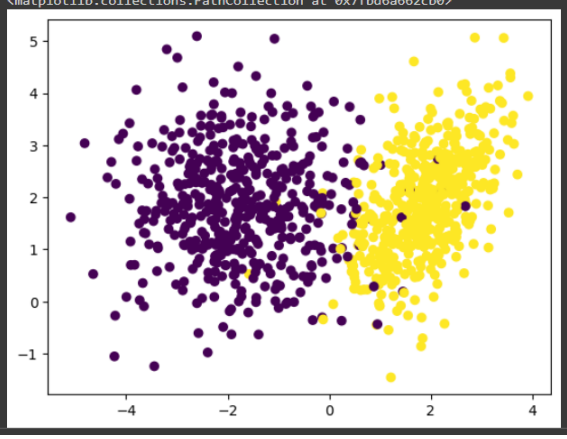
سوال اول

۱- در این قسمت با استفاده از sklearn.datasets یک دیتاست با ۱۰۰۰ نمونه و ۲ کلاس و ۲ ویژگی تعریف کردیم و سپس با استفاده از دستور scatter آنرا رسم کردیم.

Q1:

```
[ ] 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_classes=2, n_redundant=0, random_state=13, n_clusters_per_class=1, class_sep=1.8)
plt.scatter(X[:,0], X[:,1], c=y)
```

```
<matplotlib.collections.PathCollection at 0x7fbd6a662cb0>
```



*پارامتر n_clusters_per_class رو برای دریافت یک حالت از توزیع در هر کلاس استفاده کردم.

*پارامتر random_state را با توجه به دو رقم شماره دانشجویی برابر با ۱۳ قرار داده ام.

*پارامتر class_sep را برابر ۱.۸ قرار دادیم تا مقداری داده ها از همدیگر فاصله داشته باشند.

۲-

Q2:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, SGDClassifier
```

```
[150] x_train, x_test, y_train, y_test = train_test_split(X, y,random_state=13, test_size = 0.2)
```

```
[154] model1 = LogisticRegression(solver='sag', max_iter=210)
      model1.fit(x_train, y_train)
      model1.predict(x_test)

      print("Accuracy of test data: %", model1.score(x_test, y_test)*100)
      print("Accuracy of train data: %", model1.score(x_train, y_train)*100)
```

```
Accuracy of test data: % 98.0
Accuracy of train data: % 96.875
```

```
model2 = SGDClassifier(loss='log_loss', random_state=13, max_iter=2000)
model2.fit(x_train, y_train)
model2.predict(x_test)

print("Accuracy of test data: %", model2.score(x_test, y_test)*100)
print("Accuracy of train data: %", model2.score(x_train, y_train)*100)
```

```
Accuracy of test data: % 97.0
Accuracy of train data: % 96.25
```

در این قسمت با استفاده از دو طبقه بند آماده پایتون (SGDClassifier و LogisticRegression) دو قسمت موجود در دیتاستی که ایجاد کردیم را تفکیک کردیم.
۲۰ درصد از نمونه هارا بعنوان train و ۸۰ درصد داده هارا بعنوان test انتخاب کردیم.

در طبقه بندی با استفاده از LogisticRegression از پارامتر solver نوع sag برای الگوریتم بهینه سازی استفاده کردیم و همچنین از پارامتر max_iter با تعداد ۲۱۰ استفاده کردیم تا دقت افزایش یابد.

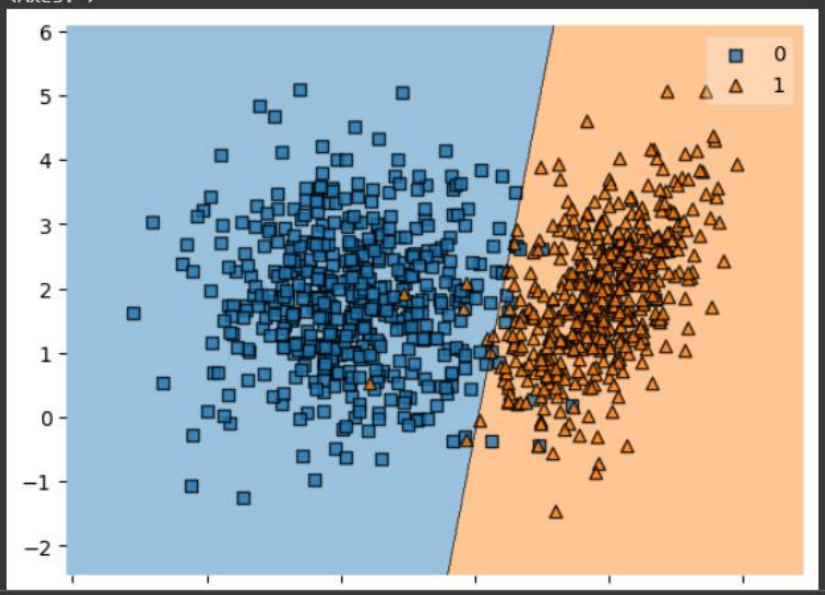
در طبقه بندی با استفاده از SGDClassifier پارامتر loss را برابر با log_loss قرار دادیم تا از کراس آنتروپی استفاده کرده باشیم.
همچنین برای دقت بیشتر از max_iter با تعداد ۲۰۰۰ استفاده کردیم.

در نهایت دقت هرنوع از روش های تفکیک را برای نمونه های تست و ترین نمایش دادیم.

نرخ یادگیری را همان مقدار دیفالت که optimal می باشد قرار دادیم.

-۳

```
[101] model3 = LogisticRegression()
      model3.fit(X,y)

      ▾ LogisticRegression
      LogisticRegression()

