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 ۱- 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 1qSgWefXndozp6JyZ4TbRqebu9AvlT579
```

```
/content/Data
Downloading...
From: https://drive.google.com/uc?id=1qSgWefXndozp6JyZ4TbRqebu9AvlT579
To: /content/Data/Perceptron.csv
100% 17.3k/17.3k [00:00<00:00, 45.9MB/s]
```

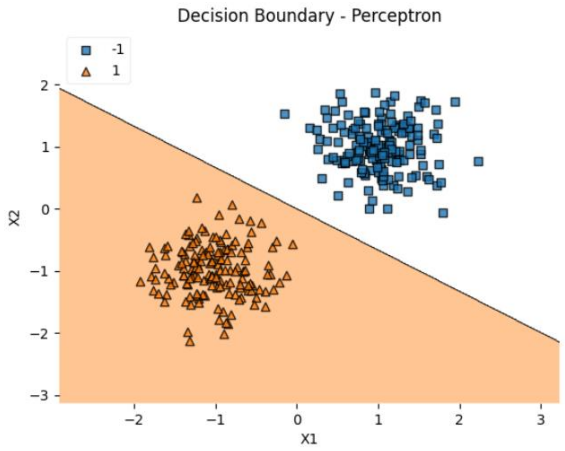
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions

# 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-

```python
# 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}")
```

```python
# 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 s
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.

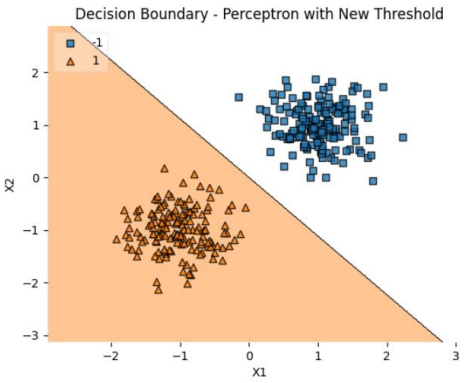
تغییر آستانه باید با دقت انجام شود و ممکن است نیاز به ارزیابی جامع تأثیرات آن بر کارایی مدل داشته باشد.

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].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 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 تاثیر 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 تاثیر 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.

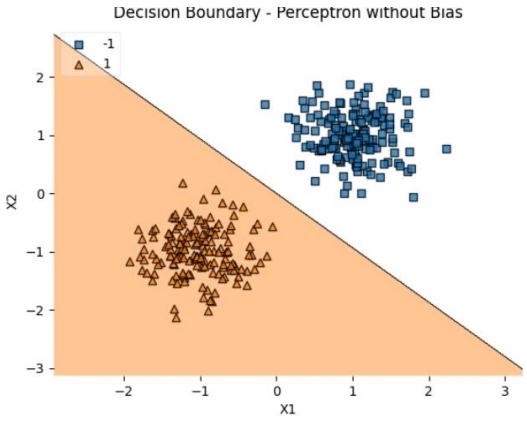
بنابراین، قبل از حذف بایاس از یک مدل، مهم است که تاثیرات آن را بر روی عملکرد مدل بررسی کنید و تصمیم‌گیری به دقت انجام شود. در بسیاری از موارد، استفاده از بایاس بهبود عملکرد و استقلال مدل از خصوصیات خاص داده‌های آموزشی را افزایش می‌دهد.

