(image_paths, start=1):
        missing_image_path = f"/content/missingpoint{i}.jpg"
        addMissingPointsToNoisyImage(image_path, missing_image_path)
        print(f"missing_image for {image_path} generated and saved as
{missing_image_path}")

def addMissingPointsToNoisyImage(input_path, output_path,
missing_ratio=0.1):
    # Open the noisy image.
    image = Image.open(input_path)

    # Determine the image's width and height in pixels.
    width, height = image.size

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

    # Calculate the number of total points in the image.
    total_points = sum(1 for i in range(width) for j in range(height))

    # Calculate the desired number of missing points based on the
specified ratio.
    num_missing_points = int(total_points * (1 - missing_ratio))

    # Create a list of all pixels in the image.
```

```
    all_pixels = [(i, j) for i in range(width) for j in range(height)]

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

    # Shuffle the list of pixels to add randomness.
    random.shuffle(all_pixels)

    # Loop until the desired number of missing points is reached.
    for i in range(num_missing_points):
        # Get the next pixel in the shuffled list.
        x, y = all_pixels[i]

        # Set the pixel color to create a missing point.
        draw.point((x, y), (255, 255, 255))

    # Save the final image with missing points as a file.
    image.save(output_path, "JPEG")

    # Clean up the drawing tool.
    del draw

# Generate noisy images and add missing points to them
generateNoisyImages()
```

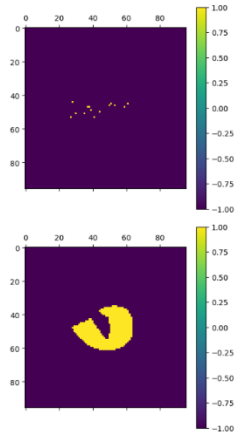
❖ حال تصاویر حروف مختلف را در ابتدا با missing_ratio برابر با 0.33 به شبکه خود میدهیم و حروف الفبا را بصورت زیر تشخیص میدهد:

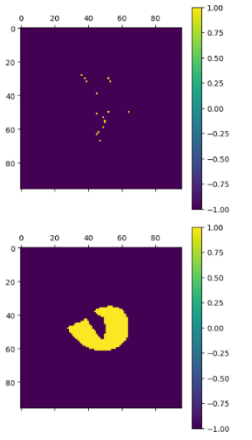


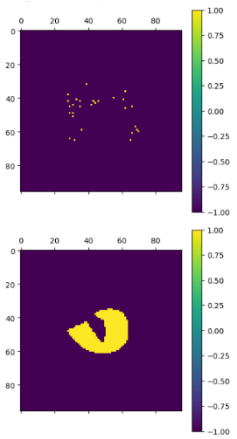


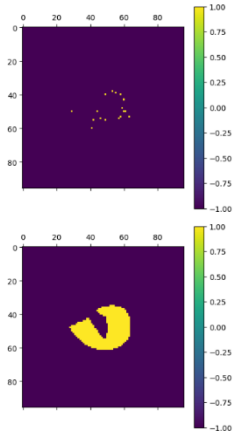


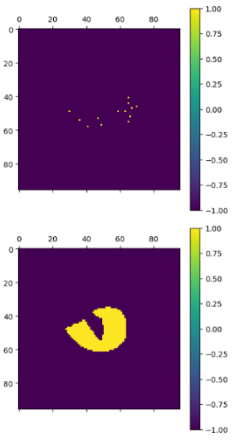
❖ حال مقدار missing_ratio بسیار کم میکنیم و عدد 0.03 قرار میدهیم و تصاویر را همانطور که در بالا چیده شده اند به ترتیب در پایین هم همانطور میچینیم:











- همانطور که در دو دسته شکل های بالا میبینیم وقتی تعداد نقاط گم شده زیاد میشود مدل ما نمیتواند حروف را بخوبی تشخیص بدهد مثلا در دسته شکل دوم همه را حرف **د** تشخیص داده و برای حل این مشکل میتوان اقدامات زیر را انجام داد:
- اگر میزان نقاط گم‌شده به حدی بزرگ شود که شبکه همینگ دچار اختلال شود، می‌توان از روش‌های متعدد برای مدیریت نقاط گم‌شده و افزایش کیفیت ورودی به شبکه استفاده کرد. در زیر چند راهکار آورده شده‌اند:

1. استفاده از تکنیک‌های مدیریت نقاط گم‌شده: از روش‌های هوشمندانه‌تری باید برای مدیریت نقاط گم‌شده استفاده کرد. مثلاً می‌توان از الگوریتم‌های پردازش تصویر و یادگیری ماشین برای تشخیص نقاط گم‌شده و تلاش برای بازسازی آنها استفاده کرد.

2. افزایش ابعاد شبکه: اگر میزان نقاط گم‌شده زیاد است، ممکن است نیاز به افزایش ابعاد شبکه (مثلاً افزایش تعداد نورون‌ها یا لایه‌ها) باشد تا شبکه بتواند با دقت بیشتری با ورودی‌های ناقص کار کند.

3. استفاده از تقنیات نویززایی: می‌توان از تقنیات نویززایی پیشرفته استفاده کرد تا تصاویر ورودی حاوی نقاط گم‌شده، بهبود یابند و در نتیجه، تاثیر منفی نقاط گم‌شده بر عملکرد شبکه کاهش یابد.

4. استفاده از شبکه‌های مقاوم به نقاط گم‌شده: برخی از شبکه‌ها به نام "شبکه‌های مقاوم به نقاط گم‌شده" طراحی شده‌اند تا بتوانند با نقاط گم‌شده بهتر مقابله کنند. این شبکه‌ها معمولاً از مکانیسم‌هایی مانند attention و recurrence استفاده می‌کنند.

5. آموزش با داده‌های نقاط گم‌شده: اگر امکان دارد، می‌توان شبکه را با داده‌های آموزشی حاوی نقاط گم‌شده نیز آموزش داد تا بتواند در مواجهه با این شرایط بهتر عمل کند.

به طور کلی، ترکیبی از این راهکارها ممکن است به بهبود عملکرد شبکه منجر شود. بهتر است هر راهکار را با توجه به شرایط و مشخصات دقیق مسئله خود امتحان نمود.

سوال چهارم

بخش اول تا چهارم

❖ ابتدا کتابخانه‌های مورد نیاز خود را قرار میدهیم:

