

به نام خداوند بخشنده مهربان

درس مبانی سیستم‌های هوشمند

گزارش مینی پروژه شماره یک – بخش اول

معصومه شریف تبار عزیزی – ۹۹۲۷۶۱۳

۱.

۱.

با استفاده از دستور `make_classification` یک دیتاست با ۱۰۰۰ نمونه و ۲ ویژگی و ۲ کلاس تولید می‌کنیم. از `random state` برای این استفاده می‌کنیم که داده‌های تولید شده در هر بار اجرا تغییر نکند. `C=y` برای نمایش کلاس‌ها `n_clusters_per_class` نمایش تعداد انواع توزیع برای هر کلاس `class_sep` نمایش میزان تفکیک داده‌ها از هم (هر چه کوچک‌تر باشد داده‌ها بیشتر در هم قاطی می‌شوند و جاسازی آن‌ها دشوارتر خواهد بود). وقتی کد را اجرا می‌کنیم برای `x.shape` داریم (۲ و ۱۰۰۰) یعنی ۱۰۰۰ نمونه و ۲ ویژگی داریم، برای `y.shape` نیز داریم (۱۰۰۰) یعنی ۱۰۰۰ تا خروجی یا همان تارگت داریم.

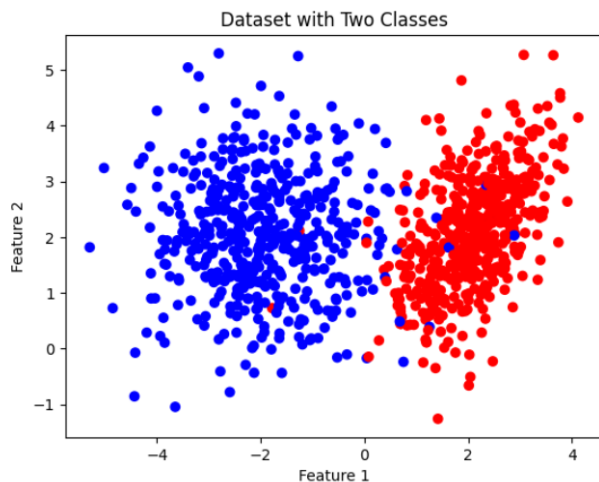
```
# Generate dataset => 1000 samples, 2 classes, 2 features
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000,
                          n_features=2,
                          n_redundant=0,
                          n_classes=2,
                          n_clusters_per_class=1,
                          class_sep=2,
                          random_state=13)

print(X.shape, y.shape)

# plotting dataset
colors = np.array(['blue', 'red'])
plt.scatter(X[:, 0], X[:, 1], c=colors[y])
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Dataset with Two Classes')
plt.show()
```

(1000, 2) (1000,)



۲.

با استفاده از دو classifier آماده پایتون (logistic و SGD) ۲ کلاس را از هم تفکیک میکنیم.
دقت فرآیند آموزش و ارزیابی را با دستور score تعیین میکنیم.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score

X, y = make_classification(n_samples=1000,
                           n_features=2,
                           n_redundant=0,
                           n_classes=2,
                           n_clusters_per_class=1,
                           class_sep=2,
                           random_state=13)

# split train test datas
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=13)

##LogisticRegression
model = LogisticRegression(random_state=13)
model.fit(x_train, y_train) #train
model_predict = model.predict(x_test) #test...it gives us y_hat_test
log_accuracy = accuracy_score(y_test, model_predict) # accuracy
print("Logistic Regression Accuracy:", log_accuracy)

##SGDClassifier()
model1 = SGDClassifier(loss='log_loss', random_state=13)
model1.fit(x_train, y_train) #train
model1_predict = model1.predict(x_test) #test
SGD_accuracy = accuracy_score(y_test, model1_predict) #accuracy
print("SGD Classifier Accuracy:", SGD_accuracy)
```

Logistic Regression Accuracy: 0.99
SGD Classifier Accuracy: 0.975

۳.

مرز و نواحی تصمیم گیری مدل آموزش دیده را با دستور mlxtend.plotting تعیین میکنیم.

```

## for LogisticRegression

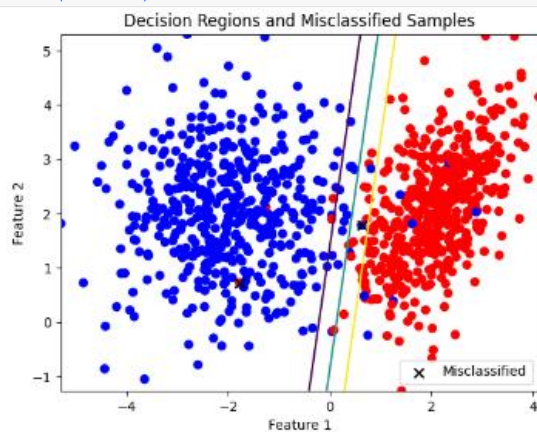
# Determine the minimum and maximum values for each feature
x1_min, x2_min = X.min(0)
x1_max, x2_max = X.max(0)
n = 500
x1r = np.linspace(x1_min, x1_max, n)
x2r = np.linspace(x2_min, x2_max, n)
x1m, x2m = np.meshgrid(x1r, x2r)
Xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
# Calculate decision values for specific points in the space
ym = model.decision_function(Xm)

# assign colors to classes (0 for blue, 1 for red)
colors = np.array(['blue', 'red'])
plt.scatter(X[:, 0], X[:, 1], c=colors[y])
# Plot decision boundary
plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels=[-1,0,1])
# Display misclassified samples with a different color
misclassified = (y_test != model.predict)
plt.scatter(x_test[misclassified, 0], x_test[misclassified, 1], c='black', marker='x', s=50, label='Misclassified')

plt.title('Decision Regions and Misclassified Samples')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()