[103] from mlxtend.plotting import plot_decision_regions

      plot_decision_regions(X, y, clf=model3)

      <Axes: >
```

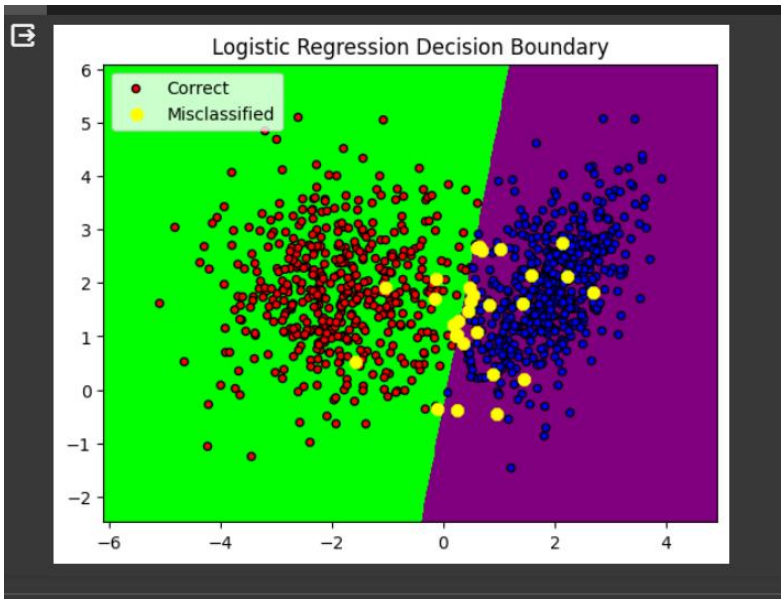



```
                                                    0s
```

در این قسمت با استفاده از کتابخوانه mlxtend.plotting و ماژول plot_decision_regions مرز و نواحی تصمیم گیری را به همراه نمونه ها نمایش دادیم.

البته بدون استفاده از ماژول بالا و توابع آماده هم میتوان به صورت دستی نواحی تصمیم گیری را مشخص کرد و نمونه های اشتباه دسته بندی شده را با رنگ دیگر نمایش داد که بصورت زیر می باشد:

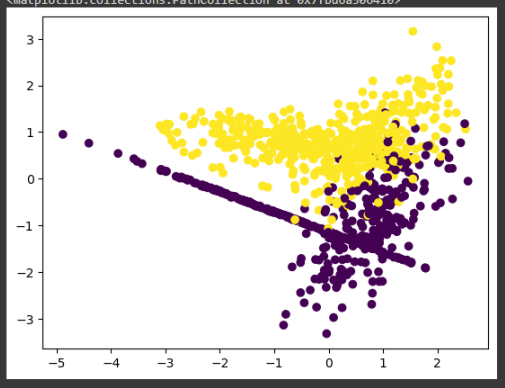
```python
# برای نمایش متفاوت نمونه هایی که اشتباه طبقه بندی شدند از کد زیر استفاده میکنیم
def plot_decision_boundary(model, X, y, title):
    h = .02  # مشخص کردن سایز هر مرحله

    # تشکیل نقشه رنگ
    cmap_light = ListedColormap(['#00FF00', '#800080'])
    cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

    # رسم نواحی تصمیم گیری
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # رسم نقاط
    correct_predictions = model.predict(X) == y
    incorrect_predictions = ~correct_predictions

    plt.scatter(X[correct_predictions, 0], X[correct_predictions, 1], c=y[correct_predictions], cmap=cmap_bold, marker='o', edgecolor='k', s=20, label='Correct')
    plt.scatter(X[incorrect_predictions, 0], X[incorrect_predictions, 1], marker='8', color='yellow', s=50, label='Misclassified')

    plt.title(title)
    plt.legend()
    plt.show()


plot_decision_boundary(model3, X, y, 'Logistic Regression Decision Boundary')
```



۴- در کل با افزایش پارامترهای n_redundant، n_informative، n_repeated و n_clusters_per_class و کاهش پارامتر class_sep میتوان دیتاست تولید شده را چالش برانگیزتر و سخت تر کرد که در ادامه ما با تغییر تعدادی از این پارامترها موارد قبلی را تکرار میکنیم.

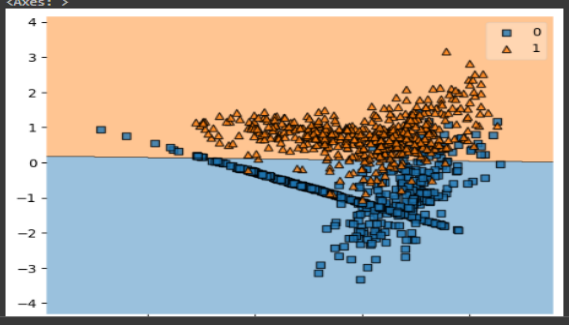
تغییرات: class_sep=0.8، n_clusters_per_class=2

```
Q4:

X, y = make_classification(n_samples=1000, n_features=2, n_classes=2, n_redundant=0, random_state=13, class_sep=0.8, n_clusters_per_class=2)
plt.scatter(X[:,0], X[:,1], c=y)

<matplotlib.collections.PathCollection at 0x7fbd6a306410>
```



تکرار مراحل یک تا سه:

```
[166] a_train, a_test, b_train, b_test = train_test_split(A, b,random_state=13, test_size = 0.2)

      model5 = LogisticRegression(solver='sag', max_iter=210)
      model5.fit(a_train, b_train)
      model5.predict(a_test)

      print("Accuracy of test data: %", model5.score(a_test, b_test)*100)
      print("Accuracy of train data: %", model5.score(a_train, b_train)*100)