```
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")
```

❖ حال دیتاست خود را با دستور gdown بارگذاری میکنیم:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1y4gApyw8HURa-j-SNiTWVkzUw8czxOZN
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (5.0.1)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.13.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.11.17)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
Downloading...
```

❖ حال دیتاست خود را به فرمت دیتافریم درآورده و پنج سطر اول آن را مشاهده میکنیم:

```
# Load the dataset from the specified file path
df = pd.read_csv('/content/data.csv')

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

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	street	city	statezip	country
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	18810 Densmore Ave N	Shoreline	WA 98133	USA
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	709 W Blaine St	Seattle	WA 98119	USA
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	26206-26214 143rd Ave SE	Kent	WA 98042	USA
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	857 170th Pl NE	Bellevue	WA 98008	USA
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	9105 170th Ave NE	Redmond	WA 98052	USA

❖ حال با زدن کد **info.** اطلاعاتی در مورد نام ستونها و تعداد داده‌های پوچ و همچنین نوع داده ها در اختیار ما قرار میگیرد:

```
# Display information about the DataFrame
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
```

❖ حال دستوری برای نمایش تعداد داده‌های null هر ستون مینویسیم و نمایش میدهیم:

```
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
```

- همانطور که میبینیم در هیچ ستونی داده null نداریم، بنابراین نیازی به استفاده از دستوری برای حذف داده null نداریم.
- حال ستون date را با بارگذاری کتابخانه مخصوص آن به دو ستون ماه و سال تبدیل میکنیم با دستور زیر:

```
# تبدیل ستون 'date' به نوع داده datetime
df['date'] = pd.to_datetime(df['date'])

# اضافه کردن ستون 'ماه'
df['month'] = df['date'].dt.month

# اضافه کردن ستون 'سال'
df['year'] = df['date'].dt.year

# نمایش DataFrame جدید
print(df)
```

- و خروجی کد بالا دو ستون ماه و سال را به دیتاست اضافه میکند.

```
2    2014-05-02  3.420000e+05       3.0       2.00        1930     11947
3    2014-05-02  4.200000e+05       3.0       2.25        2000      8030
4    2014-05-02  5.500000e+05       4.0       2.50        1940     10500
...         ...           ...       ...        ...         ...       ...
4595 2014-07-09  3.081667e+05       3.0       1.75        1510      6360
4596 2014-07-09  5.343333e+05       3.0       2.50        1460      7573
4597 2014-07-09  4.169042e+05       3.0       2.50        3010      7014
4598 2014-07-10  2.034000e+05       4.0       2.00        2090      6630
4599 2014-07-10  2.206000e+05       3.0       2.50        1490      8102

      floors  waterfront  view  condition  sqft_above  sqft_basement  \
0        1.5           0     0          3        1340              0
1        2.0           0     4          5        3370            280
2        1.0           0     0          4        1930              0
3        1.0           0     0          4        1000           1000
4        1.0           0     0          4        1140            800
...      ...         ...   ...        ...         ...            ...
4595     1.0           0     0          4        1510              0
4596     2.0           0     0          3        1460              0
4597     2.0           0     0          3        3010              0
4598     1.0           0     0          3        1070           1020
4599     2.0           0     0          4        1490              0

      yr_built  yr_renovated                    street       city  statezip  \
0         1955          2005       18810 Densmore Ave N  Shoreline  WA 98133
1         1921             0           709 W Blaine St    Seattle  WA 98119
2         1966             0  26206-26214 143rd Ave SE       Kent  WA 98042
3         1963             0            857 170th Pl NE   Bellevue  WA 98008
4         1976          1992         9105 170th Ave NE    Redmond  WA 98052
...        ...           ...                       ...        ...       ...
4595      1954          1979            501 N 143rd St    Seattle  WA 98133
4596      1983          2009          14855 SE 10th Pl   Bellevue  WA 98007
4597      2009             0           759 Ilwaco Pl NE     Renton  WA 98059
4598      1974             0         5148 S Creston St    Seattle  WA 98178
4599      1990             0         18717 SE 258th St  Covington  WA 98042

     country  month  year
0        USA      5  2014
1        USA      5  2014
2        USA      5  2014
3        USA      5  2014
4        USA      5  2014
...      ...    ...   ...
4595     USA      7  2014
4596     USA      7  2014
4597     USA      7  2014
4598     USA      7  2014
4599     USA      7  2014