## direct command
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X, y, clf=model)
plt.title('Decision Regions')
plt.xlabel('Feature 1')

```



۴.

با افزایش تعداد کلاس ها، ویژگی ها، $n_cluster_pet_class$ می توان مدل را پیچیده تر کرد. همچنین با کاهش مقدار $class_sep$ درهم تنیدگی داده ها بیشتر شده و جداسازی آن ها دشوارتر خواهد شد. در این بخش تلاش بر این بود با کاهش مقدار $class_sep$ پیچیدگی فرآیند آموزش بیشتر شود. مشاهده می شود در این حالت دقت نیز کاهش می یابد.

در بخش بعدی تعداد کلاس ها را افزایش دادم و مراحل قسمت های ۲ و ۳ را تکرار کردم.

```
# subtract class_sep

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, SGDClassifier

# Generate Data
X, y = make_classification(
    n_samples=1000,
    n_features=2,
    n_classes=2,
    n_redundant=0,
    n_clusters_per_class=1,
    class_sep=0.8,
    random_state=13
)

print(X.shape, y.shape)
colors = np.array(['blue', 'red', 'green'])
plt.scatter(X[:, 0], X[:, 1], c=colors[y])
plt.title('Dataset with Three Classes')
plt.show()

## Part 2 ##
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=13)

# check the shape of data
x_train.shape, x_test.shape, y_train.shape, y_test.shape

##LogisticRegression
model = LogisticRegression(random_state=13)
model.fit(x_train, y_train) #train
model_predict = model.predict(x_test) #test...it gives us y_hat_test
log_accuracy = accuracy_score(y_test, model_predict) #accuracy
print("Logistic Regression Accuracy:", log_accuracy)

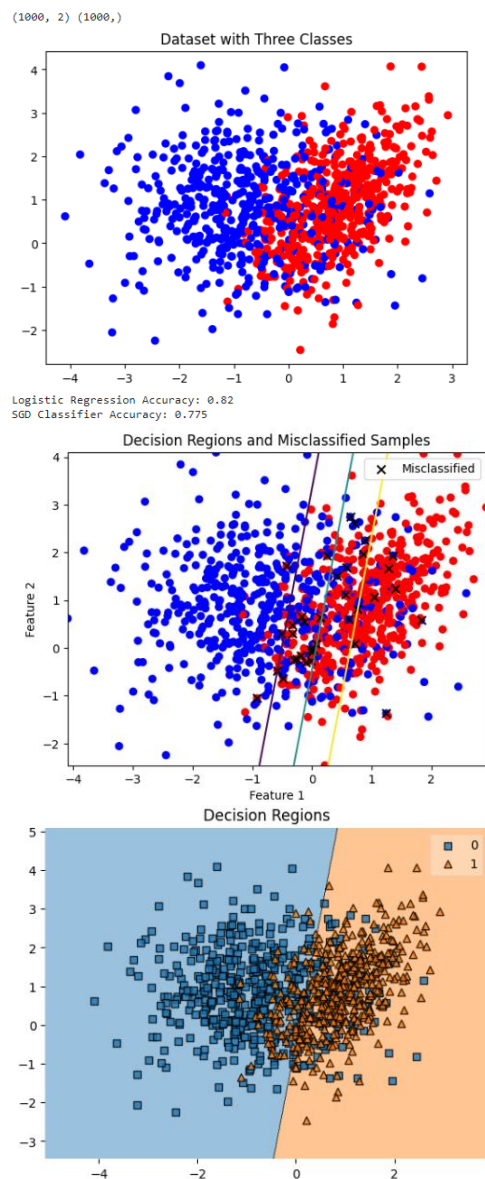
##SGDClassifier
model1 = SGDClassifier(loss='log_loss', random_state=13)
model1.fit(x_train, y_train) #train
model1_predict = model1.predict(x_test) #test
SGD_accuracy = accuracy_score(y_test, model1_predict) #accuracy

## Part 3 ##
# Determine the minimum and maximum values for each feature
x1_min, x2_min = X.min(0)
x1_max, x2_max = X.max(0)
n = 500
x1r = np.linspace(x1_min, x1_max, n)
x2r = np.linspace(x2_min, x2_max, n)
x1m, x2m = np.meshgrid(x1r, x2r)
Xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
# Calculate decision values for specific points in the space
ym = model.decision_function(Xm)

# assign colors to classes (0 for blue, 1 for red)
colors = np.array(['blue', 'red'])
plt.scatter(X[:, 0], X[:, 1], c=colors[y])
# Plot decision boundary
plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels=[-1,0,1])
# Display misclassified samples with a different color
misclassified = (y_test != model_predict)
plt.scatter(x_test[misclassified, 0], x_test[misclassified, 1], c='black', marker='x', s=50, label='Misclassified')

plt.title('Decision Regions and Misclassified Samples')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()

###
from mlxtend.plotting import plot_decision_regions
plt.title('Decision Regions')
plot_decision_regions(X, y, clf=model)
```



۵.

در این بخش مشاهده می شود که با افزایش تعداد کلاس ها دقت ارزیابی کاهش می یابد.