      Accuracy of test data: % 92.5
      Accuracy of train data: % 89.875

      model6 = SGDClassifier(loss='log_loss', random_state=13, max_iter=2000)
      model6.fit(a_train, b_train)
      model6.predict(a_test)

      print("Accuracy of test data: %", model6.score(a_test, b_test)*100)
      print("Accuracy of train data: %", model6.score(a_train, b_train)*100)

      Accuracy of test data: % 93.0
      Accuracy of train data: % 89.75

      model7 = LogisticRegression()
      model7.fit(A,b)
      plot_decision_regions(A, b, clf=model7)

      <Axes: >
```



مشاهده می کنیم که به دلیل سخت تر شدن سیستم دقت نیز کاهش یافته است.

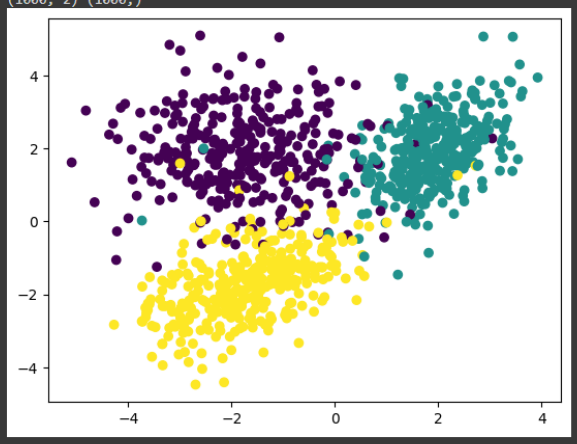
۵- اگر یک کلاس به کلاس های دیتاست قسمت ۱ اضافه کنیم، ممکن است برخی از پارامترهای make_classification همخوانی نداشته باشد که باید آن هارا تغییر دهیم. همچنین ممکن است تعادل کلاس ها از بین برود و تعداد نمونه های کلاس ها تفاوت زیادی داشته باشند که باید متعادل سازی شوند.

همچنین در فرایند آموزش و ارزیابی نیز باید دقت کنیم که تعداد نمونه های برای آموزش و ارزیابی به طور تقریبا برابر از هر کلاس باشد تا دقت افزایش یابد.

در این حالت برای اینکه از هر کلاس به تعداد برابر(به نسبت وزن هر کلاس) داده برای آموزش و ارزیابی انتخاب کنیم، از stratify در تابع train_test_split استفاده می کنیم:

```
X2, y2 = make_classification(n_samples=1000, n_features=2, n_classes=3, n_redundant=0, random_state=13, n_clusters_per_class=1, class_sep=1.8)
plt.scatter(X2[:,0], X2[:,1], c=y2)
print(X2.shape, y2.shape)
```

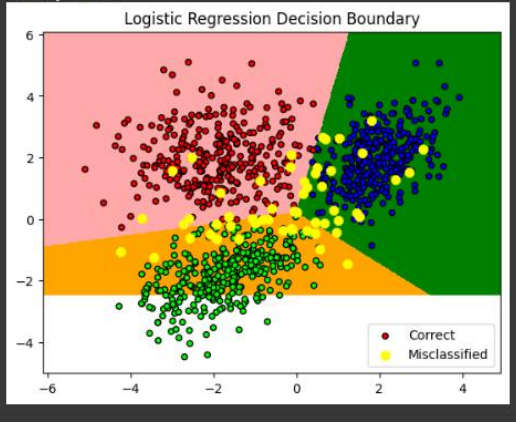
```
(1000, 2) (1000,)
```



برای رسم نواحی تصمیم گیری مشابه قسمت قبل عمل می کنیم:

```
from sklearn.metrics import accuracy_score, classification_report
# آموزش مدل به روش تجستیک رگرسیون

x2_train, x2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2, random_state=13)
logreg_model = LogisticRegression(multi_class='multinomial', solver='lbfgs', random_state=13)
logreg_model.fit(x2_train, y2_train)

# عملیات پیدیکت بر روی داده های تست
logreg_predictions = logreg_model.predict(x2_test)

# محاسبه و نمایش دقت
accuracy = accuracy_score(y2_test, logreg_predictions)
print("Accuracy: %", accuracy*100)

# تابع رسم نواحی تصمیم گیری
def plot_decision_boundary(model, X2, y2, title):
    h = .02  # Step size in the mesh

    # تشکیل نقشه رنگ
    cmap_light = ListedColormap(['#FFAAAA', '#008000', '#FFA500'])
    cmap_bold = ListedColormap(['#FF0000', '#0000FF', '#00FF00'])

    # رسم نواحی تصمیم گیری
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # رسم نقاط
    correct_predictions = model.predict(X2) == y2
    incorrect_predictions = ~correct_predictions

    plt.scatter(X2[correct_predictions, 0], X2[correct_predictions, 1], c=y2[correct_predictions], cmap=cmap_bold, marker='o', edgecolor='k', s=20, label='Correct')
    plt.scatter(X2[incorrect_predictions, 0], X2[incorrect_predictions, 1], marker='8', color='yellow', s=50, label='Misclassified')

    plt.title(title)
    plt.legend()
    plt.show()


plot_decision_boundary(logreg_model, X2, y2, 'Logistic Regression Decision Boundary')
```