[4600 rows x 20 columns]
```

- حال ستون date را طبق خواسته سوال حذف میکنیم، ستون مربوط به سال و کشور را نیز به علت اینکه دارای فقط یک unique value هستند حذف میکنیم و همچنین ستونهایی که اطلاعات خاصی ندارند؛ سپس ستون هایی که بصورت عددی نیستند، بصورت عددی درمی‌آوریم:

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

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

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

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

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	...
0	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	...
1	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	...
2	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	...
3	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	...
4	550000.0	4.0	2.50	1940	10500	1.0	0	0	4	1140	...

5 rows × 57 columns

❖ حال کد ماتریس همبستگی را میزنیم و خروجی اش را مشاهده میکنیم:

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

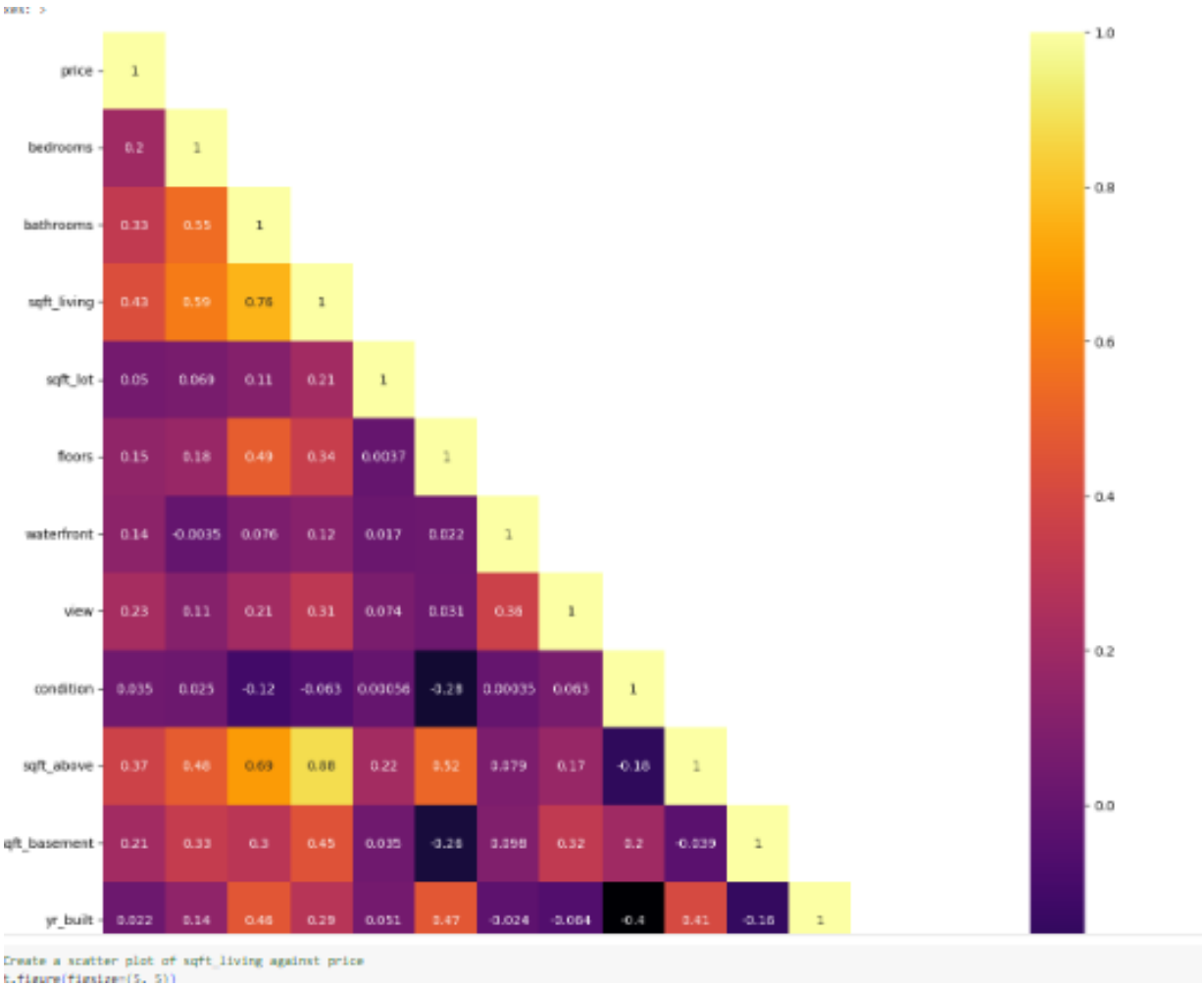
```
price                          1.000000
sqft_living                    0.430410
sqft_above                     0.367570
bathrooms                      0.327110
view                           0.228504
sqft_basement                  0.210427
bedrooms                       0.200336
floors                         0.151461
city_Mercer Island             0.140007
waterfront                     0.135648
city_Bellevue                  0.134828
city_Medina                    0.129795
city_Clyde Hill                0.066867
sqft_lot                       0.050451
city_Redmond                   0.047612
city_Sammamish                 0.047604
month                          0.041081
city_Kirkland                  0.036375
city_Seattle                   0.035642
condition                      0.034915
city_Yarrow Point              0.033640
yr_built                       0.021857
city_Newcastle                 0.017201
city_Woodinville               0.016361
city_Issaquah                  0.016139
city_Fall City                 0.012220
city_Beaux Arts Village        0.005049
city_Preston                   0.000388
city_Snoqualmie Pass          -0.000705
city_Ravensdale               -0.002624
city_Inglewood-Finn Hill      -0.003321
city_Snoqualmie               -0.003477
city_Normandy Park            -0.005022
city_Carnation                -0.005313
city_Milton                   -0.009876
city_Bothell                  -0.010633
city_Vashon                   -0.011219
city_Skykomish                -0.014453
city_Lake Forest Park         -0.016303
city_Black Diamond            -0.016677
city_Pacific                  -0.020944
city_Kenmore                  -0.022357
```

❖ حال ماتریس همبستگی را رسم میکنیم:

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