```

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, SGDClassifier

X, y = make_classification(
    n_samples=1000,
    n_features=2,
    n_classes=3,
    n_redundant=0,
    n_clusters_per_class=1,
    class_sep=2,
    random_state=13
)

print(X.shape, y.shape)
colors = np.array(['blue', 'red', 'green'])
plt.scatter(X[:, 0], X[:, 1], c=colors[y])
plt.title('Dataset with Three Classes')
plt.show()

## Part 2 ##
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=13)

# check the shape of data
x_train.shape, x_test.shape, y_train.shape, y_test.shape

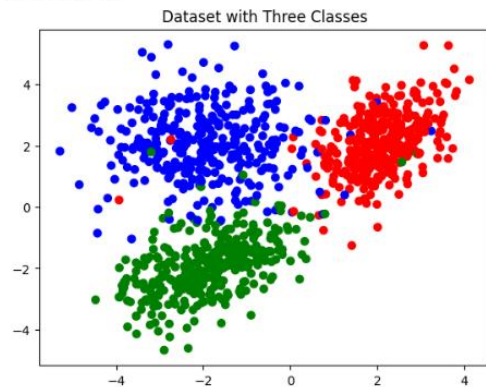
## Logistic Regression
model = LogisticRegression(random_state=13)
model.fit(x_train, y_train) #train
model_predict = model.predict(x_test) #test...it gives us y_hat_test
log_accuracy = accuracy_score(y_test, model_predict) #accuracy
print("Logistic Regression Accuracy:", log_accuracy)

## Part 3 ##

from mlxtend.plotting import plot_decision_regions
plt.title('Decision Regions')
plot_decision_regions(X, y, clf=model)

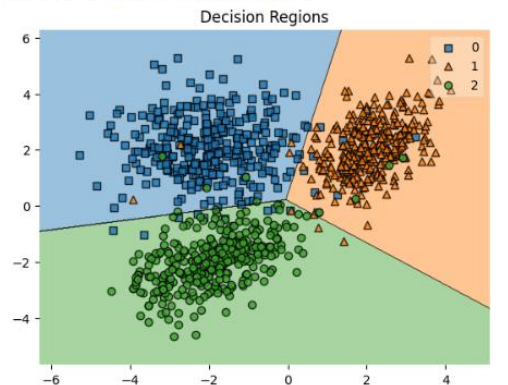
```

(1000, 2) (1000,)



Logistic Regression Accuracy: 0.96

<Axes: title='center': 'Decision Regions'>



۱.

این دیتاست شامل ۱۳۷۲ نمونه بصورت عددی است. نمونه ها تصاویری از اسکناس های جعلی و واقعی هستند. هر داده دارای ۴ ویژگی است. مسئله دارای دو کلاس (باینری کلاسیفیکیشن) است، به طوریکه • نمایانگر کلاس اسکناس جعلی و ۱ نمایانگر کلاس اسکناس واقعی است. هدف تشخیص اسکناس های واقعی و جعلی از یکدیگر است.

در این بخش تلاش میکنیم فرمت دیتاست را از حالت txt به CSV تبدیل کنیم.

۲.

اهمیت فرآیند بر زدن

شافل کردن یا بر زدن داده ها به معنای تصادفی کردن ترتیب نمونه ها در مجموعه داده است. این فعل معمولاً در مراحل آماده سازی داده در یادگیری ماشین به کار می رود. این اقدام می تواند در موارد زیر مفید باشد:

جلوگیری از overfitting، حفظ تنوع در داده ها، کاهش وابستگی به ترتیب اولیه داده ها

```
import pandas as pd
import numpy as np
from sklearn.utils import shuffle
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression , SGDClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# upload data in collab from google drive
!gdown 1QH8cd1nLFo_RgUOLnIckgBNRR2QU8r4m

# place dataset in folder
import os
folder_name = "Data"
if not os.path.exists(folder_name):
    os.makedirs(folder_name)

%cd Data

#change dataset format from txt => csv
read_file = pd.read_csv (r'/content/data_banknote_authentication.txt')
headerlist = ['f1' , 'f2','f3','f4','genuine']
read_file.to_csv("/content/data_banknote_authentication.csv" ,header = headerlist)

#upload dataset as dataframe
df = pd.read_csv("/content/data_banknote_authentication.csv")

# Shuffling datas
df = shuffle(df)

X = df[['f1', 'f2', 'f3', 'f4']].values      #feature/input
y = df[['genuine']].values                 #target/output

# Split train test datas
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2) #20% of datas go for test
df, X, y, x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

پس از بر زدن داده ها،
۲۰ درصد آن ها را برای
ارزیابی جدا کرده و باقی
داده ها را برای فرایند
آموزش به کار میگیریم.

```

Downloading...
From: https://drive.google.com/uc?id=1QH8cd1nLFo\_RgUOLnICkg8NRR2QU8r4m
To: /content/Data/Data/Data/Data/data_banknote_authentication.txt
100% 46.4k/46.4k [00:00<00:00, 58.5MB/s]
/content/Data/Data/Data/Data/Data
(      Unnamed: 0      f1      f2      f3      f4  genuine
534      534  1.94760 -4.77380  8.52700 -1.86680      0
111      111  3.23510  9.64700 -3.20740 -2.59480      0
1109     1109 -3.89520  3.81570 -0.31304 -3.81940      1
720      720 -0.45062 -1.36780  7.08580 -0.40303      0
726      726 -2.64790 10.13740 -1.33100 -5.47070      0
...      ...      ...      ...      ...      ...
1283     1283  1.20800  4.07440 -4.76350 -2.61290      1
218      218 -1.30000 10.26780 -2.95300 -5.86380      0
217      217 -0.16735  7.62740  1.20610 -3.62410      0
1177     1177 -2.07540  1.27670 -0.64206 -1.26420      1
208      208  2.17210 -0.73874  5.46720 -0.72371      0

[1371 rows x 6 columns],
array([[ 1.9476 , -4.7738 ,  8.527 , -1.8668 ],
       [ 3.2351 ,  9.647 , -3.2074 , -2.5948 ],
       [-3.8952 ,  3.8157 , -0.31304, -3.8194 ],
       ...,
       [-0.16735,  7.6274 ,  1.2061 , -3.6241 ],
       [-2.0754 ,  1.2767 , -0.64206, -1.2642 ],
       [ 2.1721 , -0.73874,  5.4672 , -0.72371]]),
array([[0],
       [0],
       [1],
       ...,
       [0],
       [1],
       [0]]),
(1096, 4),
(275, 4),
(1096, 1),
(275, 1))

```

۳.