```
Accuracy: % 94.0
```



سوال دوم

۱- در این دیتاست داده ها از تصاویری که برای ارزیابی یک روش احراز هویت برای اسکناس ها گرفته شده بود، استخراج شده اند. در این دیتاست ۱۳۷۲ داده با ویژگی های واقعی وجود دارد که از تصاویری از نمونه های واقعی و جعلی اسکناس گرفته شده اند. برای دیجیتالی کردن از یک دوربین که برای بازرسی چاپ اسکناس استفاده میشود، استفاده کرده ایم که تصاویر نهایی دارای ۴۰۰*۴۰۰ پیکسل هستند. با توجه به عدسی و فاصله تا جسم مورد بررسی، تصاویری در مقیاس خاکستری با وضوح حدود ۶۶۰ نقطه در اینچ به دست آمده است. ابزار تبدیل موجک برای استخراج ویژگی ها از تصاویر استفاده شده است.

اطلاعات متغیر اضافی این دیتاست:

۱) واریانس تصویر تبدیل موجک (پیوسته)
۲) انحراف معیار تصویر تبدیل موجک (پیوسته)
۳) کشیدگی تصویر تبدیل موجک (پیوسته)
۴) آنتروپی تصویر (پیوسته)
۵) کلاس (عدد صحیح)

```
[10] #https://drive.google.com/file/d/116ho57xYxPgjTET5qOR30VLfwoxFAaI4/view?usp=sharing

     !pip install --upgrade --no-cache-dir gdown
     !gdown 116ho57xYxPgjTET5qOR30VLfwoxFAaI4

Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.7.1)
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: six in /usr/local/lib/python3.10/dist-packages (from gdown) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
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) (3.4)
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) (2023.7.22)
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=116ho57xYxPgjTET5qOR30VLfwoxFAaI4
To: /content/data_banknote_authentication.txt
100% 46.4k/46.4k [00:00<00:00, 86.6MB/s]

     import pandas as pd
     df = pd.read_csv('/content/data_banknote_authentication.txt', delimiter=',')
     df.to_csv('/content/data_banknote_authentication.csv', index=False)
     df
```

	Variance	Skewness	kurtosis	Entropy	Class
0	3.62160	8.66610	-2.8073	-0.44699	0

در این قسمت بعد از بارگذاری فایل در گوگل درایو با استفاده از دستور gdown دیتاست را در محیط گوگل کلود قرار دادیم. چون فرمت فایل دیتاست txt. بود با استفاده از دستورات پایتونی آن را به فرمت csv. تغییر داده و فایل جدید را در همان قسمت ذخیره کردیم.

۲- فرایند برزدن یا مخلوط کردن با تصادفی کردن داده ها از بایاس در آموزش و تست مدل جلوگیری می کند و همچنین باعث می شود از پدیده overfitting که به معنای پیش پردازش در مدل سازی می باشد جلوگیری کنیم. در واقع overfitting به این معناست که مدل در برابر داده های آموزشی عملکرد مناسب دارد اما برای داده های جدید عملکرد ضعیف خواهد داشت.

```
Q2:

[7]  shuffled_df = df.sample(frac=1, random_state=13).reset_index(drop=True)

[8]  from sklearn.model_selection import train_test_split

[47] X = shuffled_df[['Variance', 'Skewness', 'kurtosis', 'Entropy']].values
     y = shuffled_df[['Class ']].values

[48] 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,

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

(توجه: ۵ سرستون Variance, Skewness, Kurtosis, Entropy, Class در فایل دیتاست وجود نداشت و خودم به صورت دستی این سرستون هارا اضافه کردم تا بتوانم در کد برای تعریف فیچرها و تارگت استفاده کنم.)

در این قسمت با استفاده از تابع sample() داده ها را مخلوط کردیم و سپس با استفاده از تعریف پارامترهای X و y توانستیم ۲۰ درصد از داده هارا به عنوان داده آموزش و ۸۰ درصد از آنهارا به عنوان داده ارزیابی دسته بندی کنیم.

۳- در این قسمت می خواهیم بدون استفاده از کتابخوانه های آماده پایتون مدل، تابع اتلاف و الگوریتم یادگیری و ارزیابی را کد نویسی کنیم.

```
Q3:

Logistic Regression Model

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

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

در این قسمت با استفاده از تابع سیگموید مدل خود را طراحی کردیم.

```
Binary Cross Entropy(BCE)

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

در این قسمت طبق تعاریف ریاضی تابع اتلاف خودرا کدنویسی کردیم.

```
Gradient

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

Gradient Descent

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

در این قسمت نیز طبق تعاریف ریاضی به محاسبه گرادیان ها و گرادیان کاهشی و اپدیت وزن ها می پردازیم.

Accuracy

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

در این قسمت نیز با توجه به فرمول های ریاضی به محاسبه دقت میپردازیم.

Train

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

```
(1097, 5)
```

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

     eta = 0.01
     n_epochs = 2000
```

```
(5, 1)
```

و در این قسمت به آموزش مدل میپردازیم. M تعداد ویژگی های ما می باشد و با مقدار دهی رندوم برای w های اولیه و مقداردهی eta = 0.01 و تعداد تکرار مراحل(n_epochs) با ۲۰۰۰ ، فرایند آموزش را انجام می دهیم.