```
Index(['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
       'waterfront', 'view', 'condition', 'sqft_above', 'sqft_basement',
       'yr_built', 'yr_renovated', 'month'],
      dtype='object')
```

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

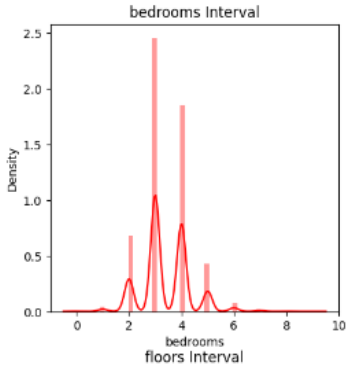


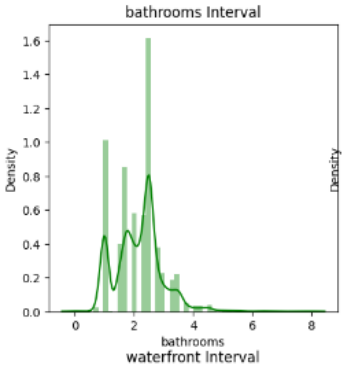
❖ حال نمودارهای توزیع قیمت برای هر ویژگی را با کدهای زیر رسم میکنیم:

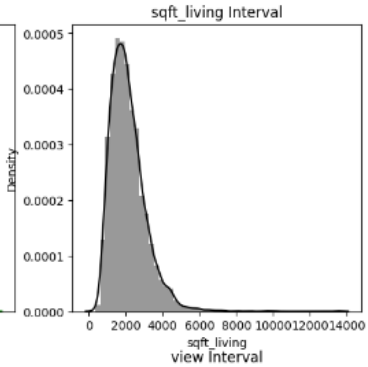
```
# Create a 4x4 grid of subplots for various numerical variables
plt.figure(figsize=(20, 20))

plt.subplot(4,4,1)
sns.distplot(df['bedrooms'], color="red").set_title('bedrooms Interval')

plt.subplot(4,4,2)
sns.distplot(df['bathrooms'], color="green").set_title('bathrooms
Interval')

plt.subplot(4,4,3)
sns.distplot(df['sqft_living'], color="black").set_title('sqft_living
Interval')

plt.subplot(4,4,4)
sns.distplot(df['sqft_lot'], color="blue").set_title('sqft_lot Interval')
```

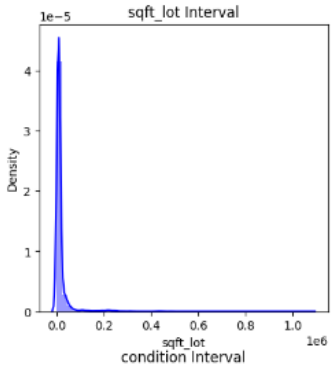
```
plt.subplot(4,4,5)
sns.distplot(df['floors'], color="red").set_title('floors Interval')

plt.subplot(4,4,6)
sns.distplot(df['waterfront'], color="green").set_title('waterfront
Interval')

plt.subplot(4,4,7)
sns.distplot(df['view'], color="black").set_title('view Interval')

plt.subplot(4,4,8)
sns.distplot(df['condition'], color="blue").set_title('condition
Interval')

plt.subplot(4,4,9)
sns.distplot(df['sqft_above'], color="red").set_title('sqft_above
Interval')

plt.subplot(4,4,10)
sns.distplot(df['sqft_basement'], color="green").set_title('sqft_basement
Interval')

plt.subplot(4,4,11)
sns.distplot(df['yr_built'], color="black").set_title('yr_built Interval')

plt.subplot(4,4,12)
sns.distplot(df['yr_renovated'], color="blue").set_title('yr_renovated
Interval')

plt.subplot(4,4,12)
sns.distplot(df['month'], color="blue").set_title('month Interval')