در این بخش بصورت دستی فرآیند آموزش را نوشته و برای هر قسمت تابع مدنظر را تعریف میکنیم. ابتدا داده های ارزیابی و آموزش را جدا میکنیم. سپس با استفاده از گرادینان نزولی داده های دو کلاس را از هم جدا میکنیم. سپس تابع اتلاف را محاسبه و در نهایت دقت ارزیابی داده ها را تعیین میکنیم.

```

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

# model
def logistic_regression(x , w):
    y_hat = sigmoid(x @ w)
    return y_hat

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

def gradient(x , y , y_hat):
    grads = (x.T @ (y_hat - y)) / len(y)
    return grads

def gradient_descent(w , eta , grads):
    w -= eta*grads
    return w

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

```

افزایش تعداد ایپاک ها نشان دهنده افزایش دقت است و کاهش خطاست.

با بررسی مقادیر محاسبه شده نیز درمی یابیم که دقت روش بالا رفته است.

ما بصورت قطعی از روی تابع اتلاف نمی توانیم درمورد عملکرد مدل نظر بدهیم، زیرا:

ممکن است مدل به جای آموزش حفظ کرده باشد و صرفا تقلید کند و یا اینکه


```
x_train = np.hstack((np.ones((len(x_train) , 1)) , x_train))
x_train.shape
```

```
(1096, 5)
```

```
m = 4          #number of features
w = np.random.randn(m+1 , 1)
print(w.shape)
eta = 0.01
n_epochs = 2000
```

```
(5, 1)
```

```
error_hist = []

for epoch in range(n_epochs):
    # prediction
    y_hat = logistic_regression(x_train , w)

    # loss
    e = bce(y_train , y_hat)
    error_hist.append(e)

    # gradients
    grads = gradient(x_train , y_train , y_hat)

    # gradient descent
    w = gradient_descent(w, eta, grads)

    # show each 100 data
    if(epoch + 1) % 100 == 0:
        print(f'Epoch = {epoch}, \t E = {e:.4}, \t w={w.T[0]}')
```

```
Epoch = 99,      E = 0.5011,      w=[-0.23403927 -0.16697414 -0.09596951  0.05391951  0.00397796]
Epoch = 199,    E = 0.2714,      w=[-0.16196924 -0.51908587 -0.23538073 -0.16203908 -0.11674525]
Epoch = 299,    E = 0.2109,      w=[-0.07868287 -0.68861279 -0.32252507 -0.27117561 -0.17916654]
Epoch = 399,    E = 0.1801,      w=[ 2.81950114e-04 -8.00674181e-01 -3.86410032e-01 -3.48746982e-01
-2.19315843e-01]
Epoch = 499,    E = 0.1603,      w=[ 0.07396893 -0.88454607 -0.43747521 -0.41036   -0.24731354]
Epoch = 599,    E = 0.1461,      w=[ 0.14284695 -0.95163818 -0.48027003 -0.46204253 -0.26771867]
Epoch = 699,    E = 0.1352,      w=[ 0.20750695 -1.00759599 -0.51722275 -0.50682741 -0.28300125]
Epoch = 799,    E = 0.1263,      w=[ 0.26847004 -1.0556306   -0.54980097 -0.54648473 -0.29464707]
Epoch = 899,    E = 0.119,      w=[ 0.32616927 -1.09774458 -0.5789675   -0.58215126 -0.30361444]
Epoch = 999,    E = 0.1128,      w=[ 0.38096205 -1.13527156 -0.6053917   -0.61460748 -0.31055362]
Epoch = 1099,   E = 0.1074,      w=[ 0.43314534 -1.16914441 -0.62955957 -0.64441608 -0.31592342]
Epoch = 1199,   E = 0.1027,      w=[ 0.48296861 -1.20004114 -0.65183602 -0.6719963   -0.3200577 ]
Epoch = 1299,   E = 0.09852,      w=[ 0.53064375 -1.22846905 -0.67250257 -0.69767884 -0.32320565]
Epoch = 1399,   E = 0.09478,      w=[ 0.57635268 -1.25482094 -0.69178124 -0.72171407 -0.32555716]
Epoch = 1499,   E = 0.0914,      w=[ 0.62025309 -1.27940066 -0.70985046 -0.74431039 -0.3272595 ]
Epoch = 1599,   E = 0.08834,      w=[ 0.66248284 -1.30245357 -0.72685589 -0.76563674 -0.32842863]
Epoch = 1699,   E = 0.08554,      w=[ 0.70316342 -1.32417786 -0.74291809 -0.78583338 -0.32915697]
Epoch = 1799,   E = 0.08298,      w=[ 0.74240259 -1.34473627 -0.75813805 -0.80501811 -0.32951906]
Epoch = 1899,   E = 0.08062,      w=[ 0.78029655 -1.36426398 -0.77260125 -0.82329899 -0.32957554]
Epoch = 1999,   E = 0.07843,      w=[ 0.81693165 -1.38287437 -0.78638073 -0.84073776 -0.3293762 ]
```

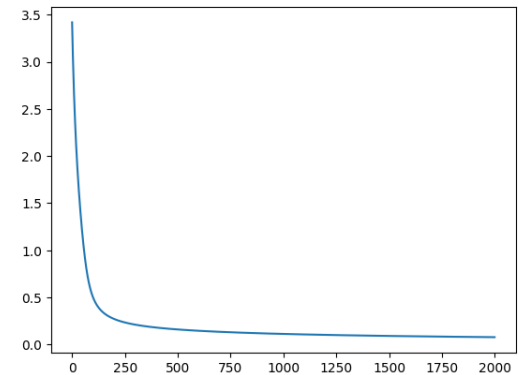
نویز و داده های پرت را نیز
یاد گرفته باشد.