Error History

```
error_hist = []

for epoch in range(n_epochs):
    # predictions
    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 decsent
    w = gradient_descent(w, eta, grads)

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

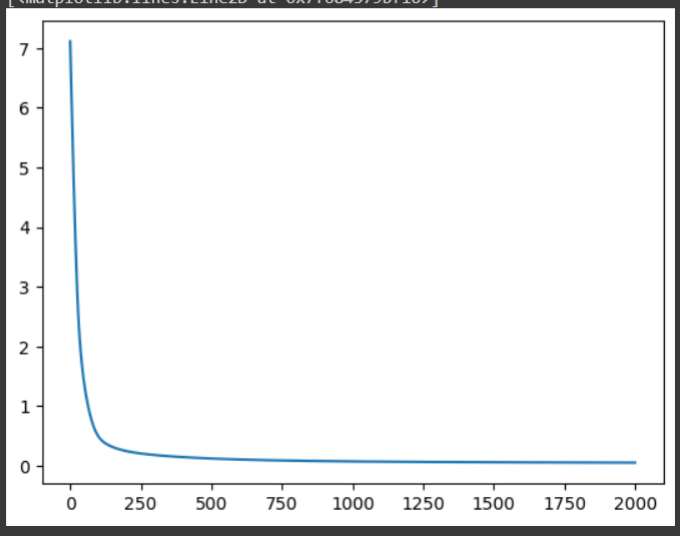
```
Epoch=99,       E=0.4986,       w+[ 0.98357198 -1.3958614  -0.21506504 -0.66564375  1.53157177]
Epoch=199,      E=0.245,        w+[ 1.10188353 -1.6078239  -0.2637444  -0.52904105  1.16389609]
Epoch=299,      E=0.1792,       w+[ 1.17770638 -1.6779093  -0.28715952 -0.55525008  0.9358144 ]
Epoch=399,      E=0.1441,       w+[ 1.23737237 -1.70553855 -0.32989868 -0.5979267   0.77220624]
Epoch=499,      E=0.1209,       w+[ 1.28798747 -1.71714001 -0.37683634 -0.64070111  0.64411152]
Epoch=599,      E=0.1044,       w+[ 1.33212539 -1.72125543 -0.42331995 -0.6802108   0.54097058]
Epoch=699,      E=0.09256,      w+[ 1.37118663 -1.72157815 -0.46769556 -0.71612529  0.4570476 ]
Epoch=799,      E=0.08376,      w+[ 1.40617749 -1.72005031 -0.50929568 -0.74881676  0.38822283]
Epoch=899,      E=0.07709,      w+[ 1.43789258 -1.71780593 -0.54788108 -0.77873792  0.33124391]
```

و در نهایت در این قسمت تاریخچه ای برای خطاها ایجاد کردیم که در آن تا زمانی که epoch در رنجی که مشخص کردیم قرار دارد، ابتدا خروجی مدل را با استفاده از تابع logistic_regression که تعریف کردیم بدست می آورد. سپس ارور را با استفاده از تابع bce محاسبه کرده و در گام بعدی با استفاده از تابع gradient با گرفتن سه مقدار x_train, y_train و y_hat گرادیان را محاسبه میکند. و در قسمت اخر گرادیان های اپدیت شده را با استفاده از تابع gradient_descent محاسبه میکنیم.

Plot Error

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

```
[<matplotlib.lines.Line2D at 0x7f684579bf10>]
```



در این قسمت با کمک دستور plt.plot() ما تابع اتلاف خود را رسم میکنیم.

Test

```
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)
```

```
0.9745454545454545
```

در نهایت نیز نتیجه دقت ارزیابی خود را بر روی داده های تست نمایش دادیم که برابر با ۰.۹۷۴۵ می باشد.

با مشاهده نمودار تابع اتلاف متوجه میشویم که در ابتدای آموزش خطاها بسیار زیاد می باشند ولی به مرور زمان با انجام دوره های بیشتر و نزدیک شدن به ۲۰۰۰ مرتبه تکرار، میزان خطاها کاهش یافته و به سمت صفر نزدیک میشود. همچنین با ادامه دادن آموزش میتوان خطارا بیشتر از این مقدار کاهش داد که ما نیازی به ادامه فرایند آموزش نداشتیم و به حد مطلوبی از دقت رسیدیم.

خیر نمیتوان از روی نمودار تابع اتلاف و قبل از مرحله ارزیابی با قطعیت درمورد عملکرد مدل نظر داد چرا که ممکن است داده های انتخاب شده برای آموزش دارای تعداد نابرابر از هر کلاس باشد بنابرین مدل ما بایاس خواهد داشت و در مرحله ارزیابی عملکرد مطلوبی نخواهد داشت. برای حل این مشکل می بایست به تعداد برابر از هر کلاس داده انتخاب کرد و برای آموزش استفاده کرد.

۴- اهمیت فرایند نرمال سازی داده ها در پایتون به دو دلیل جلوگیری از overfitting (عملکرد خوب در برابر داده های جدید) و افزایش سرعت آموزش خواهد بود.

دو روش نرمال سازی داده ها MaxScaler و StandardScaler می باشند. در StandardScaler مقادیر میانگین صفر و مقدار انحراف معیار یک می باشد در حالی که در MaxScaler بزرگترین مقدار داده ها به عنوان ۱ نرمال سازی می شود. به عبارت دیگر در MaxScaler داده ها بین ۰ و ۱ نرمال سازی می شوند.

۵- در این قسمت با استفاده از StandardScaler داده ها را نرمالسازی کرده و مراحل ۱ تا ۳ را تکرار میکنیم.

Q4:

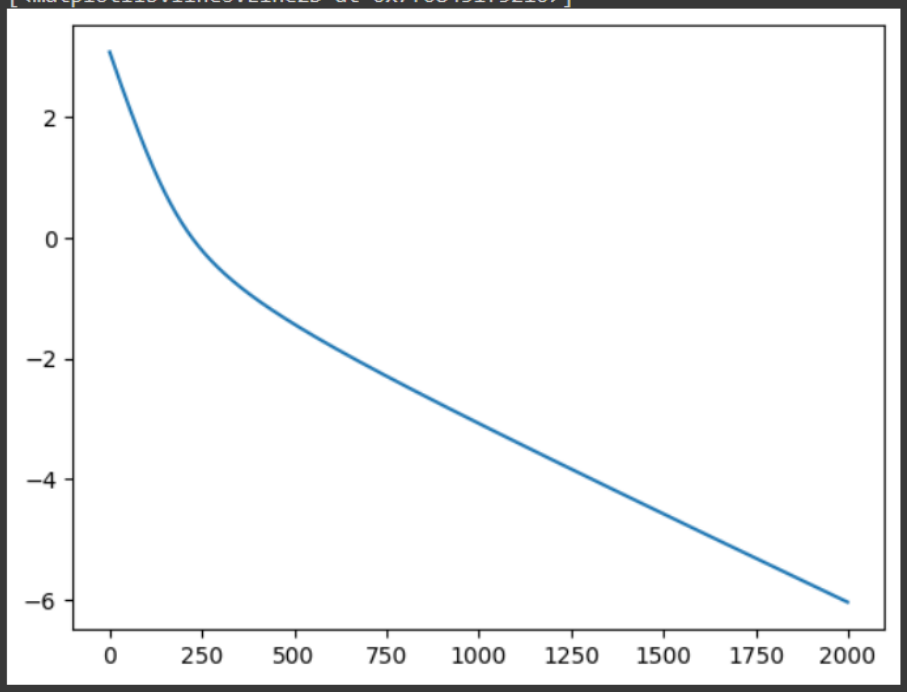
```
from sklearn.preprocessing import StandardScaler
data2 = pd.read_csv('/content/data_banknote_authentication.csv')

