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

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1Won6xkyYCcJLJ7eMpVt5VA_4P0tE1nb7
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (5.0.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.13.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.11.17)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
Downloading...
From: https://drive.google.com/uc?id=1Won6xkyYCcJLJ7eMpVt5VA_4P0tE1nb7
To: /content/data_banknote_authentication.txt
100% 46.4k/46.4k [00:00<00:00, 76.6MB/s]
```

```
df = pd.read_csv('/content/data_banknote_authentication.txt')
df
```

	x1	x2	x3	x4	y
0	3.62160	8.66610	-2.8073	-0.44699	0
1	4.54590	8.16740	-2.4586	-1.46210	0
2	3.86600	-2.63830	1.9242	0.10645	0
3	3.45660	9.52280	-4.0112	-3.59440	0
4	0.32924	-4.45520	4.5718	-0.98880	0
...	...	...	...	...	...
1367	0.40614	1.34920	-1.4501	-0.55949	1
1368	-1.38870	-4.87730	6.4774	0.34179	1
1369	-3.75030	-13.45860	17.5932	-2.77710	1
1370	-3.56370	-8.38270	12.3930	-1.28230	1
1371	-2.54190	-0.65804	2.6842	1.19520	1

1372 rows × 5 columns

```
shuffled_data = shuffle(df)
shuffled_data.to_csv('created_data.csv', index=False)
print(shuffled_data)
```

```
           x1       x2        x3        x4  y
782  -0.3481 -0.38696  -0.47841  0.626270  1
364   5.7823  5.57880  -2.40890 -0.056479  0
262   1.8114  7.60670  -0.97880 -2.466800  0
139  -0.2062  9.22070  -3.70440 -6.810300  0
1033  1.5077  1.95960  -3.05840 -0.122430  1
...      ...      ...       ...       ... ..
861  -2.3797 -1.44020   1.12730  0.160760  1
199   5.8862  5.87470  -2.81670 -0.300870  0
428   3.4246 -0.14693   0.80342  0.291360  0
1235 -3.5359  0.30417   0.65690 -0.295700  1
949  -3.1158 -8.62890  10.44030  0.971530  1

[1372 rows x 5 columns]
```

```
df1 = pd.read_csv('/content/created_data.csv')
```

```
df1 = pd.read_csv('/content/created_data.csv')
df1
```

	x1	x2	x3	x4	y
0	-0.3481	-0.38696	-0.47841	0.626270	1
1	5.7823	5.57880	-2.40890	-0.056479	0
2	1.8114	7.60670	-0.97880	-2.466800	0
3	-0.2062	9.22070	-3.70440	-6.810300	0
4	1.5077	1.95960	-3.05840	-0.122430	1
...	...	...	...	...	...
1367	-2.3797	-1.44020	1.12730	0.160760	1
1368	5.8862	5.87470	-2.81670	-0.300870	0
1369	3.4246	-0.14693	0.80342	0.291360	0
1370	-3.5359	0.30417	0.65690	-0.295700	1
1371	-3.1158	-8.62890	10.44030	0.971530	1

1372 rows × 5 columns

Logistic Regression (from Scratch)

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

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

Binary Cross Entropy (BCE)

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

Gradient

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

Gradient Descent

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

Accuracy

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

train آموزش داده های نرمالایز شده

```
X = df1[['x1','x2','x3','x4']].values
y = df1[['y']].values
X, y
```

```
(array([[-0.3481  , -0.38696 , -0.47841 ,  0.62627 ],
        [ 5.7823  ,  5.5788  , -2.4089  , -0.056479],
        [ 1.8114  ,  7.6067  , -0.9788  , -2.4668  ],
```

```
       ...,
       [ 3.4246  , -0.14693 ,  0.80342 ,  0.29136 ],
       [-3.5359  ,  0.30417 ,  0.6569  , -0.2957  ],
       [-3.1158  , -8.6289  , 10.4403  ,  0.97153 ]]),
 array([[1],
       [0],
       [0],
       ...,
       [0],
       [1],
       [1]]))
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)

x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((1097, 4), (275, 4), (1097, 1), (275, 1))
```