در حالتی که داده نا متعادل
باشند صرفا اتکا به تابع
اتلاف کافی نیست بلکه باید
به دیگر معیارهای ارزیابی
مانند دقت نیز توجه لازم را
داشت.

برای حل این مشکل می
توان از مجموعه داده های
تست، استفاده از معیارهای
ارزیابی مختلف و... بهره برد.

```
plt.plot(error_hist)
```

[<matplotlib.lines.Line2D at 0x7e1236c06da0>]



```
# accuracy
x_test = np.hstack((np.ones((len(x_test) , 1)), x_test))
x_test.shape
```

(275, 5)

```
y_hat = logistic_regression(x_test , w)
accuracy(y_test , y_hat)
```

0.9818181818181818

۴.

نرمالسازی داده ها یک مرحله مهم در پردازش و تحلیل داده هاست که هدف آن ایجاد یک دسته بندی یکنواخت از داده ها برای اجتناب از مشکلات ناشی از مقیاس ها و واحدهای مختلف در داده ها است. دو روش متداول برای نرمالسازی داده ها عبارتند از:

Min-Max Scaling: در این روش، داده ها به گونه ای تغییر میکنند که حداقل و حداکثر آنها به ترتیب به یک مقدار نگاشته می شوند. فرمول نرمالسازی Min-Max برای یک داده X به صورت زیر است:

$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

: Z-score Standardization

در این روش، داده ها به گونه ای تغییر میکنند که میانگین آنها صفر و انحراف معیاری آنها یک شود. فرمول نرمالسازی Z-score برای یک داده X به صورت زیر است:

$$X_{\text{standardized}} = \frac{X - \mu}{\sigma}$$

اگر توزیع داده ها نسبت به هم مهم است، ممکن است از Z-score Standardization استفاده کنید. اگر میخواهید داده ها را به یک بازه خاص نگاشت کنید، Min-Max Scalین مناسب تر است.

```
# normalized data :
max1 = df[['f1' , 'f2','f3','f4']].max()

min1 = df[['f1' , 'f2','f3','f4']].min()

for i in range(4):
    df[f'f{i+1}'] = (df[f'f{i+1}']-min1[i])/(max1[i]- min1[i])
    print(df[f'f{i+1}'])

534      0.648285
111      0.741132
1109     0.226936
720      0.475339
726      0.316884
...
1283     0.594949
218      0.414087
217      0.495767
1177     0.358169
208      0.664474
Name: f1, Length: 1371, dtype: float64
534      0.336741
111      0.876347
1109     0.658148
720      0.464189
726      0.894697
...
1283     0.667828
218      0.899576
217      0.800776
1177     0.563142
208      0.487727
Name: f2, Length: 1371, dtype: float64
534      0.595046
111      0.089547
1109     0.214231
720      0.532961
726      0.170379
...
1283     0.022513
218      0.100506
217      0.279673
1177     0.200058
208      0.463235
Name: f3, Length: 1371, dtype: float64
534      0.607527
111      0.541331
1109     0.429981
720      0.740625
726      0.279831
...
1283     0.539686
218      0.244087
217      0.447739
1177     0.662320
208      0.711466
Name: f4, Length: 1371, dtype: float64
```

```
X = df[['f1' , 'f2','f3','f4']].values
y = df[['genuine']].values
X , y
```

```
(array([[0.64828476, 0.33674092, 0.59504599, 0.60752703],
       [0.74113176, 0.8763466 , 0.08954703, 0.54133137],
       [0.22693609, 0.65814771, 0.21423137, 0.42998081],
       ...,
       [0.4957669 , 0.80077606, 0.27967347, 0.44773907],
       [0.35816945, 0.56314196, 0.20005773, 0.6623203 ],
       [0.6644744 , 0.48772708, 0.46323476, 0.71146603]]),
 array([[0],
       [0],
       [1],
       ...,
       [0],
       [1],
       [0]]))
```

در این بخش از روش اول
برای نرمال سازی استفاده
میکنیم. بدین منظور مینیمم
و ماکسیمم داده ی هر
ستون را مشخص میکنیم و
با فرمول گفته شده داده ها
را نرمال میکنیم.
باید توجه داشت داده های
ستون هدف نیاز به نرمال
سازی ندارند.
پس از نرمال سازی داده
های ارزیابی و آموزش را جدا
میکنیم.

در این بخش با استفاده از داده های نرمال سازی شده تمام مراحل قبل را تکرار میکنیم.