Perceptron without Bias

```python
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}")
```

```python
# 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.

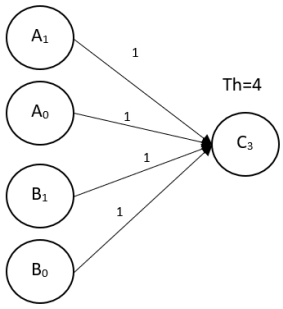
سپس وزن هر ورودی را با توجه به معادله‌ها بدست می‌آوریم.(خود ورودی: w=1 و متمم ورودی: w=-1) با استفاده از فرمول $net = \sum_{i=1}^{n} w_i x_i$ مجموع ورودی‌های وزن‌دار شده را بدست آورده و کوچکترین threshold ممکن را بدست می‌آوریم.

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

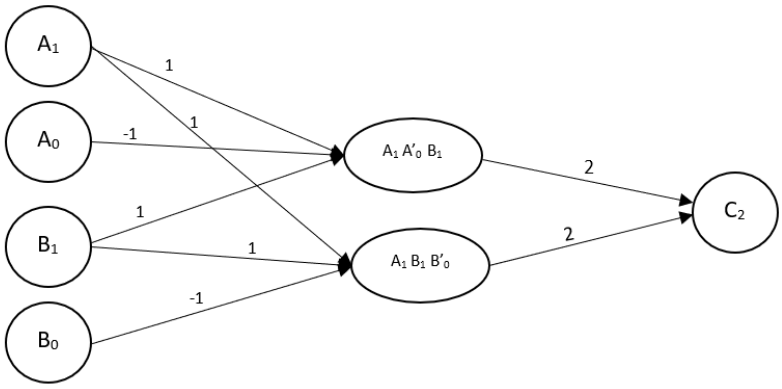
INPUT A		INPUT B		OUTPUT			
A_1	A_0	B_1	B_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1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

$C_3 = A_1A_0B_1B_0$



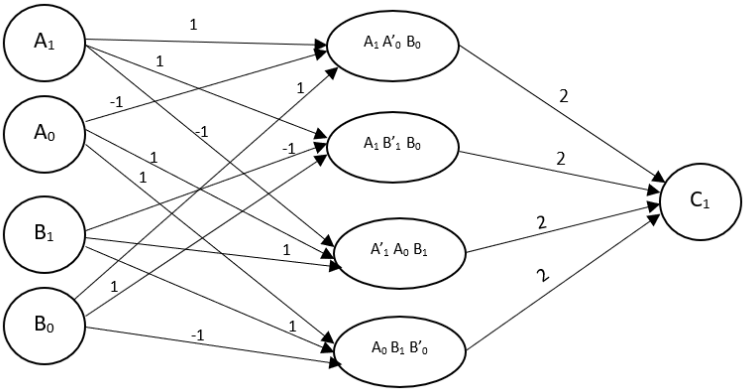
$C_2 = A_1A'_0B_1 + A_1B_1B'_0$

Th=2

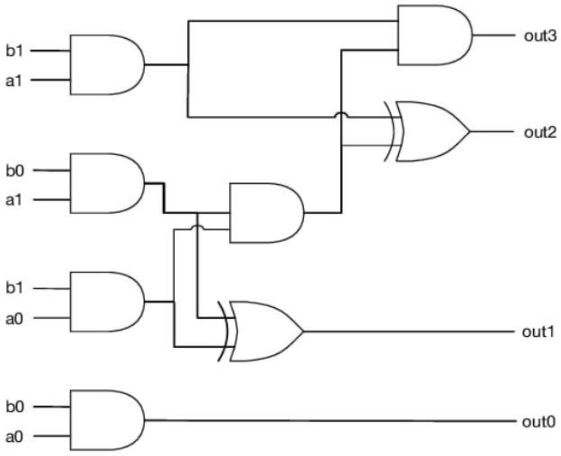


$C_1 = A'_1A_0B_1 + A_0B_1B'_0 + A_1A'_0B_0 + A_1B'_1B_0$

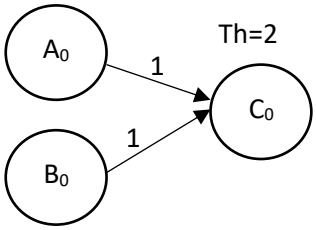
Th=2



Logic circuit



$C_0 = A_0B_0$



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 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 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 بازمی‌گرداند. این نمایش دودویی یک لیست است که هر عنصر آن به یک پیکسل متناظر است و مقدار آن یا ۱- (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 JPEGذخیره 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. مقداری 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.

۳.۲-

از شبکه Hamming استفاده می‌کنیم:

ابتدا توابع مورد نیازمان را تعریف می‌کنیم:

```python
def change(vector, a, b):
  vector = np.array(vector)
  matrix = vector.reshape((a, b))
  return matrix
```

بردار vector را به ماتریسی با ابعاد a و b تبدیل می‌کند.

ماتریس را در بردار ضرب کرده و حاصلضربشان را با مقدار آستانه جمع می‌کند.

```
def product(matrix, vector, T):
  result_vector = []
  for i in range(len(matrix)):
    sum = 0
    for j in range(len(vector)):
      sum = sum + matrix[i][j] * vector[j]
    result_vector.append((sum + T))
  return result_vector
```

مقادیر بردار را با مقدار آستانه مقایسه می‌کند.

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

درایه‌های بردار یه جز درایه jام را با هم جمع می‌کند.

```
def sum(vector, j):
  total_sum = 0
  for i in range(0, len(vector)):
    if i != j:
      total_sum = total_sum + vector[i]
  return total_sum
```

نرم بردارها را محاسبه می‌کند.

```
def norm(vector, p):
  difference = []
  for i in range(len(vector)):
    difference.append(vector[i] - p[i])
  sum = 0
  for element in difference:
    sum += element * element
  return sqrt(sum)
```

حال با استفاده از تابع convertImageToBinary تصاویر را به حالت باینری تبدیل می‌کنیم. سپس یک تصویر نویزی را انتخاب می‌کنیم تا شبگه همینگ را تست کنیم که تشخیص درستی دارد یا نه. (به ترتیب همه تصاویر نویزی را امتحان می‌کنیم.)

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

x = []
for i in path:
  x.append(convertImageToBinary(i))

image_path = "/content/noisy3.jpg"
y = convertImageToBinary(image_path)

print(os.path.basename(image_path))
```

متغیرها، ماتریس وزن و ماتری سیناپس شبکه عصبی را تعریف می‌کنیم.

```
k = len(x)
a = 96
b = 96
q = change(y, a, b)
plt.matshow(q)
m = len(x[0])
T = m / 2
Emax = 0.000001
U = 1 / Emax

w = [[(x[i][j]) / 2 for j in range(m)] for i in range(k)]
e = round(1 / len(x), 1)
E = [[0 for j in range(k)] for i in range(k)]


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)]
p = action(s[0], U, Emax)
y = [p]
i = 0
j = []
p = [0 for j in range(len(s[0]))]
```

شبکه عصبی را آموزش می‌دهیم. آموزش تا زمانی که نرم بردارهای y و p بزرگتر از Emax باشد ادامه می‌یابد.

