

به نام خداوند بخشنده مهربان

درس مبانی سیستم‌های هوشمند

گزارش مینی پروژه دوم

معصومه شریف تبار عزیزی – ۹۹۲۷۶۱۳

۱.

مجموعه داده مد نظر را با روش گفته شده در کلاس (دستور gdown) در محیط گوگل کولب بارگزاری می‌کنیم و آن را در پوشه data قرار می‌دهیم. (در این مجموعه داده x1 و x2 ویژگی‌ها و y کلاس (هدف یا target) است که بصورت باینری ۱ یا ۰- نمایش داده شده‌است).

۱.۱-

حال مجموعه داده را بصورت دیتافریم تبدیل می‌کنیم و استفاده از دستور train test split و تنظیم test_size=0.2 داده‌ها را با نسبت ۸۰ (آموزش) به ۲۰ درصد (آزمون) تقسیم می‌کنیم.

```
import os

#define the folder name
folder_name = "Data"

#check if the folder already exists, and create it if not
if not os.path.exists(folder_name):
    os.makedirs(folder_name)

%cd Data

!pip install --upgrade --no-cache-dir gdown
!gdown 1q$gWefXndoZp6jyZ4TbRqebu9AvlT579

/content/Data
Downloading...
From: https://drive.google.com/uc?id=1q\$gWefXndoZp6jyZ4TbRqebu9AvlT579
To: /content/Data/Perceptron.csv
100% 17.3k/17.3k [00:00<00:00, 45.9MB/s]

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

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

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

# 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=13)

# Initialize Perceptron classifier
clf = Perceptron()
```

۱.۲-

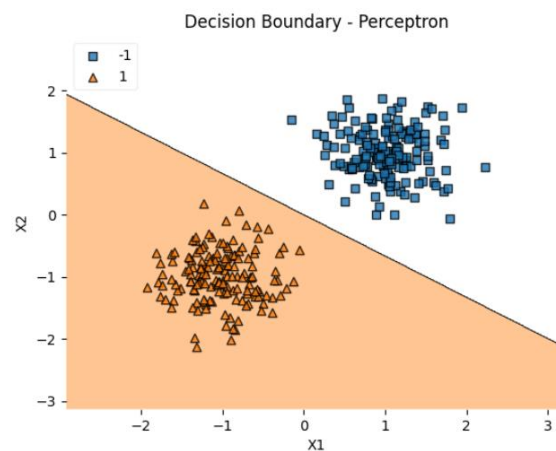
```
# Accuracy on train and test data
train_accuracy = clf.score(X_train, y_train)
test_accuracy = clf.score(X_test, y_test)

print(f"Accuracy on train set: {train_accuracy}")
print(f"Accuracy on test set: {test_accuracy}")

# Convert y_train to integer type
y_train = y_train.astype(np.int)

# Plotting decision boundary
plot_decision_regions(X_train, y_train, clf=clf, legend=2)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Decision Boundary - Perceptron')
plt.show()
```

Accuracy on train set: 1.0
Accuracy on test set: 1.0
<ipython-input-8-fc173a266e03>32: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe to use in all contexts except when interoperating with other code that uses the deprecated `np.int` alias. This will stop working in a future version of NumPy. Deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>
y_train = y_train.astype(np.int)



-۱.۳

تغییر آستانه در مدل‌های آموزش داده شده مانند پرسپترون می‌تواند تأثیر مهمی بر نتایج طبقه‌بندی داشته باشد. آستانه در واقع یک مقدار است که مشخص می‌کند چه مقدار از خروجی مدل به عنوان یک کلاس در نظر گرفته شود. در پرسپترون، اگر خروجی بیشتر از آستانه باشد، مدل یک کلاس را پیش‌بینی می‌کند و در غیر این صورت، کلاس دیگری را پیش‌بینی می‌کند.

تغییر آستانه می‌تواند به شکل زیر تأثیر بگذارد:

۱. تعیین دقت مدل: اگر آستانه را بیشتر کنید، ممکن است دقت مدل در تشخیص یک کلاس افزایش یابد اما در تشخیص دیگر کلاس کاهش یابد و بالعکس. این وابستگی به توزیع داده‌ها و نحوه‌ی آموزش مدل است.

۲. تعیین حساسیت به اشتباهات: تغییر آستانه می‌تواند حساسیت مدل به اشتباهات را تغییر دهد. به عبارت دیگر، اگر مدل شما حساسیت کمتری به اشتباهات بخواهد، می‌تواند آستانه را کاهش دهید و بالعکس.

۳. تغییر معیارهای ارزیابی: تغییر آستانه ممکن است باعث شود که معیارهای ارزیابی مدل مانند دقت (accuracy)، حساسیت (sensitivity) و ویژگی‌های دیگر تغییر کنند.

۴. اهمیت زمینه‌ی کاربردی: در برخی مسائل، اهمیت کاهش یا افزایش تعداد اشتباهات در یک کلاس نسبت به کلاس دیگر ممکن است متفاوت باشد. تغییر آستانه می‌تواند با توجه به اهمیت این موارد متغیر باشد.

تغییر آستانه باید با دقت انجام شود و ممکن است نیاز به ارزیابی جامع تأثیرات آن بر کارایی مدل داشته باشد.

Perceptron with New Threshold

```
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

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

# Separate features and target
X = data.iloc[:, 1:2].values
y = data.iloc[:, 2].values

# 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=13)

# Initialize Perceptron classifier with a different threshold (example: 0.5)
clf = Perceptron(random_state=13, tol=0.001, max_iter=1000, eta0=0.1, verbose=0, n_jobs=-1)

# Train the perceptron with the new threshold
clf.fit(X_train, y_train)

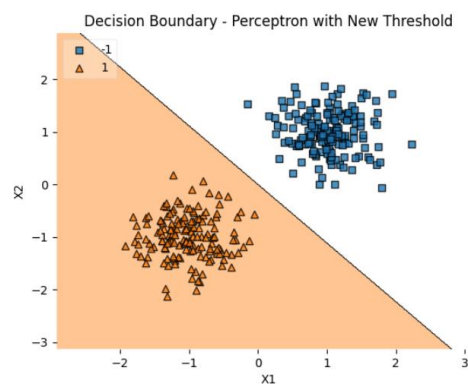
# Accuracy on train and test data with the new threshold
train_accuracy = clf.score(X_train, y_train)
test_accuracy = clf.score(X_test, y_test)

print(f"Accuracy on train set with new threshold: {train_accuracy}")
print(f"Accuracy on test set with new threshold: {test_accuracy}")

# Convert y_train to integer type
y_train = y_train.astype(np.int)

# Plotting decision boundary with the new threshold
plot_decision_regions(X_train, y_train, clf=clf, legend=2)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Decision Boundary - Perceptron with New Threshold')
plt.show()
```

Accuracy on train set with new threshold: 1.0
Accuracy on test set with new threshold: 1.0
<ipython-input-10-8433d61671d2>:32: DeprecationWarning: 'np.int' is a deprecated alias for the builtin 'int'. To silence this warning, use 'int' by itself. Doing this will not modify any behavior and is deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>
y_train = y_train.astype(np.int)



حذف بایاس (bias) در مدل‌های یادگیری ماشین، به طور گسترده تأثیر می‌گذارد. بایاس عبارت است از یک ترم ثابت که به هر نود (ویژگی) در لایه‌های مختلف یک شبکه عصبی افزوده می‌شود. این ترم باعث انتقالی تعمیم‌یافته‌تر از مدل می‌شود و توانایی مدل در تطبیق با داده‌های جدید و ناشناخته را افزایش می‌دهد. حذف بایاس می‌تواند به صورت زیر تأثیر داشته باشد:

۱. کاهش اعتماد به مدل:

حذف بایاس ممکن است باعث کاهش قدرت تعمیم مدل شود و مدل به داده‌های آموزشی خود بسیار نزدیک شود. این باعث می‌شود که مدل در مواجهه با داده‌های جدید و ناشناخته عملکرد ضعیفتری داشته باشد.