افزایش تعداد اپیاک ها =

افزایش دقت

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

```
((1096, 4), (275, 4), (1096, 1), (275, 1))
```

```
x_train = np.hstack((np.ones((len(x_train) , 1)) , x_train))
x_train.shape
```

```
(1096, 5)
```

```
m = 4
w = np.random.randn(m+1 , 1)
print(w.shape)
eta = 0.01
n_epochs = 6000
```

```
(5, 1)
```

```
error_hist = []
for epoch in range(n_epochs):
    # prediction
    y_hat = logistic_regression(x_train , w)

    # loss
    e = bce(y_train , y_hat)
    error_hist.append(e)

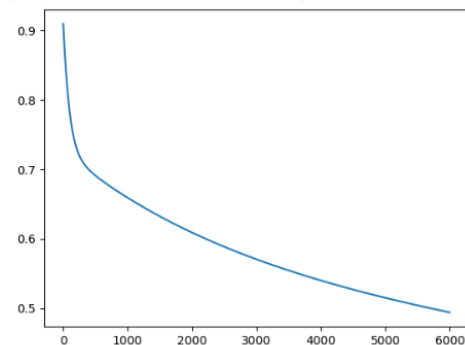
    # gradients
    grads = gradient(x_train , y_train , y_hat)

    # gradient descent
    w = gradient_descent(w , eta , grads)

    # show each 100 data
    if(epoch + 1) % 100 == 0:
        print(f"Epoch = {epoch} , \t E = {e:.4} \t w={w.T[0]}")

Epoch = 499 , E = 0.6908 w=[ 0.26769159 0.14213008 -0.17347412 -0.29588339 -0.71414765]
Epoch = 599 , E = 0.6837 w=[ 0.28104341 0.07674192 0.21827959 -0.27394388 -0.70086397]
Epoch = 699 , E = 0.6771 w=[ 0.30063055 0.01555162 -0.25849234 -0.25097861 -0.68363981]
Epoch = 799 , E = 0.6709 w=[ 0.32377172 -0.04295796 -0.29572514 -0.22772918 -0.66429526]
Epoch = 899 , E = 0.6648 w=[ 0.34890202 -0.09967478 -0.33092543 -0.20462357 -0.64388872]
Epoch = 999 , E = 0.659 w=[ 0.37510495 -0.15512182 -0.36465445 -0.1819074 -0.62303669]
Epoch = 1099 , E = 0.6533 w=[ 0.40184309 -0.20960904 -0.39724785 -0.15971945 -0.6020974 ]
Epoch = 1199 , E = 0.6478 w=[ 0.42880157 -0.26332148 -0.42890899 -0.13813561 -0.58127755]
Epoch = 1299 , E = 0.6424 w=[ 0.45579652 -0.31637086 -0.45976349 -0.11719461 -0.5606947 ]
Epoch = 1399 , E = 0.6372 w=[ 0.48272122 -0.36882575 -0.48989118 -0.09691326 -0.54041392]
Epoch = 1499 , E = 0.6321 w=[ 0.50951449 -0.42072933 -0.51934492 -0.07729539 -0.5204694 ]
Epoch = 1599 , E = 0.6272 w=[ 0.53614194 -0.47210987 -0.54816165 -0.05833721 -0.50087718]
Epoch = 1699 , E = 0.6224 w=[ 0.56258502 -0.52298683 -0.57636887 -0.04003042 -0.48164265]
Epoch = 1799 , E = 0.6178 w=[ 0.58883445 -0.57337454 -0.60398846 -0.02236412 -0.46276502]
Epoch = 1899 , E = 0.6132 w=[ 0.61488642 -0.62328426 -0.63103895 -0.00532593 -0.4442399 ]
Epoch = 1999 , E = 0.6088 w=[ 0.64074029 -0.67272546 -0.65753681 -0.01109734 -0.42606091]
Epoch = 2099 , E = 0.6045 w=[ 0.6663973 -0.72170655 -0.68349724 -0.0269193 -0.40822055]
Epoch = 2199 , E = 0.6003 w=[ 0.69185973 -0.77023529 -0.70893458 -0.04215369 -0.39071074]
Epoch = 2299 , E = 0.5962 w=[ 0.71713043 -0.81831905 -0.73386259 -0.05681427 -0.37352317]
Epoch = 2399 , E = 0.5923 w=[ 0.74221259 -0.86596493 -0.75829462 -0.07091468 -0.35664948]
Epoch = 2499 , E = 0.5884 w=[ 0.76710953 -0.91317988 -0.78224363 -0.08446843 -0.34008137]
Epoch = 2599 , E = 0.5846 w=[ 0.79182463 -0.95997068 -0.8057223 -0.09748883 -0.32381066]
Epoch = 2699 , E = 0.5809 w=[ 0.81636124 -1.00634403 -0.82874301 -0.10998894 -0.30782931]
Epoch = 2799 , E = 0.5773 w=[ 0.8407227 -1.05230651 -0.85131788 -0.1219816 -0.2921295 ]
Epoch = 2899 , E = 0.5738 w=[ 0.86491227 -1.09786462 -0.87345876 -0.1334794 -0.27670358]
.....
plt.plot(error_hist)
```

```
[<matplotlib.lines.Line2D at 0x7e1233e2e260>]
```



```
x_test = np.hstack((np.ones((len(x_test) , 1)), x_test))
x_test.shape
```

(275, 5)

```
y_hat = logistic_regression(x_test , w)
accuracy(y_test , y_hat)
```

0.8363636363636363

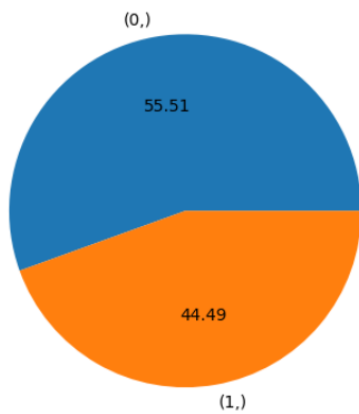
۶.