```
# Create separate StandardScaler instances for each feature
scaler_x1 = StandardScaler()
scaler_x2 = StandardScaler()
scaler_x3 = StandardScaler()
scaler_x4 = StandardScaler()

# Fit and transform each feature separately
x_train_normalized_x1 = scaler_x1.fit_transform(x_train[:, [0]])
x_train_normalized_x2 = scaler_x2.fit_transform(x_train[:, [1]])
x_train_normalized_x3 = scaler_x3.fit_transform(x_train[:, [2]])
x_train_normalized_x4 = scaler_x4.fit_transform(x_train[:, [3]])

# Transform the corresponding test data using the same scalers
x_test_normalized_x1 = scaler_x1.fit_transform(x_test[:, [0]])
x_test_normalized_x2 = scaler_x1.fit_transform(x_test[:, [1]])
x_test_normalized_x3 = scaler_x1.fit_transform(x_test[:, [2]])
x_test_normalized_x4 = scaler_x1.fit_transform(x_test[:, [3]])

# Concatenate the normalized features back into a 2D array
x_train_normalized = np.hstack((x_train_normalized_x1,x_train_normalized_x2,x_train_normalized_x3,x_train_normalized_x4))
x_test_normalized = np.hstack((x_test_normalized_x1,x_test_normalized_x2,x_test_normalized_x3,x_test_normalized_x4))

# Check the shapes of the normalized data
x_train_normalized.shape, x_test_normalized.shape, y_train.shape, y_test.shape
```

```
((1097, 4), (275, 4), (1097, 1), (275, 1))
```

```
y_hat = logistic_regression(x_train_normalized, np.random.randn(4, 1))
print(y_hat.shape)
```

```
(1097, 1)
```

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

```
(1097, 5)
```

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

```
(5, 1)
```

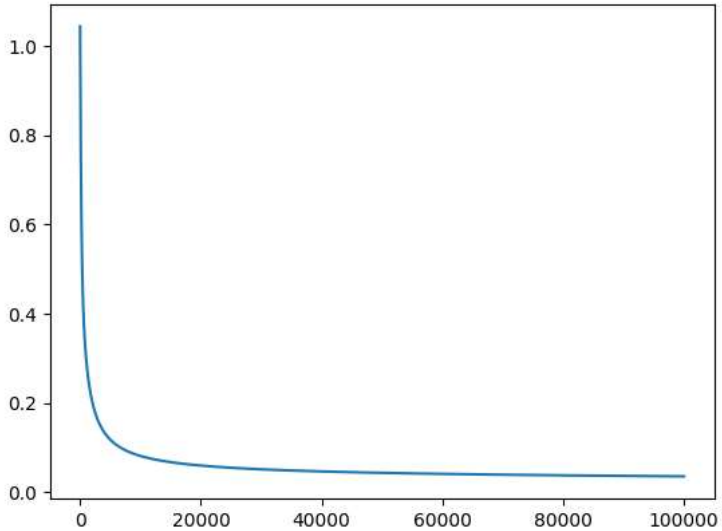
```
eta = 0.01
n_epochs = 100000 #N
```

```
error_hist = []

for epoch in range(n_epochs):
    # predictions
    y_hat = logistic_regression(x_train_normalized, w)