۲. حساسیت به تغییرات:

حذف بایاس ممکن است باعث شود که مدل به تغییرات در داده‌ها حساس‌تر شود و توانایی استدلال کمتری داشته باشد. بایاس به مدل امکان می‌دهد تا با توجه به تفاوت‌ها در داده‌ها، بهتر تطبیق پیدا کند.

۳. عدم تعادل در مسئله‌های طبقه‌بندی:

در مسائل طبقه‌بندی، اگر بیشتر داده‌ها به یکی از کلاس‌ها تمایل داشته باشند، حذف بایاس ممکن است منجر به دقت پایین‌تر در کلاس‌های کمتر تعدادی شود. بایاس این مشکل را تا حدی می‌تواند متعادل کند.

۴. سرعت آموزش:

بایاس باعث سرعت آموزش مدل نیز می‌شود. حذف بایاس ممکن است باعث افزایش زمان آموزش شود و مواردی ممکن است به دقت کمتری برسند.

بنابراین، قبل از حذف بایاس از یک مدل، مهم است که تاثیرات آن را بر روی عملکرد مدل بررسی کنید و تصمیم‌گیری به دقت انجام شود. در بسیاری از موارد، استفاده از بایاس بهبود عملکرد و استقلال مدل از خصوصیات خاص داده‌های آموزشی را افزایش می‌دهد.

Perceptron without Bias

```
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

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

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

# 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=13)

# Initialize Perceptron classifier without bias
clf = Perceptron(fit_intercept=False, random_state=13, tol=0.001, max_iter=1000, eta0=0.1, verbose=0, n_jobs=-1)

# Train the perceptron without bias
clf.fit(X_train, y_train)

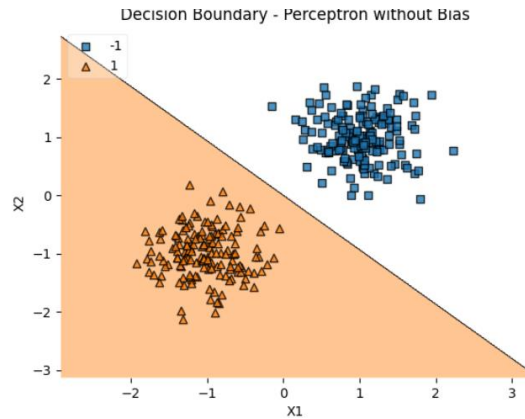
# Accuracy on train and test data without bias
train_accuracy = clf.score(X_train, y_train)
test_accuracy = clf.score(X_test, y_test)

print(f"Accuracy on train set without bias: {train_accuracy}")
print(f"Accuracy on test set without bias: {test_accuracy}")

# Convert y_train to integer type
y_train = y_train.astype(np.int)

# Plotting decision boundary without bias
plot_decision_regions(X_train, y_train, clf=clf, legend=2)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Decision Boundary - Perceptron without Bias')
plt.show()

Accuracy on train set without bias: 1.0
Accuracy on test set without bias: 1.0
<ipython-input-9-95965e0659d7>:32: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  y_train = y_train.astype(np.int)
```



۲.

۲.۱- پس از رسم جدول درستی با استفاده از جدول کارنو معادله هر خروجی را بدست می‌آوریم.

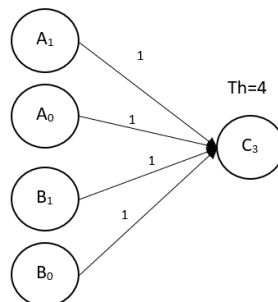
سپس وزن هر ورودی را با توجه به معادله‌ها بدست می‌آوریم. (خود ورودی: $w=1$ و متمم ورودی: $w=-1$) با استفاده از فرمول $net = \sum_{i=1}^n w_i x_i$ مجموع ورودی‌های وزن‌دار شده را بدست آورده و کوچکترین threshold ممکن را بدست می‌آوریم.

در نهایت نیز خواهیم دید می‌توان با $Th=2$ همه‌ی شبکه‌ها را رسم کرد اما از آن جاییکه اولویت کمترین تعداد نرون است، threshold را برای خروجی C_0 برابر ۴ در نظر می‌گیریم.

Table 1. Truth Table

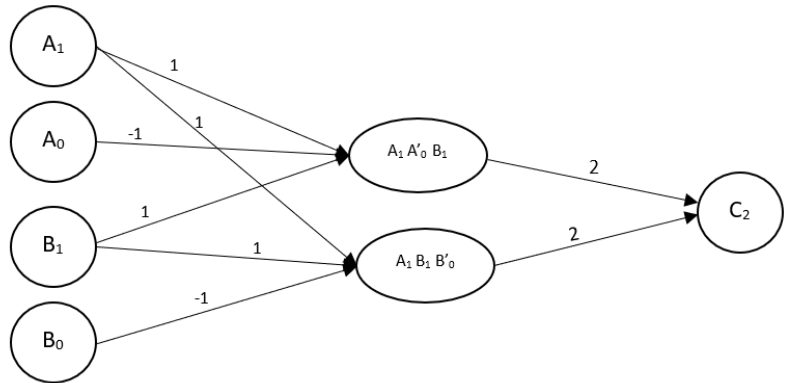
INPUT A		INPUT B		OUTPUT			
A_1	A_0	B_1	B_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

$$C_3 = A_1 A_0 B_1 B_0$$



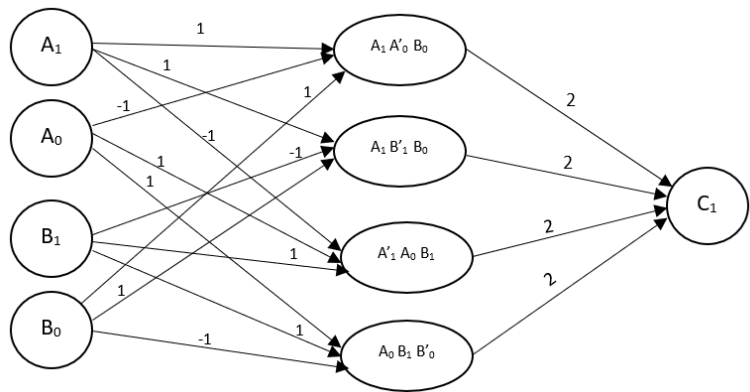
$$C_2 = A_1 A'_0 B_1 + A_1 B_1 B'_0$$

$$Th=2$$

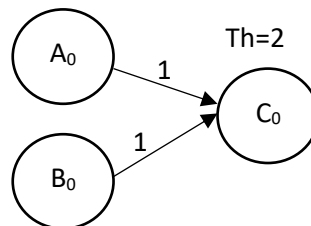


$$C_1 = A'_1 A_0 B_1 + A_0 B_1 B'_0 + A_1 A'_0 B_0 + A_1 B'_1 B_0$$

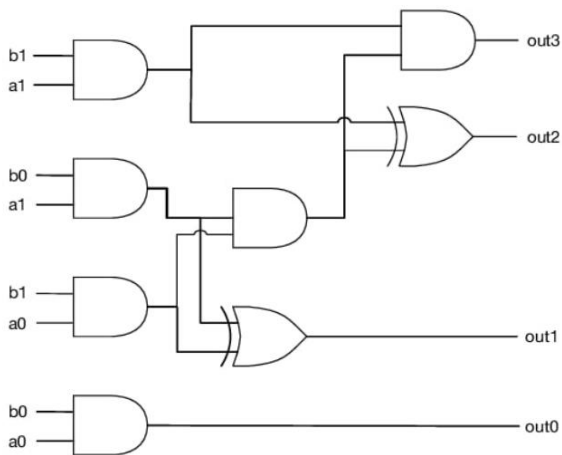
Th=2



$$C_0 = A_0 B_0$$



Logic circuit



-۲.۲

ابتدا یک کلاس برای نرون McCullochPitts می‌سازیم. در متد init آرگومان‌های ورودی اعم از وزن‌ها و آستانه‌ها را تعریف می‌کنیم.