scaler = StandardScaler()

scaled_data2 = scaler.fit_transform(data2)

scaled_df2 = pd.DataFrame(scaled_data2, columns=data2.columns)

shuffled_scaled_df2 = scaled_df2.sample(frac=1, random_state=13).reset_index(drop=True)

X2 = shuffled_scaled_df2[['Variance', 'Skewness', 'kurtosis', 'Entropy']].values
y2 = shuffled_scaled_df2[['Class ']].values

x_train2, x_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.2)

x_train2 = np.hstack((np.ones((len(x_train2), 1)), x_train2))

m_new = 4
w_new = np.random.randn(m_new+1, 1)
print(w_new.shape)

eta_new = 0.01
n_epochs_new = 2000
```

```
(5, 1)
```

```
error_hist_new = []

for epoch_new in range(n_epochs_new):
    # predictions
    y_hat2 = logistic_regression(x_train2, w_new)

    # loss
    e_new = bce(y_train2, y_hat2)
    error_hist_new.append(e_new)

    # gradients
    grads_new = gradient(x_train2, y_train2, y_hat2)

    # gradient decsent
    w_new = gradient_descent(w_new, eta_new, grads_new)

    if (epoch_new+1) % 100 == 0:
        print(f'Epoch={epoch_new}, \t E={e_new:,.4},\t w={w_new.T[0]}')
```

```
plt.plot(error_hist_new)
```

```
[<matplotlib.lines.Line2D at 0x7f68451f5210>]
```



نمایش ۵ نمونه:

```
0.0
```

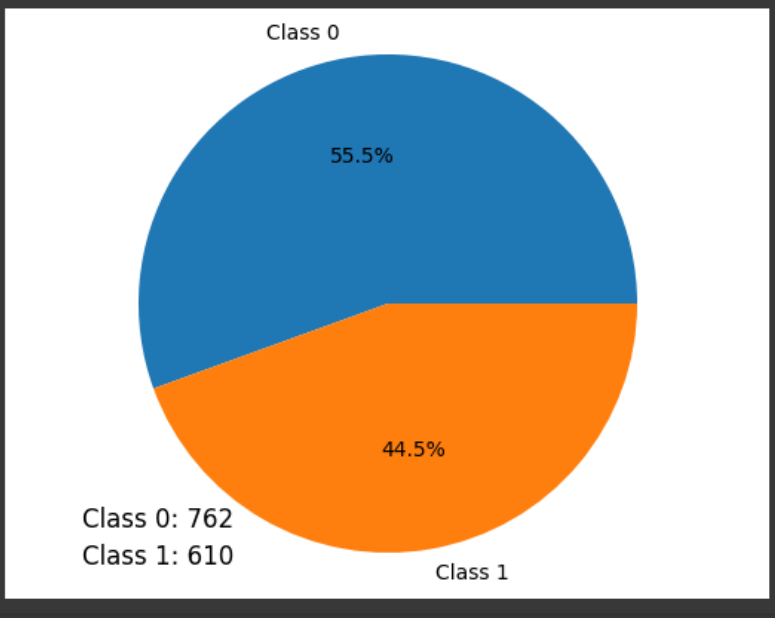
```
for i in range(5):
    print(f"data {i+1}: y_hat2 = {y_hat2[i]}, y2= {y2[i]}")
```

```
data 1: y_hat2 = [0.96242389], y2= [-0.89472059]
data 2: y_hat2 = [0.91767549], y2= [1.11766736]
data 3: y_hat2 = [0.99982648], y2= [-0.89472059]
data 4: y_hat2 = [1.40564918e-06], y2= [-0.89472059]
data 5: y_hat2 = [1.10386059e-07], y2= [-0.89472059]
```

۶- در این قسمت با استفاده از دستورات زیر و توابع موجود در کتابخانه matplotlib تعداد نمونه های هر کلاس را نمایش دادیم.

Q6:

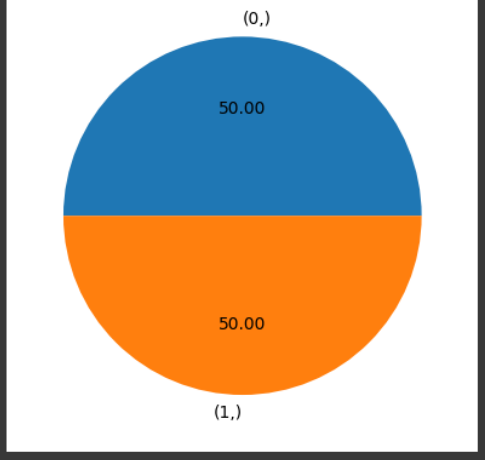
```
# محاسبه تعداد نمونه‌های هر کلاس
y_counts = df[['Class ']].value_counts()
# نمایش نمودار دایره ای
labels = ['Class 0', 'Class 1']
plt.pie(y_counts, labels=labels, autopct='%1.1f%%')
plt.axis('equal')  # این دستور برای اینه که دایره ما دایره ای باشه!