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

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

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

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

```
Epoch=99942,     E=0.03544,      w=[-2.3216952  -6.30965226 -6.29526561 -5.78164573  0.1441052 ]
Epoch=99943,     E=0.03544,      w=[-2.32170489 -6.30966762 -6.29528428 -5.78166227  0.14410393]
Epoch=99944,     E=0.03544,      w=[-2.32171458 -6.30968299 -6.29530295 -5.78167881  0.14410266]
Epoch=99945,     E=0.03544,      w=[-2.32172427 -6.30969836 -6.29532161 -5.78169534  0.14410138]
Epoch=99946,     E=0.03544,      w=[-2.32173395 -6.30971372 -6.29534028 -5.78171188  0.14410011]
Epoch=99947,     E=0.03544,      w=[-2.32174364 -6.30972909 -6.29535894 -5.78172842  0.14409883]
Epoch=99948,     E=0.03544,      w=[-2.32175333 -6.30974446 -6.29537761 -5.78174495  0.14409756]
Epoch=99949,     E=0.03544,      w=[-2.32176302 -6.30975982 -6.29539627 -5.78176149  0.14409628]
Epoch=99950,     E=0.03544,      w=[-2.32177271 -6.30977519 -6.29541494 -5.78177803  0.14409501]
Epoch=99951,     E=0.03544,      w=[-2.32178239 -6.30979055 -6.2954336  -5.78179456  0.14409373]
Epoch=99952,     E=0.03544,      w=[-2.32179208 -6.30980592 -6.29545227 -5.7818111   0.14409246]
Epoch=99953,     E=0.03544,      w=[-2.32180177 -6.30982128 -6.29547093 -5.78182763  0.14409119]
Epoch=99954,     E=0.03544,      w=[-2.32181146 -6.30983665 -6.2954896  -5.78184417  0.14408991]
Epoch=99955,     E=0.03544,      w=[-2.32182115 -6.30985202 -6.29550826 -5.7818607   0.14408864]
Epoch=99956,     E=0.03544,      w=[-2.32183083 -6.30986738 -6.29552693 -5.78187724  0.14408736]
Epoch=99957,     E=0.03544,      w=[-2.32184052 -6.30988275 -6.29554559 -5.78189377  0.14408609]
Epoch=99958,     E=0.03544,      w=[-2.32185021 -6.30989811 -6.29556425 -5.78191031  0.14408481]
Epoch=99959,     E=0.03544,      w=[-2.3218599  -6.30991348 -6.29558292 -5.78192684  0.14408354]
Epoch=99960,     E=0.03544,      w=[-2.32186958 -6.30992884 -6.29560158 -5.78194338  0.14408227]
Epoch=99961,     E=0.03544,      w=[-2.32187927 -6.30994421 -6.29562025 -5.78195991  0.14408099]
Epoch=99962,     E=0.03544,      w=[-2.32188896 -6.30995957 -6.29563891 -5.78197645  0.14407972]
Epoch=99963,     E=0.03544,      w=[-2.32189864 -6.30997494 -6.29565757 -5.78199298  0.14407844]
Epoch=99964,     E=0.03544,      w=[-2.32190833 -6.3099903  -6.29567624 -5.78200952  0.14407717]
Epoch=99965,     E=0.03544,      w=[-2.32191802 -6.31000566 -6.2956949  -5.78202605  0.14407589]
Epoch=99966,     E=0.03544,      w=[-2.32192771 -6.31002103 -6.29571356 -5.78204259  0.14407462]
Epoch=99967,     E=0.03544,      w=[-2.32193739 -6.31003639 -6.29573222 -5.78205912  0.14407335]
Epoch=99968,     E=0.03544,      w=[-2.32194708 -6.31005176 -6.29575089 -5.78207565  0.14407207]
Epoch=99969,     E=0.03544,      w=[-2.32195677 -6.31006712 -6.29576955 -5.78209219  0.1440708 ]
Epoch=99970,     E=0.03544,      w=[-2.32196645 -6.31008248 -6.29578821 -5.78210872  0.14406952]
Epoch=99971,     E=0.03544,      w=[-2.32197614 -6.31009785 -6.29580687 -5.78212526  0.14406825]
Epoch=99972,     E=0.03544,      w=[-2.32198583 -6.31011321 -6.29582554 -5.78214179  0.14406698]
Epoch=99973,     E=0.03544,      w=[-2.32199551 -6.31012858 -6.2958442  -5.78215832  0.1440657 ]