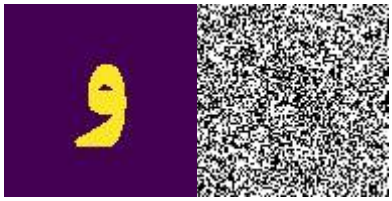
```
while norm(y[i], p) >= Emax:
  s.append([0 for j in range(len(s[0]))])
  for j in range(len(s[0])):
    s[i + 1][j] = y[i][j] - e*sum(y[i], j)
  y.append((action(s[i + 1], U, Emax)))
  i += 1
  p = y[i - 1]

result_index = y[len(y) - 1].index(max(y[len(y) - 1])) + 1
q = change(x[result_index - 1], a, b)

from matplotlib import pyplot as plt
from matplotlib import image as img

img.imsave('output.jpg', q)
output_img = Image.open('output.jpg')
output_img = output_img.transpose(Image.FLIP_TOP_BOTTOM)
output_img = output_img.transpose(Image.ROTATE_270)
output_img.save('output.jpg')

plt.show()
image = img.imread('output.jpg')
print('\n' + '\n')
plt.imshow(image)
plt.show()
```

تصاویر خروجی به ازای noise_factor = 1000:

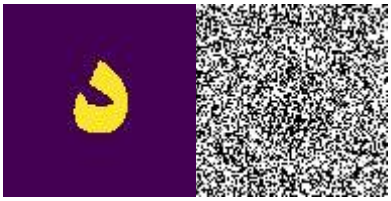
تشخیص درست

تشخیص درست

تشخیص درست



تشخیص درست



تشخیص درست

حال مشاهده می‌کنیم که با افزایش مقدار noise_factor شبکه دچار اختلال می‌شود. مثلا به ازای noise_factor=5000 مشاهده می‌شود که شبکه به‌جای حرف ج حرف و را تشخیص داده‌است.

-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 می‌کندبا این تفاوت که مقادیر رندوم فقط زمانی اضافه می‌شوند که پیکسل مدنظر مشکی بوده باشد.(=۱)

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

```python
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
enerateMissedPointImages()
```

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

داده دچار اختلال | داده اولیه | داده دچار اختلال | داده اولیه




داده دچار اختلال | داده اولیه | داده دچار اختلال | داده اولیه




داده دچار اختلال | داده اولیه


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 1vl0LDTXqy28Jhy22A70e_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.

```python
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 NANای ندارد و اگر ۱ برگرداند یعنی دارای داده NAN است.

همانطور که مشاهده می‌شود در این مجموعه داده هیچ داده NAN ای نداریم.

```python
# Count NaN values in each column
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
```

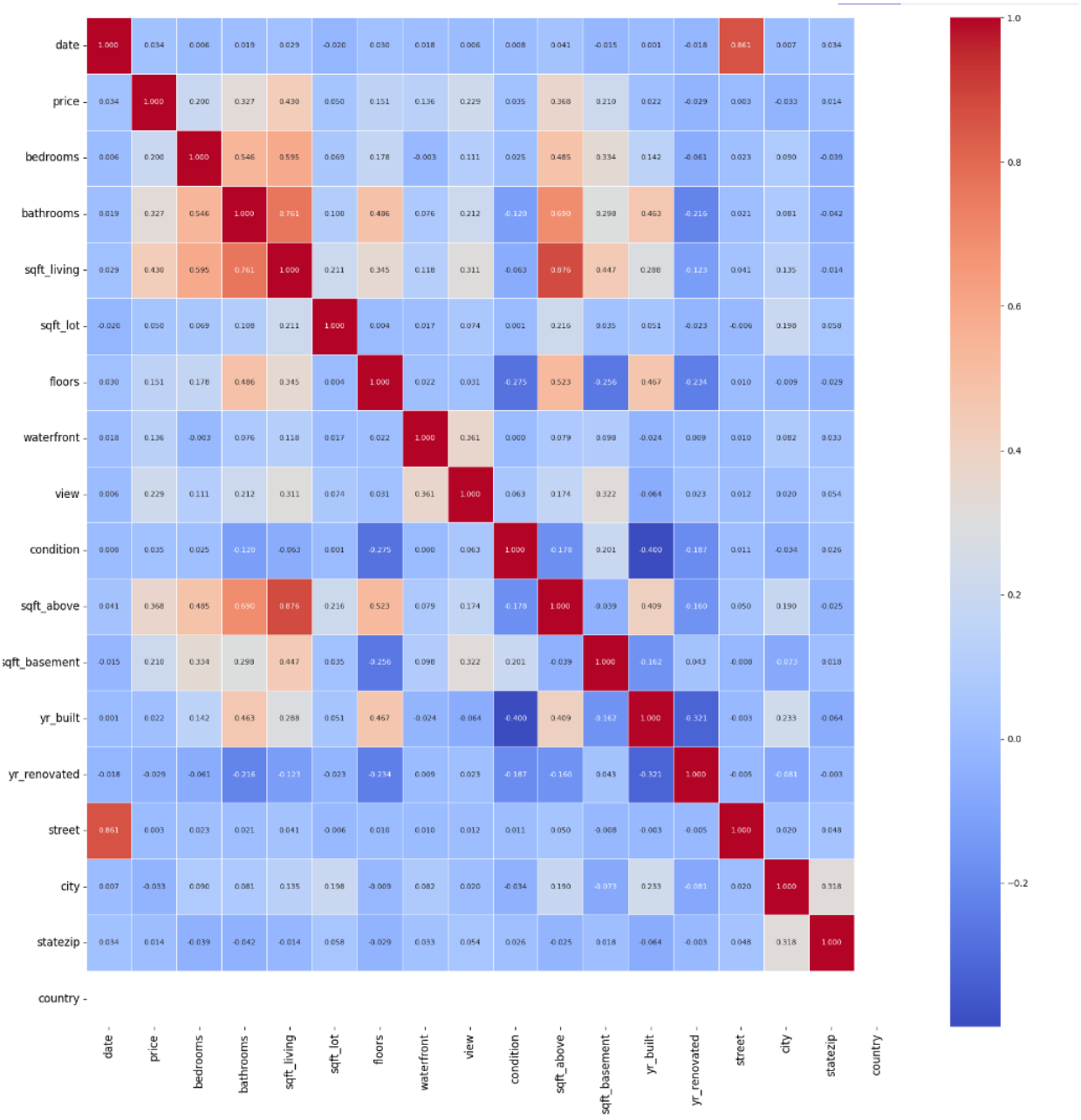
۴.۲-

برای رسم ماتریس همبستگی نیاز است ابتدا ستون‌هایی که داده‌های غیرعددی 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.