# افزودن تعداد داده‌های هر کلاس به صورت مستقیم بر روی نمودار
for i, count in enumerate(y_counts):
    plt.text(x=-0.92, y=-0.9 - i * 0.15, s=f'{labels[i]}: {count}', ha='center', fontsize=12)

plt.show()
```



مشاهده می کنیم که تعداد نمونه های دو کلاس دقیقا با یکدیگر برابر نیستند. همانطور که در گذشته گفته شد عدم تعادل در دیتاست میتواند منجر به مشکلات overfitting و بایاس مدل شود. برای حل این موضوع باید برای اموزش از تعداد برابر نمونه از هرکلاس انتخاب و استفاده کرد.

برای حل مشکلات مربوط به عدم تعادل دیتاست، میتوانیم از روشهایی مانند oversampling افزایش نمونه های کم نمونه، undersamplingکاهش نمونه های زیاد استفاده کنیم که در اینجا ما از روش undersamplingاستفاده میکنیم:

```
# 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')
```



مشاهده می کنیم که تعداد نمونه های هر کلاس دقیقا با یکدیگر برابر شد.

۷-

```
Q7:

from sklearn.linear_model import LogisticRegression

X3 = shuffled_df[['Variance', 'Skewness', 'kurtosis', 'Entropy']].values
y3 = shuffled_df[['Class ']].values

x_train3, x_test3, y_train3, y_test3 = train_test_split(X3, y3, random_state=13, stratify=y3, test_size=0.2)

model = LogisticRegression(solver='sag', max_iter=210)
model.fit(x_train3, y_train3)
model.predict(x_test3)

print("Accuracy of test data: %", model.score(x_test3, y_test3)*100)
print("Accuracy of train data: %", model.score(x_train3, y_train3)*100)

Accuracy of test data: % 99.27272727272727
Accuracy of train data: % 98.99726526891523
```

در این قسمت با استفاده از طبقه بند آماده پایتون فرایند آموزش و ارزیابی را انجام داده و دقت را نمایش دادیم. همچنین با استفاده از stratify مشکل عدم تعادل در کلاس هارا حل کردیم.

سوال سوم:

۱- این دیتاست داده حاوی انواع اطلاعات مرتبط با سلامت، عوامل سبک زندگی و اطلاعات جمعیتی برای گروهی از افراد است که آن را برای بررسی همبستگی ها و عوامل خطر بالقوه بیماری قلبی و سایر شرایط سلامتی مناسب می کند و هدف آن پیشبینی بیماری های قلبی می باشد.

این دیتاست شامل ۲۱ ویژگی و ۱ هدف و ۲۵۳۶۶۲ نمونه می باشد.

Q1:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
```

```
#https://drive.google.com/file/d/1kYwlFgwVOVX-aGZO-rMltxuL_cjPrHQz/view?usp=sharing

!pip install --upgrade --no-cache-dir gdown
!gdown 1kYwlFgwVOVX-aGZO-rMltxuL_cjPrHQz
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.6.6)
Collecting gdown
  Downloading gdown-4.7.1-py3-none-any.whl (15 kB)
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: six in /usr/local/lib/python3.10/dist-packages (from gdown) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
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) (3.4)
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) (2023.7.22)
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)
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.6.6
    Uninstalling gdown-4.6.6:
      Successfully uninstalled gdown-4.6.6
Successfully installed gdown-4.7.1
Downloading...
From: https://drive.google.com/uc?id=1kYwlFgwVOVX-aGZO-rMltxuL_cjPrHQz
To: /content/heart_disease_health_indicators.csv
100% 11.8M/11.8M [00:00<00:00, 69.0MB/s]
```

```
[5] df = pd.read_csv('/content/heart_disease_health_indicators.csv')
```

مشابه سوال قبل با استفاده از دستورgdown فایل را از گوگل درایو دریافت میکنیم.

۲-

Q2:

```
[14] X = df.iloc[:,1:21]
     y = df[['HeartDiseaseorAttack']].values

     X.shape, y.shape
```

```
((253661, 20), (253661, 1))
```

```
# انتخاب نمونه های تصادفی از هر کلاس
samples_class_0 = df[df['HeartDiseaseorAttack'] == 0].sample(n=100, random_state=13)
samples_class_1 = df[df['HeartDiseaseorAttack'] == 1].sample(n=100, random_state=13)

# ادغام داده های انتخاب شده
df_sample = pd.concat([samples_class_0, samples_class_1])

# مخلوط کردن 200 داده ادغام شده
shuffled_df = df_sample.sample(frac=1, random_state=13).reset_index(drop=True)

samples_class_0.shape, samples_class_1.shape, df_sample.shape ,shuffled_df.shape
```

```
((100, 22), (100, 22), (200, 22), (200, 22))
```

در این قسمت ابتدا ستون اول را به عنوان هدف قرار دادیم و ۲۱ ستون بعدی به عنوان فیچر ها در نظر گرفته شده اند. سپس ۱۰۰ نمونه از هر کلاس انتخاب شد. بعد این ۲۰۰ نمونه با یکدیگر ترکیب شده و مخلوط شدند.