plt.subplot(4,4,13)
sns.distplot(df['price'], color="red").set_title('price Interval')
```

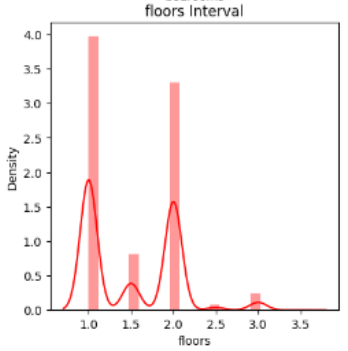
❖ نمودارها بصورت زیر رسم میشوند:

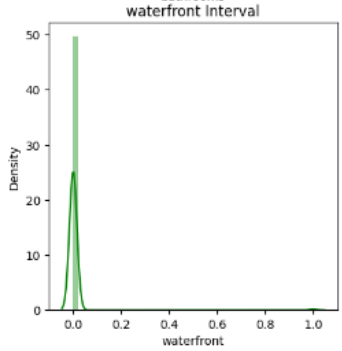

bedrooms Interval
Density
bedrooms

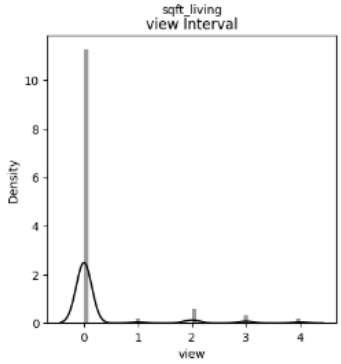

bathrooms Interval
Density
bathrooms

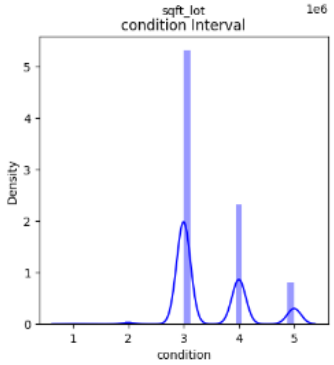

sqft_living Interval
Density
sqft_living

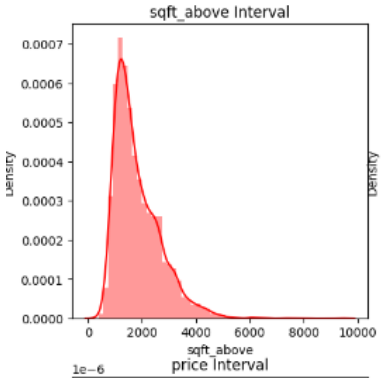

sqft_lot Interval
1e−5
Density
sqft_lot
1e6

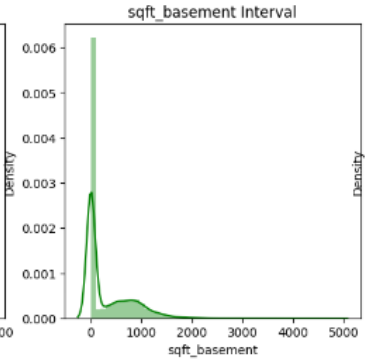

floors Interval
Density
floors

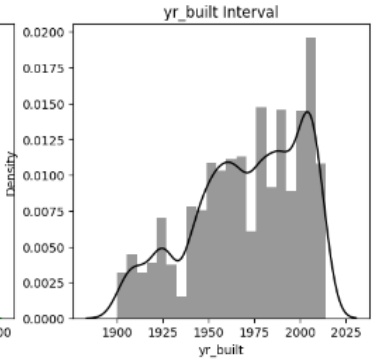

waterfront Interval
Density
waterfront

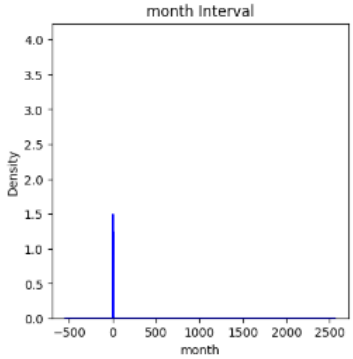

view Interval
Density
view

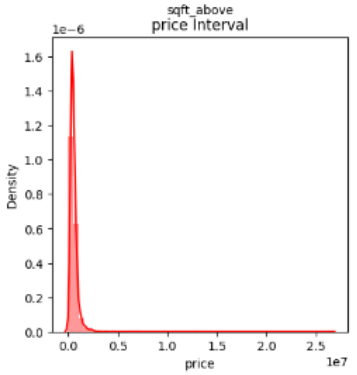

condition Interval
Density
condition


sqft_above Interval
Density
sqft_above


sqft_basement Interval
Density
sqft_basement


yr_built Interval
Density
yr_built


month Interval
Density
month


price Interval
1e−6
Density
price
1e7

❖ ویژگی sqft_living بیشترین همبستگی با قیمت را دارد پس نمودار توزیع قیمت براساس آن را رسم میکنیم:

```
# Create a scatter plot of sqft_living 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()
```

بخش پنجم

❖ در این بخش ویژگی‌ها و قیمت را از هم جدا کرده و داده‌ها را به دو دسته آموزش و ارزیابی با نسبت 80 به 20 تقسیم میکنیم و سپس داده ها را با روش min_max_scaler نرمالسازی کرده و در ادامه با مدل خود آموزش خواهیم داد:

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

```
# Perform train-test split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=7)
```

```
# Print the shapes of the datasets
print("X Train Scaler : ", x_train.shape) # Print shape of x_train
print("X Test Scaler : ",  x_test.shape)  # Print shape of x_test
print("Y Train Scaler : ", y_train.shape) # Print shape of y_train
print("Y Test Scaler : ",  y_test.shape)  # Print shape of y_test
```

```
X Train Scaler :  (3680, 56)
X Test Scaler :  (920, 56)
Y Train Scaler :  (3680,)
Y Test Scaler :  (920,)
```

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

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

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

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

scaler_2 = MinMaxScaler()

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

بخش ششم

- حال یک مدل MLP با کتابخانه keras با دو لایه پنهان میسازیم که لایه اول دارای 50 نورون و لایه دوم دارای 30 نورون است و همچنین تعداد پارامترهایی که در فرآیند آموزش باید تعیین شوند مشخص میشود بصورت زیر:

```
model_2 = Sequential()