```python
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.

```python
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.

```python
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')
```

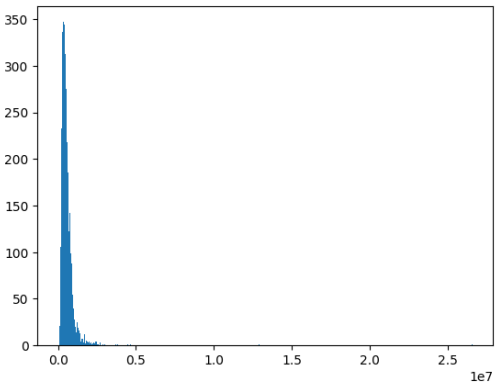
با توجه به نمودار زیر بیشترین همبستگی ستون قیمت به جز خودش(=۱) با متغیر sqft_living(=۰.۴۳۰) است.

۴.۳-

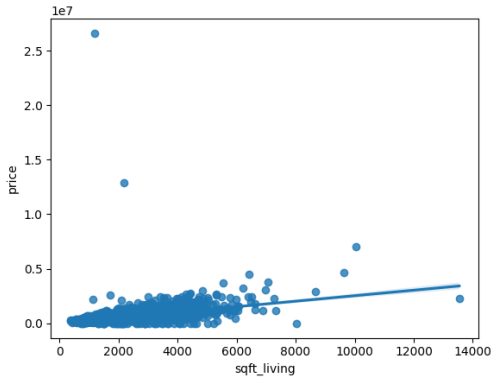
با استفاده از تابع hist توزیع قیمت خانه‌ها را نمایش می‌دهیم. برای رسم نمودار قیمت برحسب ویژگی sqft_living نیز از دستور sns.regplot استفاده می‌کنیم.

```python
import matplotlib.pyplot as plt
plt.hist(df['price'], bins = 'auto')
plt.show()

import seaborn as sns
sns.regplot(x = df['sqft_living'], y = df['price'])
```

نمودار توزیع آماری قیمت



نمودار قیمت برحسب 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.

```
df[['year', 'month', 'day']] = df['date'].str.split('-', expand = True)
df = df.drop('date', axis = 1)
df = df.drop('day', axis = 1)
print(df)
```

```
             price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  \
0     3.130000e+05       3.0       1.50         1340      7912     1.5
1     2.384000e+06       5.0       2.50         3650      9050     2.0
2     3.420000e+05       3.0       2.00         1930     11947     1.0
3     4.200000e+05       3.0       2.25         2000      8030     1.0
4     5.500000e+05       4.0       2.50         1940     10500     1.0
...            ...       ...        ...          ...       ...     ...
4595  3.081667e+05       3.0       1.75         1510      6360     1.0
4596  5.343333e+05       3.0       2.50         1460      7573     2.0
4597  4.169042e+05       3.0       2.50         3010      7014     2.0
4598  2.034000e+05       4.0       2.00         2090      6630     1.0
4599  2.206000e+05       3.0       2.50         1490      8102     2.0

      waterfront  view  condition  sqft_above  sqft_basement  yr_built  \
0              0     0          3        1340              0      1955
1              0     4          5        3370            280      1921
2              0     0          4        1930              0      1966
3              0     0          4        1000           1000      1963
4              0     0          4        1140            800      1976
...          ...   ...        ...         ...            ...       ...
4595           0     0          4        1510              0      1954
4596           0     0          3        1460              0      1983
4597           0     0          3        3010              0      2009
4598           0     0          3        1070           1020      1974
4599           0     0          4        1490              0      1990

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

     year month
0     2014    05
1     2014    05
2     2014    05
3     2014    05
4     2014    05
...    ...   ...
4595  2014    07
4596  2014    07
4597  2014    07
```

-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.

```python
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.

```python
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-

آموزش

```python
def train(model, optimizer, criterion, X_train, y_train, X_val, y_val, num_epoch = 1000):
  train_loss_list = []
  validation_loss_list = []
  train_r2score_list = []
  r2score_list = []

  for i in range(num_epoch):
    model.train()
    optimizer.zero_grad()
    yhat_train = model(X_train)
    train_loss = criterion(yhat_train.squeeze(), y_train)
    train_loss.backward()
    optimizer.step()
    train_loss_list.append(train_loss.item())
    train_r2score = r2_score(y_train, yhat_train.squeeze().detach().numpy())
    train_r2score_list.append(train_r2score)
```

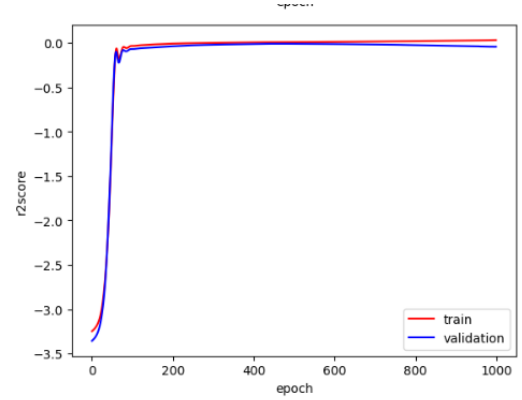
اعتبارسنجی