Epoch=99974,     E=0.03544,      w=[-2.3220052  -6.31014394 -6.29586286 -5.78217486  0.14406443]
Epoch=99975,     E=0.03544,      w=[-2.32201488 -6.3101593  -6.29588152 -5.78219139  0.14406315]
Epoch=99976,     E=0.03544,      w=[-2.32202457 -6.31017467 -6.29590018 -5.78220792  0.14406188]
Epoch=99977,     E=0.03544,      w=[-2.32203426 -6.31019003 -6.29591884 -5.78222445  0.1440606 ]
Epoch=99978,     E=0.03544,      w=[-2.32204394 -6.31020539 -6.29593751 -5.78224099  0.14405933]
Epoch=99979,     E=0.03544,      w=[-2.32205363 -6.31022075 -6.29595617 -5.78225752  0.14405806]
Epoch=99980,     E=0.03544,      w=[-2.32206331 -6.31023612 -6.29597483 -5.78227405  0.14405678]
Epoch=99981,     E=0.03544,      w=[-2.322073   -6.31025148 -6.29599349 -5.78229058  0.14405551]
Epoch=99982,     E=0.03544,      w=[-2.32208269 -6.31026684 -6.29601215 -5.78230712  0.14405423]
Epoch=99983,     E=0.03544,      w=[-2.32209237 -6.31028221 -6.29603081 -5.78232365  0.14405296]
Epoch=99984,     E=0.03544,      w=[-2.32210206 -6.31029757 -6.29604947 -5.78234018  0.14405169]
Epoch=99985,     E=0.03544,      w=[-2.32211174 -6.31031293 -6.29606813 -5.78235671  0.14405041]
Epoch=99986,     E=0.03544,      w=[-2.32212143 -6.31032829 -6.29608679 -5.78237324  0.14404914]
Epoch=99987,     E=0.03544,      w=[-2.32213111 -6.31034365 -6.29610545 -5.78238978  0.14404786]
Epoch=99988,     E=0.03544,      w=[-2.3221408  -6.31035902 -6.29612411 -5.78240631  0.14404659]
Epoch=99989,     E=0.03544,      w=[-2.32215049 -6.31037438 -6.29614277 -5.78242284  0.14404531]
Epoch=99990,     E=0.03544,      w=[-2.32216017 -6.31038974 -6.29616143 -5.78243937  0.14404404]
Epoch=99991,     E=0.03544,      w=[-2.32216986 -6.3104051  -6.29618009 -5.7824559   0.14404277]
Epoch=99992,     E=0.03544,      w=[-2.32217954 -6.31042046 -6.29619875 -5.78247243  0.14404149]
Epoch=99993,     E=0.03544,      w=[-2.32218923 -6.31043582 -6.29621741 -5.78248896  0.14404022]
Epoch=99994,     E=0.03544,      w=[-2.32219891 -6.31045119 -6.29623607 -5.78250549  0.14403894]
Epoch=99995,     E=0.03544,      w=[-2.3222086  -6.31046655 -6.29625473 -5.78252202  0.14403767]
Epoch=99996,     E=0.03544,      w=[-2.32221828 -6.31048191 -6.29627338 -5.78253855  0.1440364 ]
Epoch=99997,     E=0.03544,      w=[-2.32222797 -6.31049727 -6.29629204 -5.78255508  0.14403512]
Epoch=99998,     E=0.03544,      w=[-2.32223765 -6.31051263 -6.2963107  -5.78257161  0.14403385]
Epoch=99999,     E=0.03544,      w=[-2.32224734 -6.31052799 -6.29632936 -5.78258814  0.14403257]
```

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

```
[<matplotlib.lines.Line2D at 0x78f88f324160>]
```




```
y_hat = logistic_regression(x_train_normalized, w)
accuracy(y_train, y_hat)
```

```
0.98359161349134
```

```
x_test_normalized = np.hstack((np.ones((len(x_test_normalized), 1)), x_test_normalized))
y_hat = logistic_regression(x_test_normalized, w)
accuracy(y_test, y_hat)
```

```
0.9781818181818182
```

```
x_nemune_normalized=x_test_normalized[[154,101,54,57,30]]
y_hat = logistic_regression(x_nemune_normalized, w)
y_test1=y_test[[154,101,54,57,30]]
y_hat,y_test1
```

```
(array([[2.65402606e-05],
        [9.96407629e-01],
        [2.77419457e-05],
        [6.20260894e-06],
        [9.88040653e-01]]),
 array([[0],
        [1],
        [0],
        [0],
        [1]]))
```