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

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

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

model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 50)                2850

 dense_4 (Dense)             (None, 30)                1530

 dense_5 (Dense)             (None, 1)                 31

=================================================================
Total params: 4411 (17.23 KB)
Trainable params: 4411 (17.23 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

❖ حال با تابع اتلاف bce و بهینه‌ساز adam مدل خود را تنظیم کرده و با تعیین 15 درصد داده بعنوان داده‌های اعتبارسنجی و با تعیین تعداد ایپاک 300 ، داده های خود را آموزش داده و معیار R2score را برای مدل خود محاسبه میکنیم:

```
model_2.compile(optimizer='adam', loss='mse')
history = model_2.fit(x_train, y_train, validation_split=0.15, epochs=300 ,batch_size=100, verbose=0)
```

```
#Evaluate the model
loss = model_2.evaluate(x_test , y_test)
```

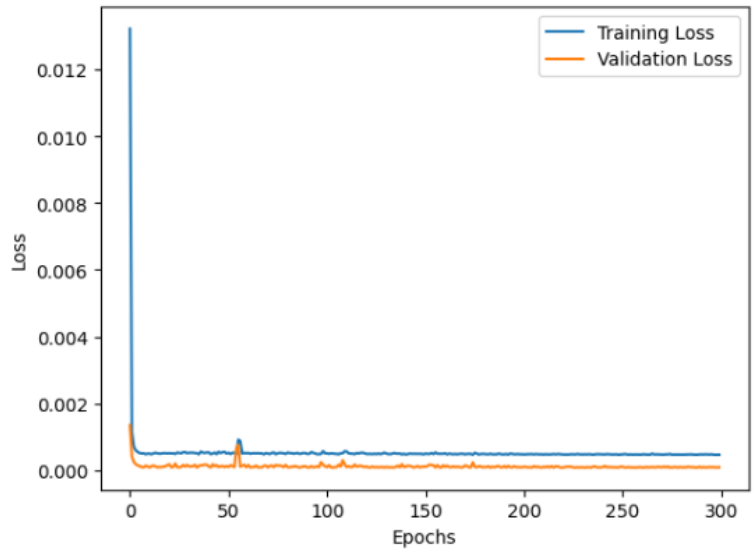
```
29/29 [==============================] - 0s 2ms/step - loss: 8.3767e-05
```

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

rscore_2
```

```
29/29 [==============================] - 0s 2ms/step
0.5583630194225643
```

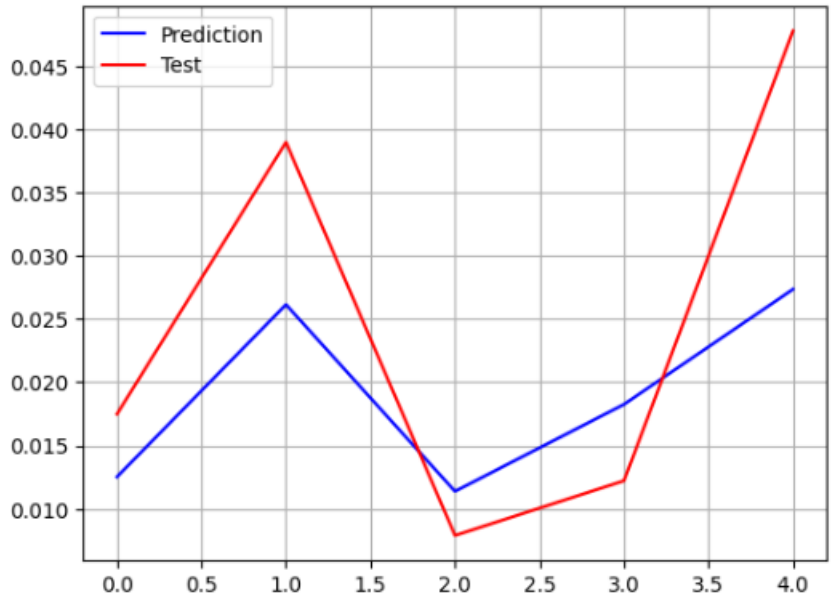
❖ میبینیم که دقت مدل ما متوسط است، حال نمودارهای اتلاف داده‌های آموزش و اعتبارسنجی را رسم میکنیم:

```
# 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()
```



- در نمودار بالا همانطور که دیده میشود مدل ما بعد از حدود 20 ایپاک به همگرایی میرسد و البته نمودار اتلاف دارای نویز میباشد و همین دلیل بر این است که دقیق نمیتواند پیشبینی کند بلکه با یک محدوده خطا و بصورت کلی پیشبینی میکند.
- حال قیمت خانه را میخواهیم برای پنج نمونه داده به صورت زیر پیشبینی کند:

```
# 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()
```



- همانطور که میبینیم با توجه به نمودار بالا قیمت خانه بصورت دقیق تعیین نمیشود بلکه بصورت کلی و با یک محدوده خطایی میتواند پیشبینی کند.

بخش هفتم

- حال با تابع اتلاف mae و بهینه‌ساز sgd مدل خود را پیاده‌سازی کرده و با تعیین 15 درصد داده بعنوان داده‌های اعتبارسنجی و با تعیین تعداد ایپاک 300 ، داده های خود را آموزش داده و معیار R2score را برای مدل خود محاسبه میکنیم:

```
model_2.compile(optimizer='sgd', loss='mae')
history = model_2.fit(x_train, y_train, validation_split=0.15, epochs=300 ,batch_size=100, verbose=0)
```