سپس متد model را تعریف می‌کنیم. این متد اگر حاصلضرب وزن‌ها در ورودی‌ها بیشتر از threshold باشد ۱ و در غیر اینصورت ۰ را بر می‌گرداند.(تفکیک کلاس‌ها)

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

#define mukuloch 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 np.dot(self.weights, x) >= self.threshold:
            #if self.weights @ x >= self.threshold:
                return 1
        else:
            return 0
```

برای هر خروجی ما به یک نرون McCullochPitts نیاز داریم. جمعا همرا با نرون های لایه های میانی به ۱۰ نرون نیاز داریم.

برای تولید خروجی نیز باید از متد model استفاده کنیم.(انجام محاسبات)

Return خروجی ۴ بیتی مدل ما را برمی‌گرداند.

```
#define model for dataset
def BiMultiplier(input):
    neur1 = McCulloch_Pitts_neuron([1, 1, 1, 1] , 4)    #c3
    neur2 = McCulloch_Pitts_neuron([1, -1, 1, 0] , 2)    #a1a0'b1
    neur3 = McCulloch_Pitts_neuron([1, 0, 1, -1] , 2)    #a1b1b0'
    neur4 = McCulloch_Pitts_neuron([2, 2] , 2)    #c2
    neur5 = McCulloch_Pitts_neuron([1, -1, 0, 1] , 2)    #a1a0'b0
    neur6 = McCulloch_Pitts_neuron([1, 0, -1, 1] , 2)    #a1b1'b0
    neur7 = McCulloch_Pitts_neuron([-1, 1, 1, 0] , 2)    #a1'a0b1
    neur8 = McCulloch_Pitts_neuron([0, 1, 1, -1] , 2)    #a0b1b0'
    neur9 = McCulloch_Pitts_neuron([2, 2, 2, 2] , 2)    #c1
    neur10 = McCulloch_Pitts_neuron([0, 1, 0, 1] , 2)    #c0

    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([z2, z3]))
    z5 = neur5.model(np.array([input[0], input[1], input[2], input[3]]))
    z6 = neur6.model(np.array([input[0], input[1], input[2], input[3]]))
    z7 = neur7.model(np.array([input[0], input[1], input[2], input[3]]))
    z8 = neur8.model(np.array([input[0], input[1], input[2], input[3]]))
    z9 = neur9.model(np.array([z5, z6, z7, z8]))
    z10 = neur10.model(np.array([input[0], input[1], input[2], input[3]]))

    # 4 bit output
    return list([z1,z4,z9,z10])
```

و برای نمایش خارجی از دستورات زیر استفاده می‌کنیم.

```
import itertools
# inputs
input = [1, 0]
X = list(itertools.product(input, input, input, input))
for i in X:
    res = BiMultiplier(i)
    print("BiMultiplier with input as", ''.join(map(str, i)), "goes to output ", ''.join(map(str, res)))
```

و در نتیجه داریم:

```

BiMultiplier with input as 1111 goes to output 1001
BiMultiplier with input as 1110 goes to output 0110
BiMultiplier with input as 1101 goes to output 0011
BiMultiplier with input as 1100 goes to output 0000
BiMultiplier with input as 1011 goes to output 0110
BiMultiplier with input as 1010 goes to output 0100
BiMultiplier with input as 1001 goes to output 0010
BiMultiplier with input as 1000 goes to output 0000
BiMultiplier with input as 0111 goes to output 0011
BiMultiplier with input as 0110 goes to output 0010
BiMultiplier with input as 0101 goes to output 0001
BiMultiplier with input as 0100 goes to output 0000
BiMultiplier with input as 0011 goes to output 0000
BiMultiplier with input as 0010 goes to output 0000
BiMultiplier with input as 0001 goes to output 0000
BiMultiplier with input as 0000 goes to output 0000

```

که نتیجه مشاهده شده در خروجی همان جدول درستی را نشان می‌دهد.

۳.

۳.۱-

نحوه عملکرد تابع اول (convertImageToBinary):

به طور خلاصه، این تابع یک فایل تصویر را به عنوان ورودی دریافت می‌کند، هر پیکسل را بر اساس شدت آن پردازش می‌کند، و یک نمایش دودویی از تصویر را باز می‌گرداند. این نمایش دودویی یک لیست است که هر عنصر آن به یک پیکسل متناظر است و مقدار آن ۱ یا ۰- (سفید) یا ۱ (سیاه) است.

نحوه عملکرد تابع دوم (generateNoisyImages)

این تابع به عنوان بخشی از پردازش تصویر، تصاویر ورودی را با اضافه کردن نویز تصادفی تغییر می‌دهد و تصاویر نویزی را در فرمت JPEG ذخیره می‌کند. توضیحات عملکرد تابع به شرح زیر است:

ابتدا لیست مسیر فایل تصاویر تعیین شده، سپس با استفاده از توابع مورد نیاز برای بارگیری و ترسیم تصاویر استفاده می‌شود. مقدار نویز تصادفی به هر پیکسل افزوده می‌شود. تصاویر نویزی با استفاده از تابع `image.save` در فرمت JPEG ذخیره می‌شوند. مسیر و نام فایل‌های جدید بر اساس تصاویر اصلی وارد شده به تابع تعیین می‌شوند. پس از ایجاد هر تصویر نویزی، یک پیام چاپ می‌شود تا اعلام کند که تصویر نویزی برای تصویر ورودی ایجاد و ذخیره شده است.

به طور کلی، این تابع به تصاویر ورودی نویز افزوده و تصاویر نویزی را ذخیره می‌کند تا برای آزمایش و تحلیل الگوریتم‌ها یا مدل‌های مختلف استفاده شوند.

۳.۳-

با الهام از تابع دوم تابع جدیدی با اسم generateMissedPointImage ساختیم که این تابع همانند تابع قبلی یک مقدار رندوم را به مقادیر قرمز و سبز و آبی اضافه می‌کند با این تفاوت که مقادیر رندوم فقط زمانی اضافه می‌شوند که پیکسل مدنظر مشکی بوده باشد. (=۱)

در واقع در این حالت گویی برخی از پیکسل‌های مشکی را سفید در نظر گرفتیم به همین دلیل داده دچار اختشاش شده است.


```

from PIL import Image, ImageDraw
import random

def generateMissedPointImages():
    # 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):
        missedpoint_image_path = f"/content/missedpoint{i}.jpg"
        getMissedPointBinaryImage(image_path, missedpoint_image_path)
        print(f"MissedPoint image for {image_path} generated and saved as {missedpoint_image_path}")

def getMissedPointBinaryImage(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 missedpoint 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 missedpoint.
missedpoint_factor = 10000000

# Loop through all pixels in the image.
for i in range(width):
    for j in range(height):

        # Generate a random missedpoint value within the specified factor.
        rand = random.randint(-missedpoint_factor, missedpoint_factor)

        # Add the missedpoint to 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]
        if red == 0 & green == 0 & blue == 0:
            red = red + rand
            green = green + rand
            blue = blue + 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

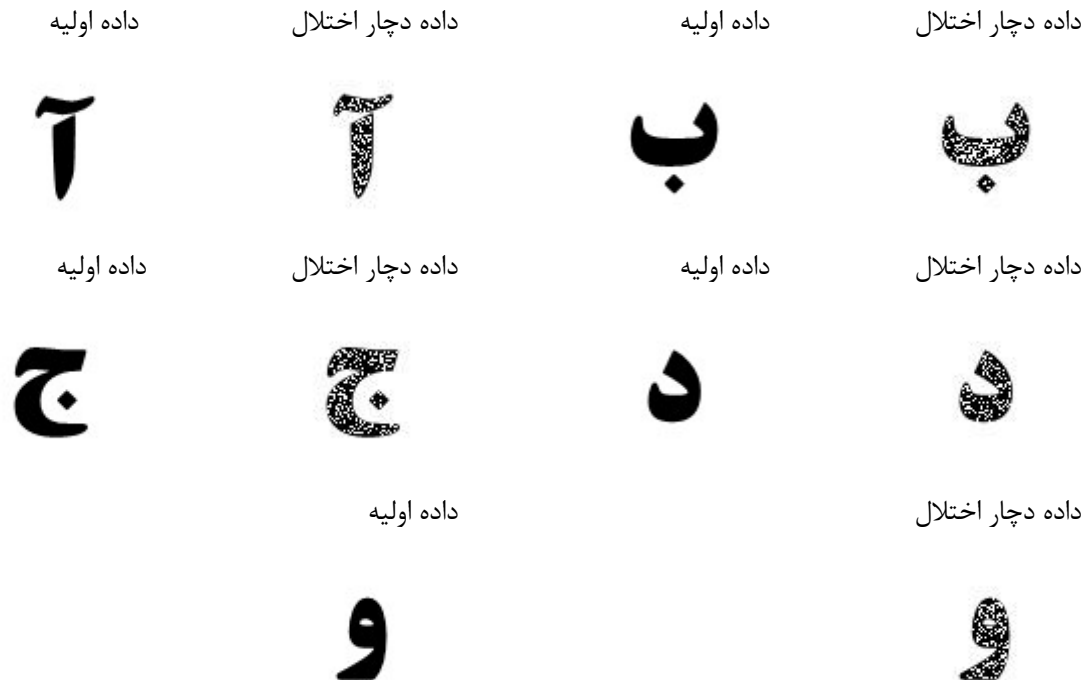
# Generate missedpoint images and save them
generateMissedPointImages()

```