با استفاده از این دستور تعیین میکنیم داده ها متعادل هستند یا نه.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# value_count
new_y = pd.DataFrame(y, columns=['Column_A'])
new_y.value_counts()
new_y.value_counts().plot.pie(autopct = "%.2f")
```

<Axes: >



با بررسی نمودار درمی یابیم که داده ها متعادل نبوده و تعداد نمونه های ملاس ها با هم برابر نیست.

این موضوع مشکلاتی را به همراه دارد:

یادگیری نا کافی کلاس هایی با نمونه کم، بایاس در پیش بینی ها، عدم تعادل در دقت برای کلاس های مختلف، افزایش هزینه و ... برای حل این مشکل ما از الگوریتم

undersampling بهره

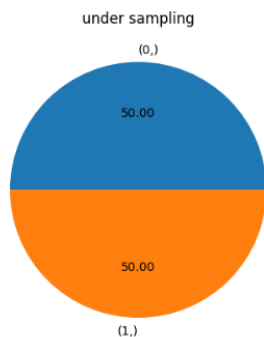
میگیریم تا نمونه هایی که بیشتر هستند را کم کنیم. به این صورت تعداد نمونه های کلاس ها را با هم برابر میکنیم.

```
! pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.10.1)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.11.0-py3-none-any.whl (235 kB)
    235.6/235.6 kB 4.3 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.11.3)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.2.0)
Installing collected packages: imbalanced-learn
  Attempting uninstall: imbalanced-learn
    Found existing installation: imbalanced-learn 0.10.1
    Uninstalling imbalanced-learn-0.10.1:
      Successfully uninstalled imbalanced-learn-0.10.1
Successfully installed imbalanced-learn-0.11.0
```

```
# undersampling
from imblearn.under_sampling import RandomUnderSampler

y = pd.DataFrame(y, columns=[''])
rus = RandomUnderSampler(sampling_strategy=1)
x_res_undersampling, y_res_undersampling = rus.fit_resample(X, y)
ax = y_res_undersampling.value_counts().plot.pie(autopct = '%.2f')
_ = ax.set_title("under sampling")
```



.Y

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

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

```
model = LogisticRegression()
model.fit(X, y)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataC
y = column_or_1d(y, warn=True)
```

```
LogisticRegression
```

```
y_hat = model.predict(x_test)
model.score(x_test, y_test)
#y_test.shape ,
y_hat = y_hat.reshape(244, 1)
#y_test.shape , y_hat.shape

from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_hat)
score
```

```
0.9631147540983607
```

افزایش دقت ارزیابی و
آموزش داده ها با استفاده از
طبقه بندی داده ها و
متعادل کردنشان با کتابخانه
های آماده

دقت در حالت بدون تعادل
کتر از حالت معادل است.

```
# Unbalanced datas
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression , SGDClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

X = df[["f1" , "f2" , "f3" , "f4"]].values
y = df[["genuine"]].values
x_train , x_test , y_train , y_test = train_test_split(X , y , test_size = 0.2)
model = LogisticRegression(random_state = 13, solver='sag', max_iter=200)
model.fit(X , y)
y_hat = model.predict(x_test)
model.score(x_test , y_test)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataCo
  y = column_or_1d(y, warn=True)
0.9781818181818182
```

۳.

۱.

۲۵۳۶۶۲ داده داریم.

این دیتاست را می توان در دسته کلاسیفیکیشن باینری قرار داد، زیرا حالت های تارگت ما دو تاست.
ستون اول دیتاست مربوط به عارضه حمله قلبی (۱ و ۰) و همان تارگت/هدف است.
۲۱ ستون دیگر ویژگی ها هستند. برخی از این ویژگی ها دو وضعیتی هستند.

```
# upload data in collab from google drive
!gdown 1itgkDw3V40d8yzG8GUhIPs2HyPiJRrKB

# place dataset in folder
import os
folder_name = "Data"
if not os.path.exists(folder_name):
    os.makedirs(folder_name)

%cd Data

Downloading...
From: https://drive.google.com/uc?id=1itgkDw3V40d8yzG8GUhIPs2HyPiJRrKB
To: /content/Data/heart_disease_health_indicators.csv
100% 11.8M/11.8M [00:00<00:00, 142MB/s]
/content/Data/Data
```

۲.

در این بخش ۱۰۰ نمونه از هر کلاس را انتخاب میکنیم و یک دیتافریم جدید تشکیل میدهیم.

```

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/heart_disease_health_indicators.csv')
#df

# Separate the data into two classes
class_1_data = df[df['HeartDiseaseorAttack'] == 1].head(100)
#print("class 1 = \n", class_1_data, "\n")
class_0_data = df[df['HeartDiseaseorAttack'] == 0].head(100)
#print("class 0 = \n", class_0_data, "\n")

# Create two new DataFrames for each class
df_class_1 = pd.DataFrame(class_1_data, copy=True)
df_class_0 = pd.DataFrame(class_0_data, copy=True)