```
#Evaluate the model
loss = model_2.evaluate(x_test , y_test)
```

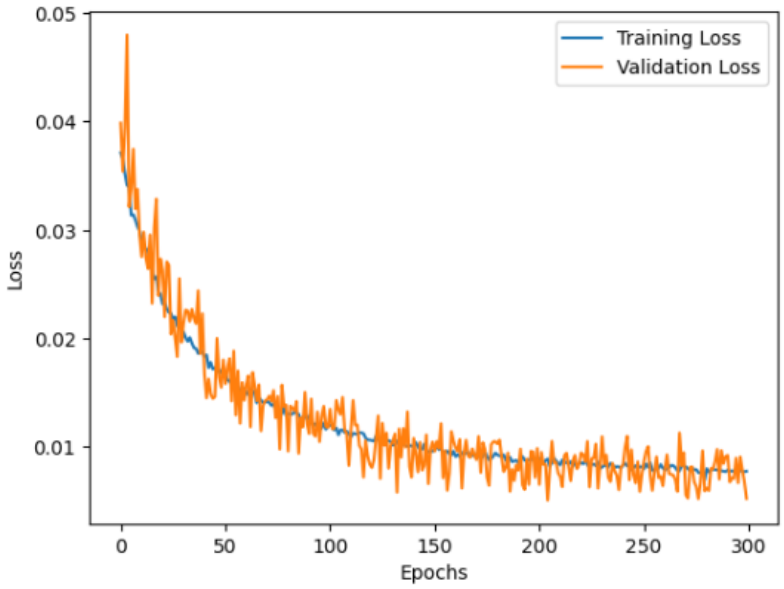
```
29/29 [==============================] - 0s 1ms/step - loss: 0.0052
```

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

rscore_2
```

```
29/29 [==============================] - 0s 1ms/step
0.5888380408132119
```

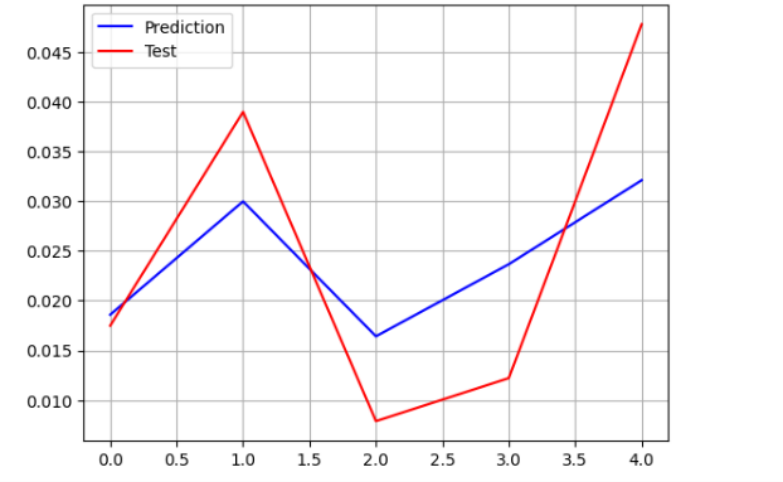
- میبینیم که دقت مدل ما مانند قبل متوسط است و چندان اختلافی ندارد، البته میزان نویز آنها را در تابع اتلاف نیز باید مشاهده نمود، قطعا آنکه نویزش کمتر باشد بهتر است، حال نمودارهای اتلاف داده-های آموزش و اعتبارسنجی را رسم میکنیم:

```
# 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()
```



- در نمودار بالا همانطور که دیده میشود مدل ما بعد از حدود 300 ایپاک به همگرایی میرسد و البته نسبت به مدل قبل که تابع اتلاف و بهینه‌ساز متفاوتی داشت دیرتر به همگرایی رسید نمودار اتلاف آن دارای نویز بیشتری نسبت به مدل قبل میباشد و همین دلیل بر این است که دقیق نمیتواند پیشبینی کند بلکه با یک محدوده خطای بیشتری نسبت به مدل قبل و بصورت کلی‌تر پیشبینی میکند.
- حال قیمت خانه را میخواهیم برای پنج نمونه داده به صورت زیر پیشبینی کند:

```
random_pred = list()
random_test = list()

for i in range(5):
  j = random.randint(0, len(y_pred_2))
  random_pred.append(y_pred_2[i])
  random_test.append(y_test[i])

# 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()
```



- همانطور که میبینیم با توجه به نمودار بالا قیمت خانه بصورت دقیق تعیین نمیشود بلکه بصورت کلی و با یک محدوده خطایی میتواند پیشبینی کند و از مدل قبل با تابع اتلاف و بهینه‌ساز متفاوت ضعیفتر است.

بخش هشتم

❖ اختلاف قیمت پیشبینی شده با قیمت واقعی نه زیاد است و نه کم بلکه بصورت تقریبی متوسط است و برای حل این مشکل راه‌حل‌های زیر وجود دارد:

برای بهبود دقت مدل MLP (شبکه عصبی چند لایه) در پیش‌بینی قیمت خانه، می‌توانید مراحل زیر را انجام دهید:

1. تنظیم هایپرپارامترها:

- تعداد لایه‌ها و نرون‌ها: توسعه یا کاهش تعداد لایه‌ها و نرون‌ها را میشود امتحان کرد.
- نرخ یادگیری (Learning Rate): انتخاب نرخ یادگیری مناسب مهم است. ممکن است نیاز به تغییر نرخ یادگیری باشد تا به یک مقدار مناسب رسید.
- تعداد دوره‌ها (Epochs): اگر مدل ما به سرعت به دقت خوبی می‌رسد و سپس دقت بهبود نمی‌یابد، ممکن است نیاز به افزایش تعداد دوره‌ها باشد.
- تعداد دسته‌ها (Batch Size): اندازه دسته‌ها می‌تواند تأثیری بر سرعت و دقت آموزش داشته باشد. امتحان کنید تا مقدار بهینه را پیدا کنید.