```

MissedPoint image for /content/1.jpg generated and saved as /content/missedpoint1.jpg
MissedPoint image for /content/2.jpg generated and saved as /content/missedpoint2.jpg
MissedPoint image for /content/3.jpg generated and saved as /content/missedpoint3.jpg
MissedPoint image for /content/4.jpg generated and saved as /content/missedpoint4.jpg
MissedPoint image for /content/5.jpg generated and saved as /content/missedpoint5.jpg

```



۴.

مجموعه داده مد نظر را با روش گفته شده در کلاس (دستور gdown) در محیط گوگل کولب بارگزاری می‌کنیم و آن را در پوشه data قرار می‌دهیم. (در این مجموعه داده ستون price هدف یا target و باقی ستون‌ها ویژگی هستند).

```
import os

#define the folder name
folder_name = "Data"

#check if the folder already exists, and create it if not
if not os.path.exists(folder_name):
    os.makedirs(folder_name)

%cd Data

!pip install --upgrade --no-cache-dir gdown
!gdown 1v10LDTXqy28Jhy22A70e_KjqhFdoA3xA
```

-۴.۱

حال مجموعه داده را بصورت دیتافریم تبدیل می‌کنیم، سپس با دستور df.info() تابع info را از کتابخانه pandas فراخوانی می‌کنیم. با استفاده از این دستور داده‌های همه‌ی ستون‌ها را مشاهده می‌کنیم.

```
import pandas as pd
import numpy as np
#import matplotlib.pyplot as plt
#from sklearn.model_selection import train_test_split

#upload dataset as dataframe
df = pd.read_csv('/content/Data/data.csv')

# to print the full summary
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
```

۴۶۰۰ داده

۱۸ ستون

منظور از NAN، Not a Number یا همان داده یا همان داده null است. مثل ۰/۰. برای نمایش این داده‌ها از دستور `df.isnull()` استفاده می‌کنیم. اگر مقدار ۰ برگردانده شود یعنی ستون هیچ داده NAN ای ندارد و اگر ۱ برگرداند یعنی دارای داده NAN است.

```
# Count NaN values in each column
df.isnull().sum()
```

همانطور که مشاهده می‌شود در این مجموعه داده هیچ داده

NAN ای نداریم.

```
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
```

برای رسم ماتریس همبستگی نیاز است ابتدا ستون‌هایی که داده‌های غیر عددی دارند را به حالت عددی تبدیل کنیم. با دستور `pd.factorize` می‌توان این کار را انجام داد. این دستور به این صورت عمل می‌کند که مقادیر یک آرایه را به یک `tuple` با دو عضو تبدیل می‌کند. عضو اول مربوط به فاکتورهای عددی منسوب به ورودی و عضو دوم شامل لیست مقادیر یکتای موجود در آرایه‌ی ورودی است.

```
import pandas as pd

#upload dataset as dataframe
df = pd.read_csv('/content/Data/data.csv')

#iterate over each column in the dataframe
for col in df.columns:
    #check if column has non-numeric data
    if df[col].dtype == 'object':
        #encode values as integer
        df[col] = pd.factorize(df[col])[0]

#save new dataframe
df.to_csv('/content/Data/data2.csv', index=False)
```

حال برای ساخت ماتریس همبستگی از دستور `df.corr` استفاده می‌کنیم.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

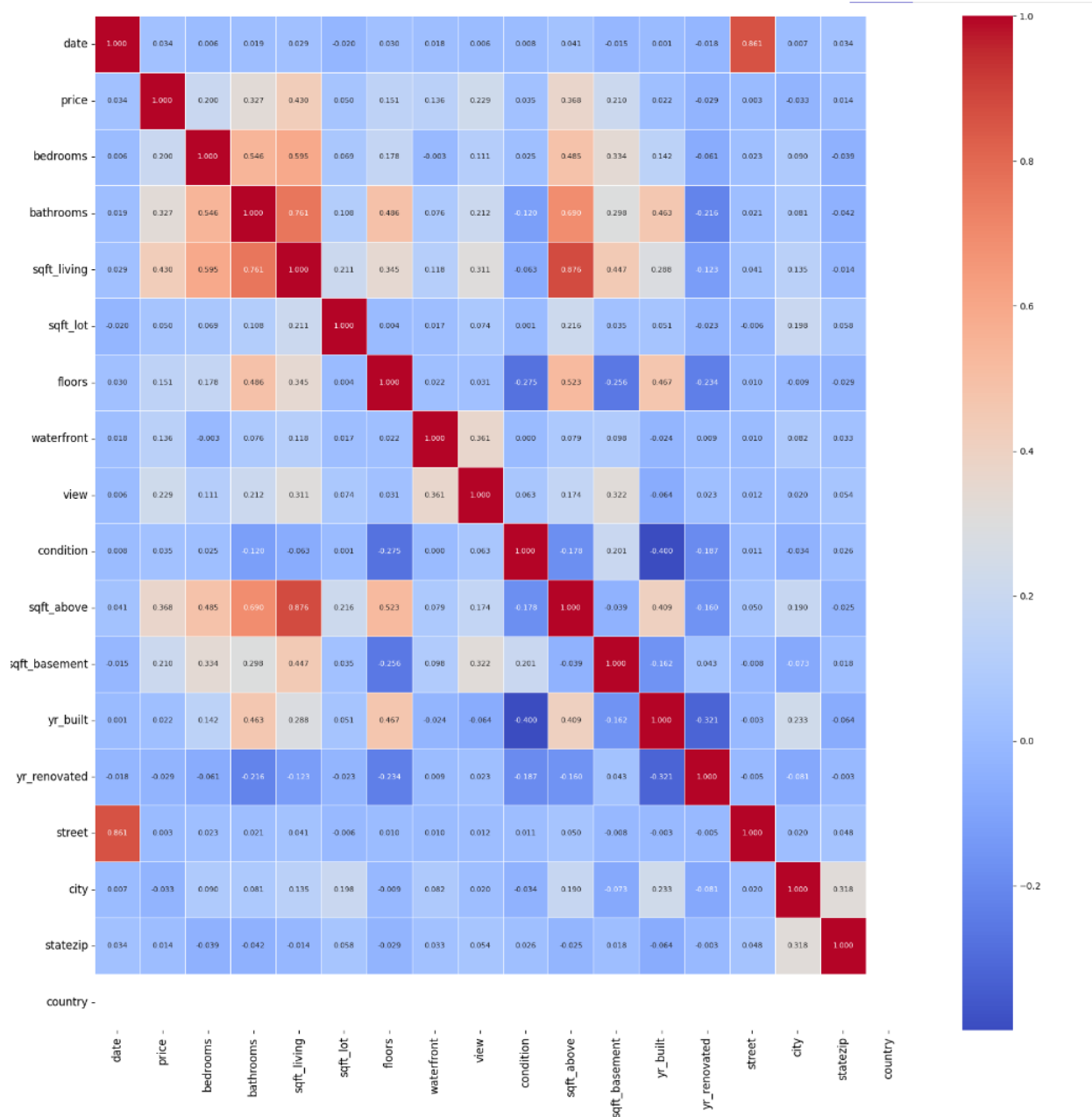
# Read CSV file into DataFrame
df = pd.read_csv('/content/Data/data2.csv')

# Calculate correlation matrix
corr_matrix = df.corr()

# Create heatmap using seaborn
plt.figure(figsize=(20,20))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5, annot_kws={"size": 8}, fmt='.3f', yticklabels=corr_matrix.columns)

# Adjust font size of annotations
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Adjust margins of PDF file
plt.savefig('PICS1.pdf', bbox_inches='tight')
```