```python
    model.eval()
    with torch.no_grad():
      yhat_val = model(X_val)
      val_loss = criterion(yhat_val.squeeze(), y_val)
      validation_loss_list.append(val_loss.item())
      r2score = r2_score(y_val, yhat_val.squeeze().detach().numpy())
      r2score_list.append(r2score)
```

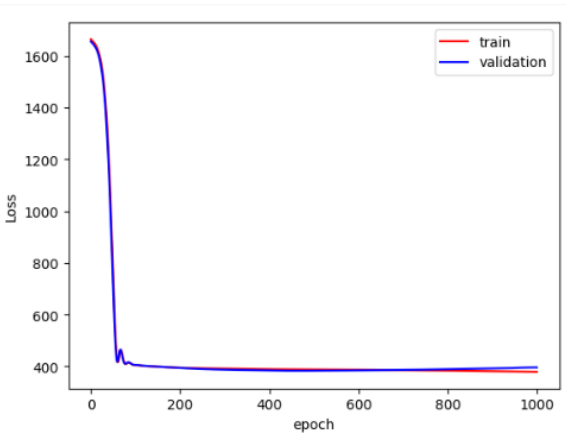
ایجاد یک نمونه کلاس MLP و آموزش مدل با دیتاست سوال

```python
mlp = MLP(12, 50, 100, 150, 1)
optimizer = optim.Adam(mlp.parameters(), lr = 0.001)
criterian = nn.MSELoss()
mlp_ = train(mlp, optimizer, criterian, X_train, y_train, X_val, y_val, num_epoch = 1000)

mlp.eval()
with torch.no_grad():
  yhat = mlp(X_test)
  test_loss = criterian(yhat.squeeze(), y_test)
  test_r2score = r2_score(y_test, yhat.squeeze().detach().numpy())

print("Loss: " + str(test_loss) + "R2 Score: " + str(test_r2score))
```

نتیجه



Loss: tensor(386.7848)R2 Score: -0.012291764611467304

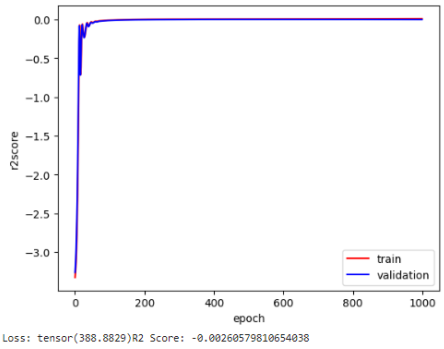


-۴.۷

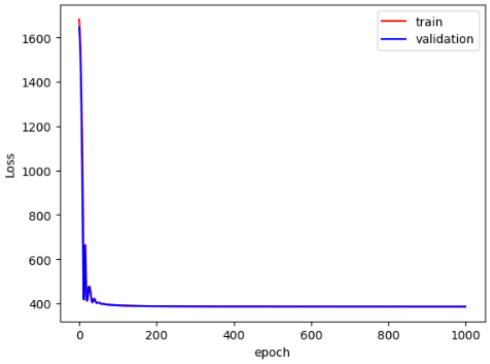
تغییر تابع بهینه‌ساز به Adamx

```
mlp2 = MLP(12, 50, 100, 150, 1)
optimizer = optim.Adamax(mlp2.parameters(), lr = 0.01)
criterian = nn.MSELoss()
mlp_ = train(mlp2, optimizer, criterian, X_train, y_train, X_val, y_val, num_epoch = 1000)

mlp2.eval()
with torch.no_grad():
  yhat = mlp2(X_test)
  test_loss = criterian(yhat.squeeze(), y_test)
  test_r2score = r2_score(y_test, yhat.squeeze().detach().numpy())

print("Loss: " + str(test_loss) + "R2 Score: " + str(test_r2score))
```

نتیجه



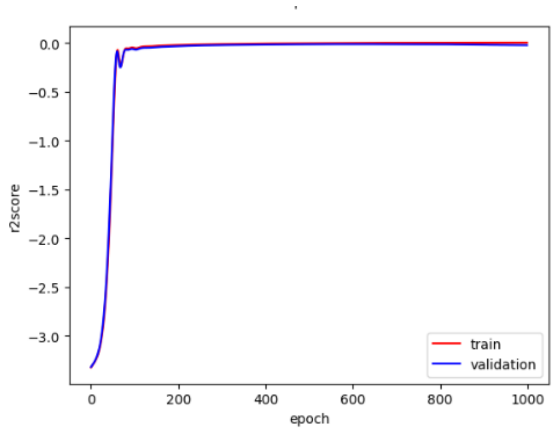
Loss: tensor(388.8829)R2 Score: -0.00260579810654038



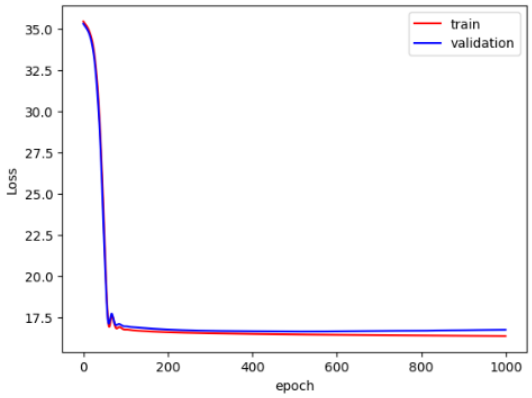
تغییر تابع اتلاف به HuberLoss

```
mlp3 = MLP(12, 50, 100, 150, 1)
optimizer = optim.Adam(mlp3.parameters(), lr = 0.001)
criterian = nn.HuberLoss()
mlp_ = train(mlp3, optimizer, criterian, X_train, y_train, X_val, y_val, num_epoch = 1000)

mlp3.eval()
with torch.no_grad():
  yhat = mlp3(X_test)
  test_loss = criterian(yhat.squeeze(), y_test)
  test_r2score = r2_score(y_test, yhat.squeeze().detach().numpy())

print("Loss: " + str(test_loss) + "R2 Score: " + str(test_r2score))
```

نتیجه



Loss: tensor(16.5387)R2 Score: -0.015055563539662975



۴.۸-

با استفاده از یک حلقه for پنج داده تصادفی در بازه ۰ تا تعداد داده‌های ارزیابی تولید می‌کنیم.