# to use these two DataFrames for further steps
df_class_0.to_csv('class_0_data.csv', index=False)
df_class_1.to_csv('class_1_data.csv', index=False)
class_1_data
class_0_data

```

	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income
8	1	1	1	1	30	1	0	2	0	1	...	1	0	5	30	30	1	0	9	5	1
20	1	1	1	1	22	0	1	0	0	1	...	1	0	3	30	0	1	0	12	4	4
26	1	1	1	1	37	1	1	2	0	0	...	1	0	5	0	0	1	1	10	6	5
27	1	1	1	1	28	1	0	2	0	0	...	1	0	4	0	0	0	1	12	2	4
47	1	1	1	1	25	1	0	0	0	1	...	1	0	2	1	0	0	1	10	4	7
...
750	1	1	1	1	25	1	0	2	1	1	...	1	0	2	1	0	0	0	11	6	5
774	1	0	1	1	29	0	0	0	0	1	...	1	0	5	0	30	1	1	13	5	6
784	1	1	0	1	31	0	0	2	0	0	...	1	0	5	0	30	1	1	13	4	5
797	1	1	1	1	30	0	0	0	1	0	...	1	0	5	0	30	1	0	13	3	3
807	1	1	1	1	32	0	0	2	1	0	...	1	0	5	30	30	0	0	7	5	7

100 rows × 22 columns

	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Income
0	0	1	1	1	40	1	0	0	0	0	...	1	0	5	18	15	1	0	9	4	3
1	0	0	0	0	25	1	0	0	1	0	...	0	1	3	0	0	0	0	7	6	1
2	0	1	1	1	28	0	0	0	0	1	...	1	1	5	30	30	1	0	9	4	8
3	0	1	0	1	27	0	0	0	1	1	...	1	0	2	0	0	0	0	11	3	6
4	0	1	1	1	24	0	0	0	1	1	...	1	0	2	3	0	0	0	11	5	4
...
111	0	1	1	1	26	0	0	0	0	1	...	1	0	5	0	20	1	0	13	6	4
112	0	0	0	1	30	1	0	0	0	0	...	1	0	3	0	0	0	1	9	4	6
113	0	1	0	1	27	0	0	0	1	1	...	1	0	3	0	0	0	1	8	6	8
114	0	0	1	1	26	0	0	0	1	1	...	1	0	2	1	0	0	0	8	6	6
115	0	0	0	1	28	1	0	0	1	1	...	1	1	3	30	30	1	1	3	5	6

100 rows × 22 columns

۳.

دو کلاسی که در دو دیتافریم قرار دادیم را در این بخش با استفاده از کتابخانه های آماده تفکیک میکنیم.


```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score
from sklearn.utils import shuffle

# Combine the two classes into a new DataFrame
combined_df = pd.concat([df_class_0, df_class_1], ignore_index=True)
combined_df = shuffle(combined_df)
# Separate features (X) and target (y)
X = combined_df.drop('HeartDiseaseorAttack', axis=1) # Assuming 'target' is the column you want to predict
y = combined_df['HeartDiseaseorAttack']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

##### Train & find Accuracy #####
##LogisticRegression
model = LogisticRegression(solver='sag', max_iter=1000, random_state=13)
model.fit(X_train, y_train) #train
model_predict = model.predict(X_test) #test...it gives us y_hat_test
log_accuracy = accuracy_score(y_test, model_predict) # accuracy
print(f'Logistic Regression Accuracy: {log_accuracy:.2f}')

##SGDClassifier()
model1 = SGDClassifier(loss='log_loss', random_state=13)
model1.fit(X_train, y_train) #train
model1_predict = model1.predict(X_test) #test
SGD_accuracy = accuracy_score(y_test, model1_predict) #accuracy
print(f'SGD Classifier Accuracy: {SGD_accuracy:.2f}')

X.shape, model_predict.shape, model1_predict.shape

```

```

Logistic Regression Accuracy: 0.72
SGD Classifier Accuracy: 0.62
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:350: Com
warnings.warn(
((200, 21), (40,)), (40,))

```

```

## Plot decision boundary for LogisticRegression

import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification

# Apply PCA to reduce the data to 2D for visualization
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)

# Logistic Regression
model = LogisticRegression()
model.fit(X_train_pca, y_train)

# Create a meshgrid to plot the decision boundaries
h = 0.02
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1

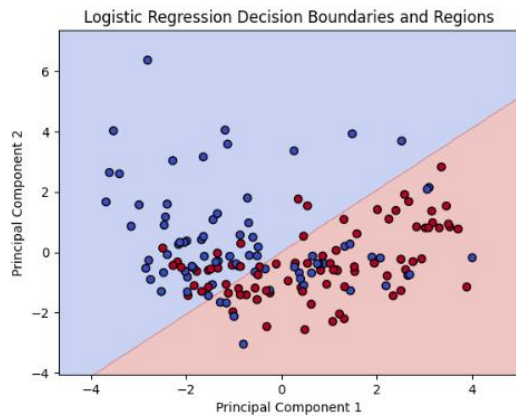
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict the labels for each point in the meshgrid
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundaries and regions
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.3)

# Plot the examples
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap=plt.cm.coolwarm, edgecolors='k', marker='o')
plt.title('Logistic Regression Decision Boundaries and Regions')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

```



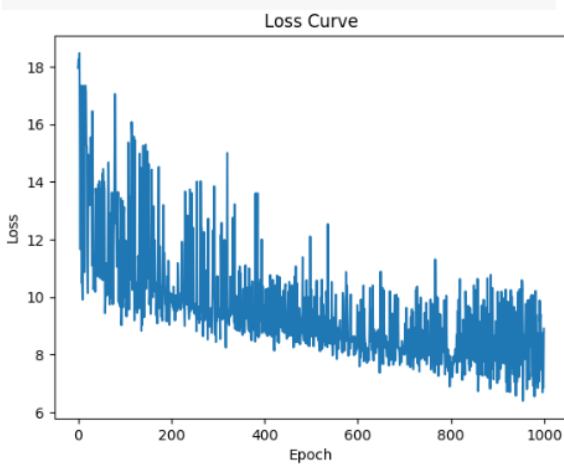
۴.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import log_loss

modell = SGDClassifier(loss='log', random_state=13)
losses = []
epochs = 1000

for _ in range(epochs):
    modell.partial_fit(x_train, y_train, [0, 1])
    loss = log_loss(y_train, modell.predict_proba(x_train))
    losses.append(loss)

plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss Curve')
plt.show()
```



۵.

از دیگر شاخص های ارزیابی می توان به confusion matrix اشاره کرد. که در آن دیتاهای واقعی و پیش بینی شده کنار هم قرار گرفته و مقایسه میشوند.

```

from sklearn.metrics import confusion_matrix , f1_score
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test , model_predict)
F1 = f1_score(y_test , model_predict , average=None)
F1
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, model_predict)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()

```