برای نمایش مقدار همبستگی متغیر هر ستون با قیمت از دستورات زیر استفاده می‌کنیم.

```

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Read CSV file into DataFrame
df = pd.read_csv('/content/Data/data2.csv')

# Select columns to include in correlation matrix
cols = df.columns.tolist()
cols.remove('price')

# Calculate correlation matrix
corr_matrix = df[cols].corrwith(df['price']).sort_values(ascending=False)

# Create heatmap using seaborn
plt.figure(figsize=(2,30))
sns.heatmap(corr_matrix.to_frame(), annot=True, cmap='coolwarm', linewidths=0.5, annot_kws={"size": 12}, fmt='.3f', cbar=False)

# Rotate x-axis tick labels to be horizontal
plt.xticks(rotation=0)

# Adjust font size of annotations
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Adjust margins of PDF file
plt.tight_layout()
plt.savefig('priceCM1.pdf', bbox_inches='tight')

```

با توجه به نمودار زیر بیشترین همبستگی ستون قیمت به جز خودش(=1) با متغیر sqft_living(0.43) است.



-۴.۳

با استفاده از تابع hist توزیع قیمت خانه‌ها را نمایش می‌دهیم. محور افقی قیمت و محور عمودی تعداد خانه‌هایی است که در هر بازه قیمتی وجود دارند. با تابع boxplot نیز قیمت خانه‌ها بصورت یک جعبه نمایش داده می‌شود.

```

import pandas as pd
import matplotlib.pyplot as plt

# Load the CSV file into a pandas dataframe
df = pd.read_csv('/content/Data/data2.csv')

# Plot the distribution of the 'price' column as a histogram
plt.figure(figsize=(8, 6)) # Set the size of the figure
plt.subplot(2, 1, 1) # Create the first subplot
plt.hist(df['price'], bins=100, color='blue')
plt.xlabel('Price')
plt.ylabel('Count')
plt.title('Distribution of Prices')

# Plot a boxplot of the 'price' column
plt.subplot(2, 1, 2) # Create the second subplot
plt.boxplot(df['price'], vert=False, width=0.7)
plt.xlabel('Price')
plt.title('Boxplot of Prices')

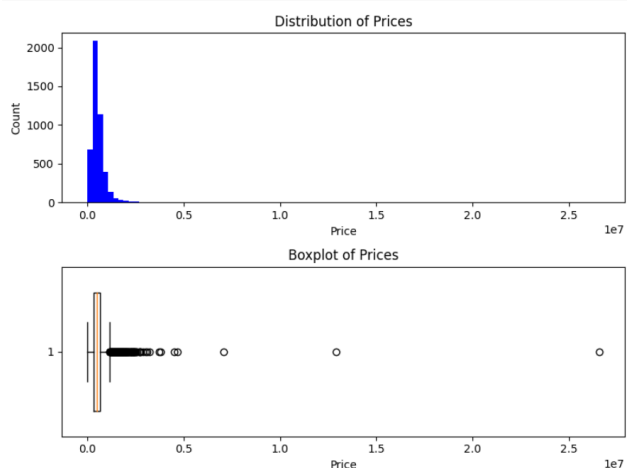
plt.tight_layout() # Automatically adjust subplot parameters

# Save the plot as a PDF file with a fit margin
plt.savefig('dist1.pdf', bbox_inches='tight')
plt.show()

# Define the bins for the price ranges
price_ranges = pd.cut(df['price'], bins=range(0, 1200000, 8000))
# Calculate the count of prices in each range
price_counts = price_ranges.value_counts().sort_index()
# Display the price range counts
print(price_counts)

```

نمودار توزیع آماری قیمت



با دستور `pd.cut` قیمت خانه‌ها را به بازه‌های ۸۰۰۰ دلاری تقسیم کردیم. غالب خانه‌ها در بازه قیمتی ۲۲۴۰۰۰ تا ۲۸۰۰۰۰ قرار گرفتند.

```
(224000, 232000]    53
(232000, 240000]    74
(240000, 248000]    50
(248000, 256000]    93
(256000, 264000]    52
(264000, 272000]    72
(272000, 280000]    66
```

از قسمت قبل می‌دانیم قیمت با `sqft_living` بیشترین همبستگی را دارد. برای رسم نمودار از دستور `seaborn` استفاده می‌کنیم. با دستور `sns.regplot` میانگین خطی بین دو متغیر را به نمودار اضافه می‌کنیم. هر نقطه نمودار نشان‌دهنده یک خانه و موقعیت آن نشان‌دهنده قیمت خانه بر حسب `sqft-living` است.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

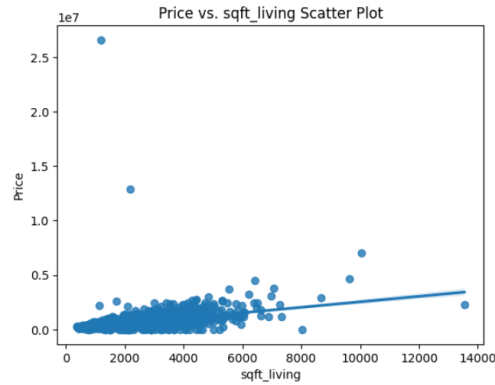
# Load the CSV file into a pandas dataframe
df = pd.read_csv('/content/Data/data2.csv')

# Create a scatter plot of 'price' versus 'sqft_living'
sns.regplot(x=df['sqft_living'], y=df['price'])
plt.xlabel('sqft_living')
plt.ylabel('Price')
plt.title('Price vs. sqft_living Scatter Plot')

# Save the plot as a PDF file with a fit margin
plt.savefig('pricevsqft_living.pdf', bbox_inches='tight')

plt.show()
```

نمودار قیمت بر حسب ویژگی با بیشترین همبستگی



-۴.۴

قسمت اول ستون Date را برای ستون جدید سال و قسمت دوم آن را برای ستون جدید ماه در نظر گرفته و ستون date را حذف می‌کنیم. سپس مجموعه داده جدیدی را تشکیل می‌دهیم.

```
import pandas as pd
import difflib

# list of valid month and year names
valid_names = ['04', '05', '06', '07', '2014']

# read the original csv file into a pandas dataframe
df = pd.read_csv('/content/Data/data2.csv')

# extract the first word of each date into a new year column
df['year'] = df['date'].str.split().str[0]

# replace any invalid names with the nearest match from the list
df['year'] = df['date'].apply(lambda x: difflib.get_close_matches(x, valid_names, n=1)[0])

# extract the second word of each date into a new month column
df['month'] = df['date'].str.split().str[1]

# replace any invalid names with the nearest match from the list
df['month'] = df['date'].apply(lambda x: difflib.get_close_matches(x, valid_names, n=1)[0])

cols = df.columns.tolist()
cols = cols[-1:] + cols[:-1]
df = df[cols]