```
import tensorflow as tf

mlp = MLP(12, 50, 100, 150, 1)
optimizer = optim.Adamax(mlp.parameters(), lr = 0.01)
criterian = nn.MSELoss()
mlp_ = train(mlp, optimizer, criterian, X_train, y_train, X_val, y_val, num_epoch = 1000)

mlp.eval()
with torch.no_grad():
  yhat = mlp(X_test)
  test_loss = criterian(yhat.squeeze(), y_test)
  test_r2score = r2_score(y_test, yhat.squeeze().detach().numpy())

print("Loss: " + str(test_loss) + "R2 Score: " + str(test_r2score))

y_test.reshape([1, 920])
yhat.reshape([1, 920])

for i in range(0, 5):
  n = np.random.randint(0, int(len(y)* 0.2))
  print("y_test: " + str(y_test[n]))
  print("predicted y: " + str(yhat[n]))
  print('\n')
```

```
y_test: tensor(21.)
predicted y: tensor([34.3551])


y_test: tensor(10.)
predicted y: tensor([36.3831])


y_test: tensor(18.)
predicted y: tensor([37.5239])


y_test: tensor(54.)
predicted y: tensor([39.7128])
```

جهت بهبود عملکرد شبکه عصبی می‌توان تعداد epoch، لایه‌های پنهان و تعداد نورون‌های آن را افزایش و یا مقدار learning rate را کاهش داد. همچنین می‌توان از توابع اتلاف و بهینه‌ساز متفاوتی استفاده کرد.

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 نمونه ها = ۱۵۰ تا

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

تعداد کلاس ها = 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 داده ها را بر می زنیم و سپس با نسبت ۸۰ به ۲۰ درصد به دو قسمت آموزش و آزمون تقسیم می کنیم.

```
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 1bDuT0miTwHPS087HP8jr3zSFP1wy63O7

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 ازداده‌ها 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.

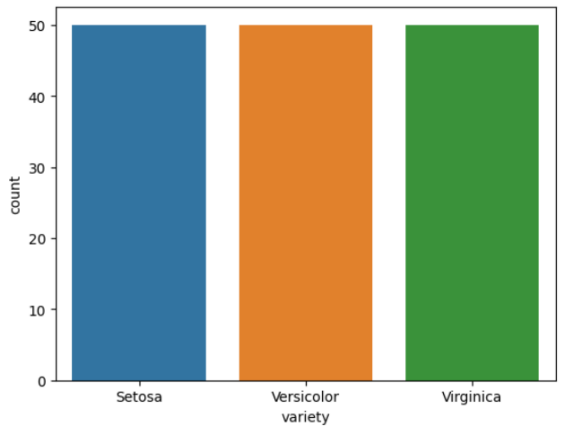
```
df.value_counts("variety")
```

```
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.

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


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

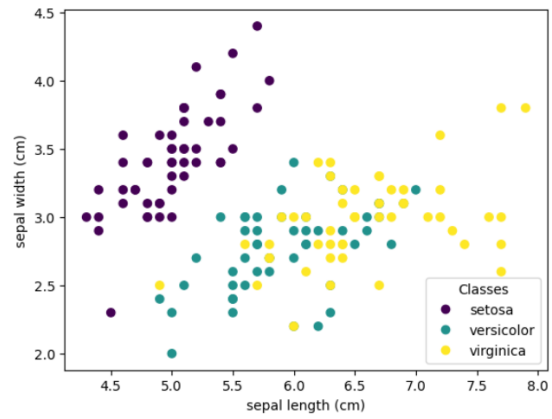


رابطه بین متغیرها

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

```python
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:

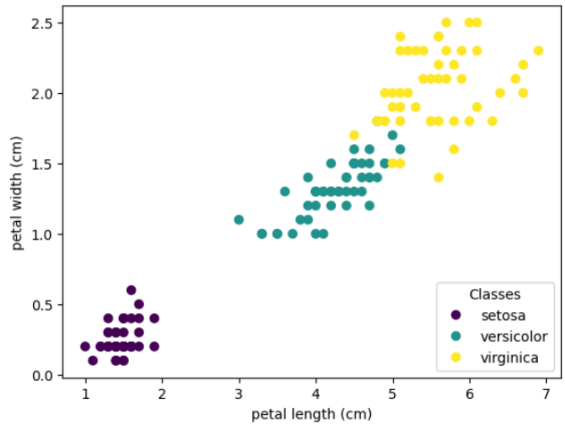
- گونه Setosa دارای طول کاسبرگ کوچکتر اما عرض کاسبرگ بزرگتری است.

- 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.