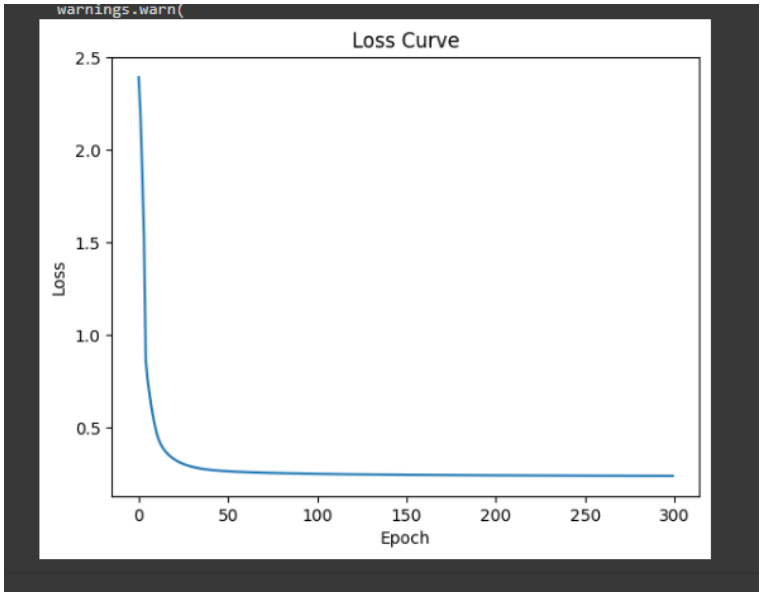
۳-

```
Q3:

[59] from sklearn.linear_model import LogisticRegression, SGDClassifier

[60] x_train, x_test, y_train, y_test = train_test_split(X, y,random_state=13, test_size = 0.2)

y_train_reshaped = np.ravel(y_train)
y_test_reshaped = np.ravel(y_test)

model1 = LogisticRegression(solver='sag', max_iter=300)
model1.fit(x_train, y_train_reshaped)
model1.predict(x_test)

print("Accuracy of test data: %", model1.score(x_test, y_test_reshaped)*100)
print("Accuracy of train data: %", model1.score(x_train, y_train_reshaped)*100)

Accuracy of test data: % 90.74763960341396
Accuracy of train data: % 90.75435622486793

[76] model2 = SGDClassifier(loss='log_loss', random_state=13, max_iter=2000)
model2.fit(x_train, y_train_reshaped)
model2.predict(x_test)

print("Accuracy of test data: %", model2.score(x_test, y_test_reshaped)*100)
print("Accuracy of train data: %", model2.score(x_train, y_train_reshaped)*100)

Accuracy of test data: % 90.72792856720477
Accuracy of train data: % 90.7262674446109
```

در این قسمت با استفاده از دوطبقه بند آماده پایتون و با در نظر گرفتن پارامترهای مناسب دو کلاس موجود در دیتاست را از همم تفکیک کرده و دقت هرروش را برروی داده های آموزش و ارزیابی نمایش دادیم.

۴- بله. با استفاده از کتابخانه های اماده در پایتون می توان تابع اتلاف را بدست آورد و رسم کرد. اما چون در مدل LogisticRegression این دستورات اماده بصورت نقطه ای عمل میکنند، نیازمند تعریف یک حلقه for به صورت زیر می باشیم که در هر مرحله میزان تابع اتلاف را محاسبه کند و پس از ذخیره کردن، آن را به کمک دستور plt نمایش دهد:

```
+ Code  + Text

[12] from sklearn.metrics import log_loss

model = SGDClassifier(loss='log', random_state=83)
losses = []
epochs = 300

for _ in range(epochs):

    model.partial_fit(x_train, y_train_reshaped, [0, 1])
    loss = log_loss(y_train , model.predict_proba(x_train))
    losses.append(loss)

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



۵- F1-Score یکی از شاخص های مهم در ارزیابی عملکرد مدل های دسته بندی است که به دقت(Precision) و بازخوانی (Recall) بستگی دارد.

دقت برابر است با تعداد پیش بینی های صحیح نسبت به تعداد کل پیش بینی ها و بازخوانی برابر است با تعداد پیش های صحیح نسبت به تعداد کل پیش بینی های مثبت است.

F1-Score به صورت زیر محاسبه میشود:

F1-Score =۲* (دقت * بازخوانی) / (دقت + بازخوانی)

مقدار F1-Scoreبین ۰ و ۱ قرار دارد که مقدار بالاتر به معنای عملکرد بهتر مدل است. F1-Score برای مواردی که تعداد نمونه‌های مثبت و منفی در داده‌ها نامتوازن است، به عنوان یک شاخص مناسب برای ارزیابی عملکرد مدل استفاده می‌شود.

Q5:

```
[42] from sklearn.metrics import f1_score

     model2 = SGDClassifier(loss='log_loss', random_state=13, max_iter=2000)
     model2.fit(x_train, y_train_reshaped)
     y_pred = model2.predict(x_test)

     f1 = f1_score(y_test, y_pred)
     print("F1-Score on test data: ", f1)

     F1-Score on test data:  0.2112676056338028
```

در واقع انتخاب معیار مناسب برای ارزیابی عملکرد مدل طبقه بندی به اهمیت هر کلاس، و قابل بخشش بودن خطای کلاس ها و موارد دیگر بستگی دارد که ما به عنوان نمونه از F1-Score استفاده کردیم.