# drop the date column
df = df.drop(['date'], axis=1)

# save the modified dataframe to a new csv file
df.to_csv('C:/content/Data/data3.csv', index=False)
```

-۴.۵

داده‌های ستون قیمت بعنوان هدف و بقیه ستون‌ها ویژگی هستند. داده‌ها را به نسبت ۸۰ به ۲۰ پس از مخلوط کردن به دو بخش آموزش و آزمون تقسیم می‌کنیم.

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read data from CSV file into a Pandas dataframe
data = pd.read_csv('/content/Data/data2.csv')

# Split the data into train and test sets
train_data, test_data, train_label, test_label = train_test_split(data.drop(['price'], axis=1), data['price'], test_size=0.2, random_state=13)

# Save train and test data to new CSV files
train_data.to_csv('train_data.csv', index=False)
test_data.to_csv('test_data.csv', index=False)

# Save train and test labels to new CSV files
train_label.to_csv('train_label.csv', index=False)
test_label.to_csv('test_label.csv', index=False)

# Print shape of each set
print(f"train_data shape: {train_data.shape}")
print(f"test_data shape: {test_data.shape}")
print(f"train_label shape: {train_label.shape}")
print(f"test_label shape: {test_label.shape}")

train_data shape: (3680, 17)
test_data shape: (920, 17)
train_label shape: (3680,)
test_label shape: (920,)
```

با استفاده از MinMaxScaler داده‌ها را بین ۰ و ۱ مقیاس می‌کنیم. داده‌های مجموعه آزمون نباید در این مراحل حضور داشته باشند زیرا تقلب صورت می‌گیرد.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# read csv file into a pandas dataframe
data = pd.read_csv('/content/Data/data2.csv')

# extract label column as y
y = data['price']

# extract all other columns as X
X = data.drop(columns=['price'])

# split data into train and test sets with a 80/20 split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=13)

# scale the training data
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)

# scale the test data using the same scaler used for training data
X_test_scaled = scaler.transform(X_test)

# save train and test data/label in new files
pd.DataFrame(X_train_scaled).to_csv('Xtrain.csv', index=False)
y_train.to_csv('ytrain.csv', index=False)
pd.DataFrame(X_test_scaled).to_csv('Xtest.csv', index=False)
y_test.to_csv('ytest.csv', index=False)
```

-۴.۶

در این قسمت یک مدل شبکه عصبی چندلایه تعریف می‌کنیم. هر مدل شامل لایه خطی است که با تابع فعال‌ساز غیرخطی ReLU به هم وصل شدند. این مدل را با یک کلاس از `nn.module` پایتورچ انجام دادیم. تابع سازنده مدل ورودی را برای تعریف مدل دریافت می‌کند و لایه مدنظر را با استفاده از `n.linear` تعریف می‌کند. این مدل یک تابع پیش‌بینی به سمت جلو دارد که با دریافت ورودی ابتدا خروجی هر لایه خطی با استفاده از تابع فعال‌ساز محاسبه شده و سپس خروجی نهایی برگردانده می‌شود.

```
import torch
import torch.nn as nn

# Define the MLP models
class MLP2(nn.Module):
    def __init__(self, input_dim, hidden_dim1, hidden_dim2, output_dim):
        super(MLP2, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim1)
        self.fc2 = nn.Linear(hidden_dim1, hidden_dim2)
        self.fc3 = nn.Linear(hidden_dim2, output_dim)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

افزودن مجموعه اعتبارسنجی

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Read the CSV file
data = pd.read_csv('/content/Data/data2.csv')

# Separate the label and data
label = df['price']
data = df.drop('price', axis=1)

# Remove the header from the variables
data = data.values
label = label.values

# Split the data into train/validation/test sets
train_data, test_data, train_label, test_label = train_test_split(data, label, test_size=0.2, random_state=13)
train_data, val_data, train_label, val_label = train_test_split(train_data, train_label, test_size=0.2, random_state=13)

# Scale the data using MinMaxScaler
scaler = MinMaxScaler()
train_data = scaler.fit_transform(train_data)
val_data = scaler.transform(val_data)
test_data = scaler.transform(test_data)

# Print shape of each set
print(f"train_data shape: {train_data.shape}")
print(f"val_data shape: {val_data.shape}")
print(f"test_data shape: {test_data.shape}")
print(f"train_label shape: {train_label.shape}")
print(f"val_label shape: {val_label.shape}")
print(f"test_label shape: {test_label.shape}")

# Save the train/validation/test data/label to new files
pd.DataFrame(train_data).to_csv('/content/finaltrain_data.csv', index=False, header=False)
pd.DataFrame(val_data).to_csv('/content/finalval_data.csv', index=False, header=False)
pd.DataFrame(test_data).to_csv('/content/finaltest_data.csv', index=False, header=False)
pd.DataFrame(train_label).to_csv('/content/finaltrain_label.csv', index=False, header=False)
pd.DataFrame(val_label).to_csv('/content/finalval_label.csv', index=False, header=False)
pd.DataFrame(test_label).to_csv('/content/finaltest_label.csv', index=False, header=False)

train_data shape: (2944, 17)
val_data shape: (736, 17)
test_data shape: (920, 17)
train_label shape: (2944,)
val_label shape: (736,)
test_label shape: (920,)
```

۵.

توضیحات مربوط به دیتاست

مجموعه داده گل زنبق (به انگلیسی Iris flower data set): یا مجموعه داده زنبق فیشر یک مجموعه داده چند متغیره است که توسط رانلد فیشر، آماردان و زیست‌شناس بریتانیایی در سال ۱۹۳۶ معرفی شد. این مجموعه داده همچنین مجموعه داده زنبق اندرسون نیز نامیده می‌شود.

این مجموعه شامل ۱۵۰ نمونه‌ی جمع‌آوری شده از گل‌های زنبق است که این نمونه‌ها ۵۰ نمونه از هر یک از سه نوع گل زنبق را شامل می‌شوند. برای هر یک از نمونه‌ها ۴ ویژگی گل زنبق اندازه‌گیری شده‌است. این ویژگی‌ها شامل طول و عرض کاسبرگ و

گلبرگ، بر حسب سانتی متر است. بر اساس ترکیبی از این چهار ویژگی، فیشر یک مدل تشخیص خطی برای تفکیک کردن گونه‌های این گل از یکدیگر ایجاد کرد.

این مجموعه داده به عنوان یک مثال پرکاربرد در زمینه‌های آماری و یادگیری ماشین مورد استفاده قرار گرفته‌است.

این مجموعه داده در روش‌های خوشه‌بندی معمولاً مورد استفاده قرار نمی‌گیرد. دلیل این مسئله آن است که داده‌های موجود در این مجموعه هنگام نمایش در فضا فقط دو خوشه‌ی مشخص از سه خوشه را نمایش می‌دهند و داده‌های مربوط به دو کلاس در یک دسته خوشه‌بندی می‌شوند.

مجموعه داده گل زنبق، اطلاعات مربوط به سه نوع از گل‌های زنبق از جمله زنبق نوک‌زبر، زنبق رنگارنگ و زنبق ویرجینیا را شامل می‌شوند که با اعمال خوشه‌بندی بر روی این مجموعه داده، یکی از خوشه‌ها حاوی نمونه‌های مربوط به زنبق نوک‌زبر و خوشه‌ی دیگر حاوی نمونه‌های مربوط به هر دو گونه زنبق رنگارنگ و زنبق ویرجینیا خواهد بود.

با این وجود هر سه گونه‌ی این مجموعه داده با استفاده از نگاشت غیرخطی به فضایی دیگر، قابل تفکیک هستند.

تعداد نمونه‌ها = ۱۵۰ تا

تعداد ویژگی‌ها = ۴ عدد (طول کاسبرگ، عرض کاسبرگ، طول گلبرگ و عرض گلبرگ)

تعداد کلاس‌ها = ستوسا، ورسیکالر و ویرجینیکا

از بین ۱۵۰ نمونه، ۵۰ تای اول در دسته ستوسا، دوم در کلاس ورسیکالر و سوم در ویرجینیکا قرار دارند. محض اطمینان، ابتدا داده‌ها را بر می‌زنیم و سپس با نسبت ۸۰ به ۲۰ درصد به دو قسمت آموزش و آزمون تقسیم می‌کنیم.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

#shuffle
df = shuffle(df)

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

# 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=13)
df, X, y, X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

با استفاده از کتابخانه Pandas فایل CSV دیتاست را آپلود خواهیم کرد، و آن را به دیتافریم تبدیل می‌کنیم. متد `read_csv()` را برای خواندن فایل دیتاست استفاده می‌کنیم.

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1bDuT0miTwHPS087HP8jr3zSFP1wy6307
```

```
import pandas as pd

# Reading the CSV file
df = pd.read_csv('/content/iris.csv')

# Printing top 5 rows
df.head()
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

از پارامتر `shape` برای دریافت ابعاد مجموعه داده استفاده خواهیم کرد. می‌توانیم ببینیم که دیتافریم شامل ۵ ستون و ۱۵۰ ردیف می‌باشد.