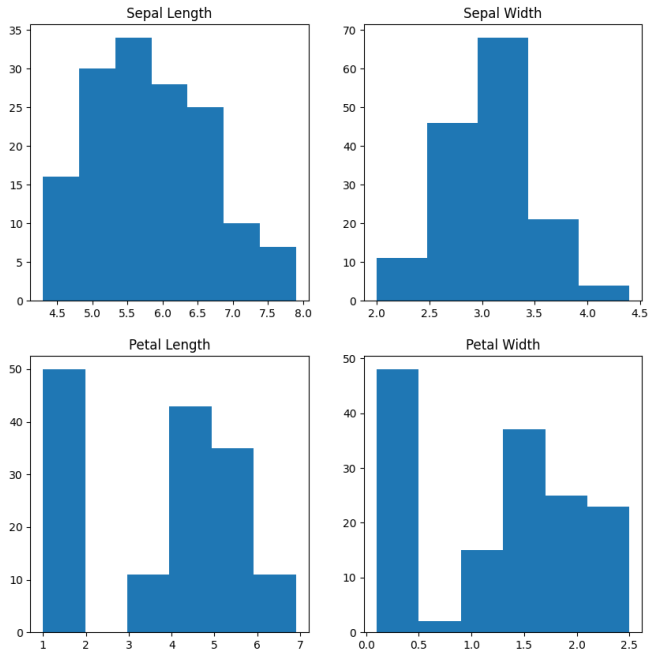
- گونه Virginica دارای بزرگترین طول و عرض گلبرگ است.

مشاهده می‌کنیم که جداسازی کلاس ستوسا از دو کلاس دیگر است. جداسازی کلاس ها با طول و عرض کاسبرگ دشوارتر است و داده ها همپوشانی بیشتری دارند. (به خصوص در کلاس های ورسیکالر و ویرجینیکا) اما با ویژگی های طول و عرض گلبرگ جداسازی آسان تر 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')
```

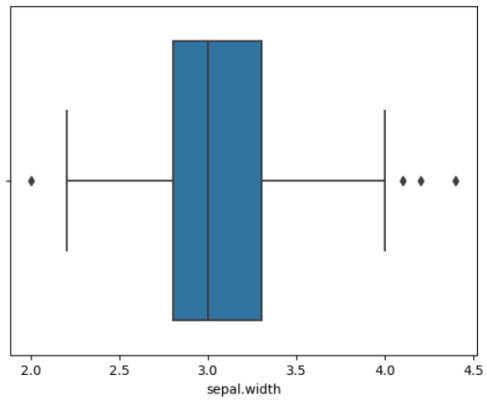
	sepal.length	sepal.width	petal.length	petal.width
sepal.length	1.000000	-0.999226	0.795795	0.643817
sepal.width	-0.999226	1.000000	-0.818999	-0.673417
petal.length	0.795795	-0.818999	1.000000	0.975713
petal.width	0.643817	-0.673417	0.975713	1.000000

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).

شاخص‌های ارزیابی را بصورت توابع جداگانه تعریف می‌کنیم.

```
import pandas as ps
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split

df = datasets.load_iris()
X = df['data']
y = df['target']
y = y.reshape((-1, 1))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

شاخص accuracy

```
def accuracy(yhat, y):
  n = 0
  for i in range(0, len(y)):
    if (y[i] == yhat[i]):
      n = n + 1
  a = n/len(y)
  return a
```

شاخص precision

```
def precision(yhat, y):
  TP_0 = 0
  FP_0 = 0
  TP_1 = 0
  FP_1 = 0
  TP_2 = 0
  FP_2 = 0
  for i in range(len(y)):
    if (y[i] == 0):
      if (yhat[i] == 0):
        TP_0 = TP_0 + 1
    if (yhat[i] == 0):
      if (y[i] != 0):
        FP_0 = FP_0 + 1
    if (y[i] == 1):
      if (yhat[i] == 1):
        TP_1 = TP_1 + 1
    if (yhat[i] == 1):
      if (y[i] != 1):
        FP_1 = FP_1 + 1
    if (y[i] == 2):
      if (yhat[i] == 2):
        TP_2 = TP_2 + 1
    if (yhat[i] == 2):
      if (y[i] != 2):
        FP_2 = FP_2 + 1
  a_0 = TP_0 / (TP_0 + FP_0)
  a_1 = TP_1 / (TP_1 + FP_1)
  a_2 = TP_2 / (TP_2 + FP_2)
  a = (a_0 + a_1 + a_2) / 3
  return a
```

شاخص recall

```
def recall(yhat, y):
  TP_0 = 0
  FN_0 = 0
  TP_1 = 0
  FN_1 = 0
  TP_2 = 0
  FN_2 = 0
  for i in range(len(y)):
    if (y[i] == 0):
      if (yhat[i] == 0):
        TP_0 = TP_0 + 1
      else:
        FN_0 = FN_0 + 1
    if (y[i] == 1):
      if (yhat[i] == 1):
        TP_1 = TP_1 + 1
      else:
        FN_1 = FN_1 + 1
    if (y[i] == 2):
      if (yhat[i] == 2):
        TP_2 = TP_2 + 1
      else:
        FN_2 = FN_2 + 1
  a_0 = TP_0 / (TP_0 + FN_0)
  a_1 = TP_1 / (TP_1 + FN_1)
  a_2 = TP_2 / (TP_2 + FN_2)
  a = (a_0 + a_1 + a_2) / 3
  return a
```

شاخص F1score

```
def f1score(p, r):
  a = (2*p*r) / (p + r)
  return a
```

ماتریس درهم ریختگی

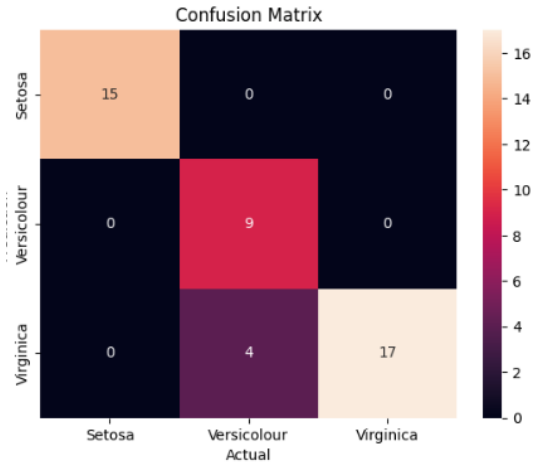
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
def confusion_m(y, yhat):
  cm = confusion_matrix(y, yhat)
  sns.heatmap(cm, annot = True, xticklabels = ['Setosa', 'Versicolour', 'Virginica'], yticklabels = ['Setosa', 'Versicolour', 'Virginica'])
  plt.ylabel('Prediction')
  plt.xlabel('Actual')
  plt.title('Confusion Matrix')
  plt.show()
```

آموزش و ارزیابی مدل‌ها

مدل‌ها را با استفاده از کتابخانه آماده پایتون می‌سازیم.

رگرسیون لجستیک (Logistic Regression):