2. استفاده از ویژگی‌های مناسب:

- باید مطمئن شد که از ویژگی‌های مهم و تأثیرگذار برای پیش‌بینی قیمت خانه استفاده می‌کنیم.
- اگر تعداد ویژگی‌ها زیاد است، ممکن است نیاز به انجام انتخاب ویژگی (Feature Selection) یا کاهش ابعاد (Dimensionality Reduction) باشد.

سوال پنجم

بخش اول

❖ ابتدا کتابخانه‌های موردنیاز خود را آپلود میکنیم:

```python
from sklearn.neural_network import MLPClassifier, MLPRegressor

from sklearn.datasets import load_iris

from imblearn.under_sampling import RandomUnderSampler

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

import numpy as np
```

❖ حال دیتاست iris را به خروجی و ورودی تقسیم کرده و تعداد نمونه‌های هر کلاس در خروجی را مشاهده میکنیم:

```python
X, y = load_iris(return_X_y=True)
print(X.shape, y.shape)
```

```
(150, 4) (150,)
```

```python
# Count the number of samples in each class
unique, counts = np.unique(y, return_counts=True)
print("Class counts before balancing:", dict(zip(unique, counts)))
```

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

❖ تعداد نمونه‌های کلاس ها برابر است پس نیازی به بالانس کردن داده‌ها نیست.

❖ در ادامه داده‌ها را با نسبت 80 به 20 به دو دسته آموزش و ارزیابی تقسیم میکنیم:

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

بخش دوم

❖ حال با سه روش رگرسیون لجستیک و کلاس‌بندی با mlp و RBF داده های خود را آموزش داده و با چهار شاخص داده‌های ارزیابی خود را چک میکنیم، کد این سه روش بصورت زیر است:

MLPClassifier

```python
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

```
# Assuming you've trained your model already
model = MLPClassifier(hidden_layer_sizes=(90), random_state=12)
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))
```

LogisticRegression

```
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

# Assuming you've trained your model already
model = LogisticRegression(max_iter=1000, random_state=12)
# model = LogisticRegression(max_iter=10000, random_state=12)
model.fit(x_train, y_train)
```

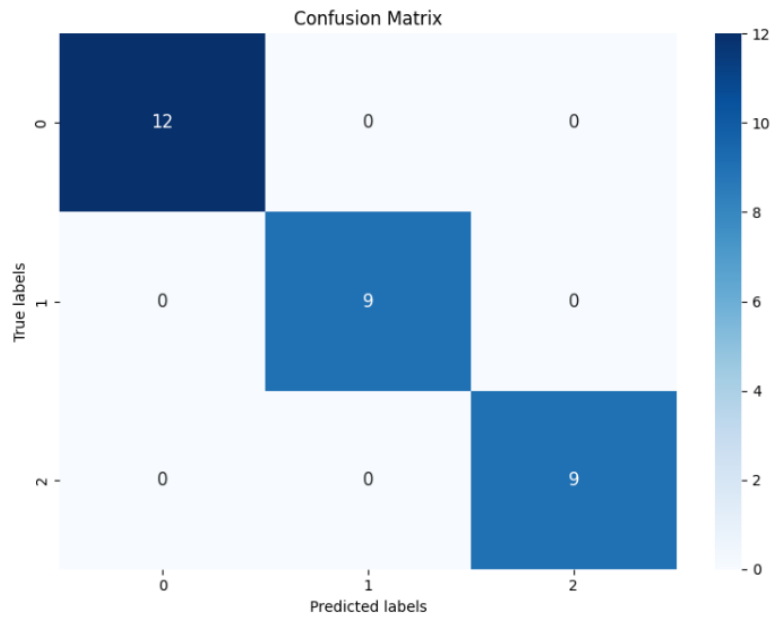
```
# Making predictions on the test set
y_pred = model.predict(x_test)

# 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))
```

RBF

```
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

# Assuming you've trained your model already
model = SVC(kernel='rbf', random_state=12)
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))
```

❖ با اجرای هر سه روش و رسم ماتریس درهمریختگی و محاسبات همه شاخص‌ها، میبینیم که این ماتریس و این شاخص‌ها در هر سه روش دقیقا برابرند . بصورت زیر هستند:



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

           0       1.00      1.00      1.00        12
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00         9

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

- در هر سه روش تمامی شاخص ها عدد 100 درصد را نشان میدهند و این یعنی کار کلاس‌بندی در هر سه روش به نحو احسن انجام شده است از جمله این شاخص ها، شاخص دقت و حساسیت و شاخص f1-score که ترکیبی از دقت و حساسیت را نشان میدهد در همه کلاس های خروجی مقدار آنها 100 درصد است و همچنین شاخص accuracy که روی همه داده‌ها اعمال میشود نیز 100 درصد بدست آمده و این بسیار عالیست.

پایان