```
df.shape
```

```
(150, 5)
```

حالا ستون‌ها و انواع داده‌های آن‌ها را بررسی می‌کنیم. برای این کار، از متد `info()` استفاده خواهیم کرد. می‌توانیم ببینیم که تنها یک ستون داده‌های دسته‌ای (`categorical`) دارد و سایر ستون‌ها از نوع عددی هستند و همگی دارای مقادیر غیر-تهی (`non-Null`) هستند.

```
df.info()
```

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

یک خلاصه آماری از مجموعه داده با استفاده از متد `describe()` می‌گیریم. تابع `describe()` محاسبات آماری ابتدایی را بر روی مجموعه داده اعمال می‌کند، از جمله مقادیر بیشینه و کمینه، تعداد نقاط داده، انحراف معیار و غیره. هر مقدار ناپدید یا `NaN` به طور خودکار نادیده گرفته می‌شود. تابع `describe()` یک تصویر خوب از توزیع داده ارائه می‌دهد. در اینجا می‌توانیم تعداد هر ستون به همراه میانگین، انحراف معیار، حداقل و حداکثر مقادیر را ببینیم.

```
df.describe()
```

	sepal.length	sepal.width	petal.length	petal.width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

بررسی می‌کنیم که آیا داده‌های ما حاوی مقادیر ناپدید (missing values) هستند یا خیر. مقادیر ناپدید ممکن است زمانی اتفاق بیافتند که برای یک یا چند مورد از داده‌ها یا یک واحد کلی اطلاعاتی فراهم نشده باشد. برای این کار، از متد `isnull()` استفاده خواهیم کرد. می‌توانیم ببینیم که هیچ ستونی حاوی مقدار ناپدید نیست.

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

```
sepal.length    0
sepal.width     0
petal.length    0
petal.width     0
variety         0
dtype: int64
```

بررسی می‌کنیم که آیا مجموعه داده حاوی موارد تکراری است یا خیر. متد `drop_duplicates()` در Pandas به حذف موارد تکراری از دیتافریم کمک می‌کند. می‌بینیم که تنها سه گونه منحصر به فرد وجود دارد.

```
data = df.drop_duplicates(subset = "variety",)
data
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
50	7.0	3.2	4.7	1.4	Versicolor
100	6.3	3.3	6.0	2.5	Virginica

بررسی می‌کنیم که آیا مجموعه داده متوازن است یا خیر، به عبارت دیگر، آیا تمام گونه‌ها حاوی تعداد یکسانی از ردیف‌ها هستند یا نه. برای این کار، از تابع `Series.value_counts()` استفاده خواهیم کرد. این تابع یک سری (Series) را با شمارش مقادیر منحصر به فرد باز می‌گرداند. می‌بینیم که تمام گونه‌ها حاوی تعداد یکسانی از ردیف‌ها هستند، بنابراین نباید هیچ ورودی را حذف کنیم.

```
df.value_counts("variety")
```

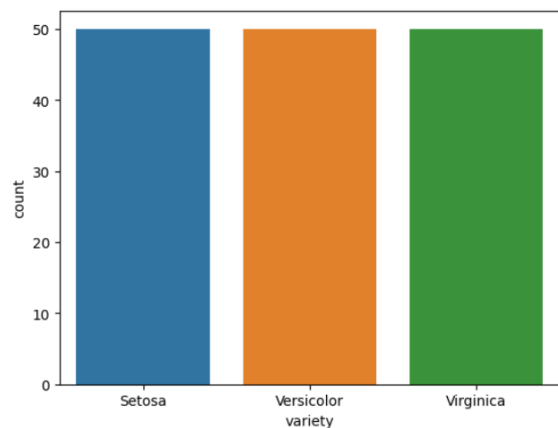
```
variety
Setosa      50
Versicolor  50
Virginica   50
dtype: int64
```

نمایش ستون هدف

ستون هدف ما ستون **variety** خواهد بود، زیرا در نهایت ما به نتایج بر اساس گونه‌ها نیاز خواهیم داشت. یک نمودار **countplot** برای این ستون می‌بینیم.

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='variety', data=df, )
plt.show()
```



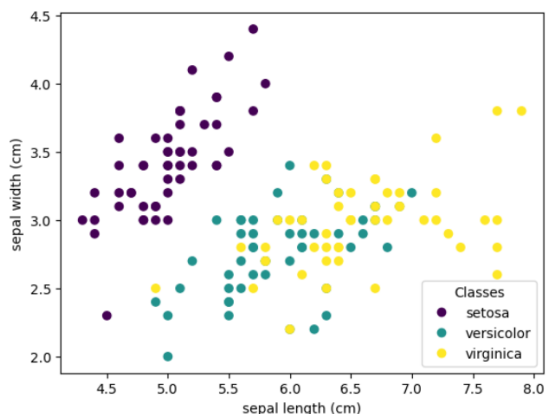
رابطه بین متغیرها

در این قسمت رابطه بین طول کاسبرگ و عرض کاسبرگ، و همچنین بین طول گلبرگ و عرض گلبرگ را خواهیم دید.

مقایسه طول کاسبرگ و عرض کاسبرگ

```
import matplotlib.pyplot as plt

_, ax = plt.subplots()
scatter = ax.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
ax.set(xlabel=iris.feature_names[0], ylabel=iris.feature_names[1])
_ = ax.legend(
    scatter.legend_elements()[0], iris.target_names, loc="lower right", title="Classes"
)
```

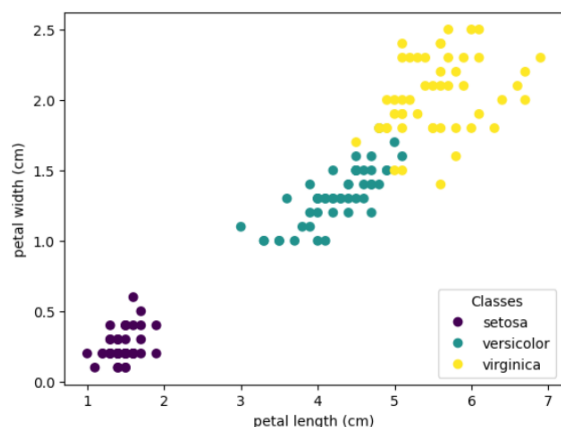


از نمودار فوق می‌توانیم استنباط کنیم:

- گونه Setosa دارای طول کاسبرگ کوچکتر اما عرض کاسبرگ بزرگتری است.
- گونه Versicolor در میان دو گونه دیگر از نظر طول و عرض کاسبرگ قرار دارد.
- گونه Virginica دارای طول کاسبرگ بزرگتر اما عرض کاسبرگ کوچکتری است.

مقایسه طول گلبرگ و عرض گلبرگ