```
#>LogisticRegression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state = 13)
model.fit(X_train, y_train)
yhat = model.predict(X_test)
a = accuracy(yhat, y_test)
p = precision(yhat, y_test)
r = recall(yhat, y_test)
F = F1score(p, r)
cm = confusion_m(y_test, yhat)
print('\n' + '\n')
print("The 'accuracy' of 'LogisticRegression': " + str(a))
print("The 'precision' of 'LogisticRegression': " + str(p))
print("The 'recall' of 'LogisticRegression': " + str(r))
print("The 'F1score' of 'LogisticRegression': " + str(F))
```

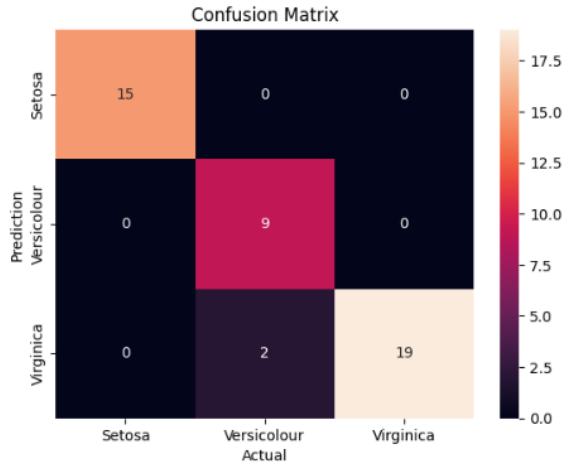



```
The 'accuracy' of 'LogisticRegression': 0.9111111111111111
The 'precision' of 'LogisticRegression': 0.8974358974358975
The 'recall' of 'LogisticRegression': 0.9365079365079364
The 'F1score' of 'LogisticRegression': 0.9165557035064359
```

شاخص‌ها دقت حدود ۹۰٪ دارند. ۴ نمونه اشتباه پیش‌بینی شده‌است.

شبکه‌های عصبی چندلایه (MLP Classifier):

```
#MLP
from sklearn.neural_network import MLPClassifier
model = MLPClassifier(hidden_layer_sizes = (50, 20, 5), max_iter = 500, random_state = 13)
model.fit(X_train, y_train)
yhat = model.predict(X_test)
a = accuracy(yhat, y_test)
p = precision(yhat, y_test)
r = recall(yhat, y_test)
F = F1score(p, r)
cm = confusion_m(y_test, yhat)
print('\n' + '\n')
print("The 'accuracy' of 'MLP': " + str(a))
print("The 'precision' of 'MLP': " + str(p))
print("The 'recall' of 'MLP': " + str(r))
print("The 'F1score' of 'MLP': " + str(F))
```

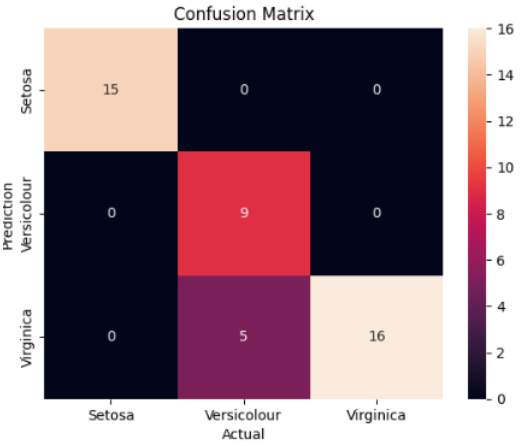



```
The 'accuracy' of 'MLP': 0.9555555555555556
The 'precision' of 'MLP': 0.9393939393939394
The 'recall' of 'MLP': 0.9682539682539683
The 'F1score' of 'MLP': 0.9536056480080685
```

شاخص‌ها دقت حدود ۹۵٪ دارند. ۲ نمونه اشتباه پیش‌بینی شده‌است.

شبکه عصبی پایه شعاعی (RBF SVM):

```
#RBF
from sklearn.svm import SVC
model = SVC(kernel = 'rbf')
model.fit(X_train, y_train)
yhat = model.predict(X_test)
a = accuracy(yhat, y_test)
p = precision(yhat, y_test)
r = recall(yhat, y_test)
F = F1score(p, r)
cm = confusion_m(y_test, yhat)
print('\n' + '\n')
print("The 'accuracy' of 'RBF': " + str(a))
print("The 'precision' of 'RBF': " + str(p))
print("The 'recall' of 'RBF': " + str(r))
```




```
The 'accuracy' of 'RBF': 0.8888888888888888
The 'precision' of 'RBF': 0.8809523809523809
The 'recall' of 'RBF': 0.9206349206349206
The 'F1score' of 'RBF': 0.9003566184182925
```

شاخص‌ها دقت حدود ۹۰٪ دارند. ۵ نمونه اشتباه پیش‌بینی شده‌است.

تحلیل نتایج

دقت کل مدل‌ها نزدیک به یکدیگر است، اما MLP نسبت به سایرین دقت بالاتری دارد.