```
_, ax = plt.subplots()
scatter = ax.scatter(iris.data[:, 2], iris.data[:, 3], c=iris.target)
ax.set(xlabel=iris.feature_names[2], ylabel=iris.feature_names[3])
_ = ax.legend(
    scatter.legend_elements()[0], iris.target_names, loc="lower right", title="Classes"
)
```



از نمودار فوق می‌توانیم استنباط کنیم:

- گونه Setosa دارای طول و عرض گلبرگ کوچکتری است.
 - گونه Versicolor در میان دو گونه دیگر از نظر طول و عرض گلبرگ قرار دارد.
 - گونه Virginica دارای بزرگترین طول و عرض گلبرگ است.
- مشاهده می‌کنیم که جداسازی کلاس ستوسا از دو کلاس دیگر است. جداسازی کلاس‌ها با طول و عرض کاسبرگ دشوارتر است و داده‌ها همپوشانی بیشتری دارند. (به خصوص در کلاس‌های ورسیکالر و ویرجینیکا) اما با ویژگی‌های طول و عرض گلبرگ جداسازی آسان‌تر است.

هیستوگرام

با استفاده از هیستوگرام‌ها می‌توانیم توزیع داده‌ها برای ستون‌های مختلف را ببینیم. این می‌تواند برای تحلیل یک متغیره و یا بی‌متغیره استفاده شود.

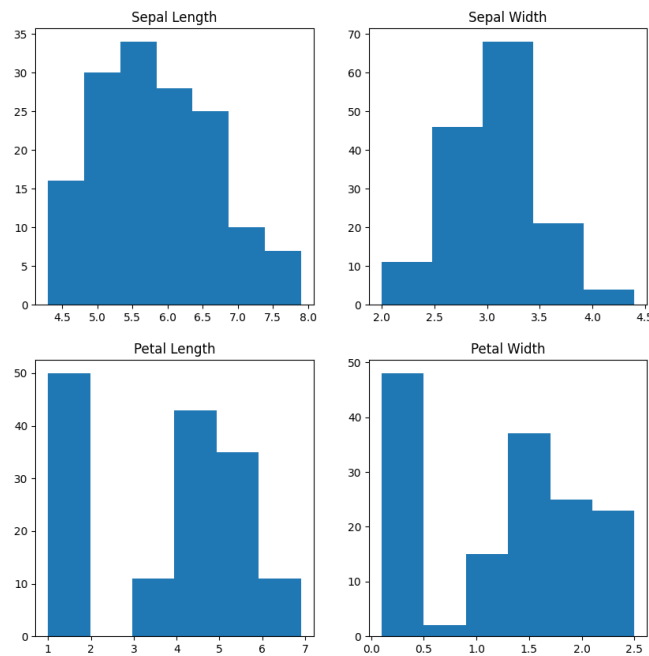

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 2, figsize=(10,10))

axes[0,0].set_title("Sepal Length")
axes[0,0].hist(df['sepal.length'], bins=7)
axes[0,1].set_title("Sepal Width")
axes[0,1].hist(df['sepal.width'], bins=5);

axes[1,0].set_title("Petal Length")
axes[1,0].hist(df['petal.length'], bins=6);

axes[1,1].set_title("Petal Width")
axes[1,1].hist(df['petal.width'], bins=6);
```



از نمودار فوق می‌توانیم ببینیم:

- بیشترین فراوانی برای طول کاسبرگ بین ۳۰ و ۳۵ قرار دارد که معادل با ۵.۵ تا ۶ است.
- بیشترین فراوانی برای عرض کاسبرگ در حدود ۷۰ است که معادل با ۳.۰ تا ۳.۵ است.
- بیشترین فراوانی برای طول گلبرگ حدود ۵۰ است که معادل با ۱ تا ۲ است.
- بیشترین فراوانی برای عرض گلبرگ بین ۴۰ و ۵۰ قرار دارد که معادل با ۰.۰ تا ۰.۵ است.

همبستگی

در Pandas، متد `dataframe.corr()` برای یافتن همبستگی دو به دوی تمام ستون‌ها در دیتافریم استفاده می‌شود. هر مقدار NA به طور خودکار حذف می‌شود. برای ستون‌هایی که نوع داده‌ای غیر عددی دارند، این متد نادیده گرفته می‌شود.

```
data.corr(method='pearson')
```

```
<ipython-input-34-c50c7eb58c83>:1: FutureWarning: The default value of numeric_only i
data.corr(method='pearson')
```

	sepal.length	sepal.width	petal.length	petal.width
sepal.length	1.000000	-0.999226	0.795795	0.643817
sepal.width	-0.999226	1.000000	-0.818999	-0.673417
petal.length	0.795795	-0.818999	1.000000	0.975713
petal.width	0.643817	-0.673417	0.975713	1.000000

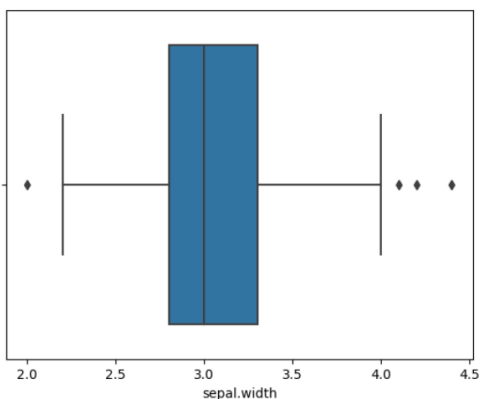
داده دورافتاده

یک داده دورافتاده (Outlier) یک مورد یا شیء داده‌ای است که به طور قابل توجهی از بقیه (معمولاً معمول) موارد دور افتاده است. این انحراف ممکن است ناشی از خطاهای اندازه‌گیری یا اجرا باشد. تجزیه و تحلیل برای کشف نقاط نادرست به عنوان استخراج نقاط نادرست شناخته می‌شود. روش‌های مختلفی برای شناسایی نقاط نادرست وجود دارد، و فرآیند حذف مشابه حذف یک مورد داده‌ای از دیتافریم pandas است. حال این عملیات را روی مجموعه داده آیریس را در نظر می‌گیریم و نمودار جعبه‌ای (boxplot) برای ستون sepal.width را رسم می‌کنیم.

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sns.boxplot(x='sepal.width', data=df)
```

```
<Axes: xlabel='sepal.width'>
```



در نمودار فوق، مقادیر بالای ۴ و پایین از ۲ به عنوان داده دورافتاده عمل می‌کنند.

-۵.۲

مجموعه داده Iris فراخوانی و داده‌ها با نسبت دلخواه به مجموعه‌های آموزش و ارزیابی تقسیم شدند (۸۰٪ آموزش، ۲۰٪ ارزیابی).

آموزش و ارزیابی مدل‌ها

Logistic Regression Metrics:
Accuracy: 0.967
Precision: 0.970
Recall: 0.967
F1 Score: 0.967
Confusion Matrix:
[[9 0 0]
[0 8 0]
[0 1 12]]

رگرسیون لجستیک (Logistic Regression):

- داده‌ها به وسیله استانداردسازی مقیاس داده‌ها آماده شدند.

- دقت آموزش ۹۶.۷٪

- مدل رگرسیون لجستیک با عملکردی بسیار خوب، ۹ نمونه از گونه اول را به درستی شناسایی کرده است.
- تنها یک نمونه از گونه دوم اشتباهاً به عنوان گونه سوم تشخیص داده شده است.
- تمام نمونه‌های گونه سوم به درستی تشخیص داده شده‌اند.

MLP Classifier Metrics:
Accuracy: 0.933
Precision: 0.947
Recall: 0.933
F1 Score: 0.934
Confusion Matrix:
[[9 0 0]
[0 8 0]
[0 2 11]]

شبکه‌های عصبی چندلایه (MLP Classifier):

- دقت آموزش ۹۳.۳٪

- MLP با عملکردی خوب، ۹ نمونه از گونه اول و ۱۱ نمونه از گونه سوم را به درستی تشخیص داده است.
- تنها دو نمونه از گونه دوم اشتباهاً به عنوان گونه سوم تشخیص داده شده‌اند.

RF SVM Metrics:
Accuracy: 0.967
Precision: 0.970
Recall: 0.967
F1 Score: 0.967
Confusion Matrix:
[[9 0 0]
[0 8 0]
[0 1 12]]

شبکه عصبی پایه شعاعی (RF SVM):

- داده‌ها به وسیله استانداردسازی آماده شدند.

- دقت آموزش ۹۶.۷٪

- RF SVM با عملکردی بسیار خوب، تمام نمونه‌ها را به درستی تشخیص داده است.
- تنها یک نمونه از گونه سوم اشتباهاً به عنوان گونه دوم تشخیص داده شده‌است.

تحلیل نتایج

هر سه مدل رگرسیون لجستیک، MLP، و RF SVM عملکردی بسیار خوب از خود نشان داده‌اند.

دقت کل مدل‌ها بسیار نزدیک به یکدیگر است، اما MLP نسبت به سایرین دقت کمی پایین‌تر دارد.