دانشگاه صنعتی خواجه نصیرالدین طوسی

مینی پروژه اول درس مبانی سیستم هوشمند

استاد: جناب اقای دکتر مهدی علیاری شوره دلی

فاطمه اسلامی 9819393

چکیده :

در این پروژه، سه سوال در مورد یادگیری ماشین مطرح شده است .سوال اول و سوم مربوط به طبقه بندی داده ها با استفاده از الگوریتم های مختلف هستند .سوال دوم مربوط به چالش های موجود در فرآیند طبقه بندی داده ها مانند عدم تعادل کلاس ها و نرمال سازی داده ها است.

در مجموع، نتایج این پروژه نشان می دهد که الگوریتم های یادگیری ماشین می توانند دقت بالایی در طبقه بندی داده ها داشته باشند . با این حال، انتخاب الگوریتم مناسب و تنظیم فراپارامترهای آن می تواند بر عملکرد مدل تأثیر بگذارد .همچنین، چالش های موجود در فرآیند طبقه بندی داده ها مانند عدم تعادل کلاس ها و نرمال سازی داده ها می توانند بر عملکرد مدل تأثیر منفی بگذارند.

مقدمه :

یادگیری ماشین یک زمینه هوش مصنوعی است که با توسعه الگوریتم‌هایی برای یادگیری از داده‌ها سروکار دارد .این الگوریتم‌ها می‌توانند برای حل طیف گسترده‌ای از مسائل، از جمله طبقه‌بندی، رگرسیون، تشخیص الگو و یادگیری تقویتی استفاده شوند.

طبقه‌بندی یکی از متداول‌ترین مسائل در یادگیری ماشین است .در این مسئله، هدف این است که داده‌های جدید را به یکی از چندین دسته از پیش تعریف‌شده تقسیم کنیم .به عنوان مثال، می‌توان از طبقه‌بندی برای شناسایی تصاویر، طبقه‌بندی متن یا تشخیص بیماری استفاده کرد.

در این پروژه، ما سه سوال در مورد طبقه‌بندی داده‌ها با استفاده از الگوریتم‌های یادگیری ماشین بررسی خواهیم کرد . سوال اول مربوط به انتخاب الگوریتم مناسب و تنظیم فراپارامترهای آن است .سوال دوم مربوط به چالش عدم تعادل کلاس ها است .سوال سوم مربوط به استفاده از شاخص‌های ارزیابی مختلف است.

به طور کلی از هدف ما در انجام این پروژه می توان به موارد زیر اشاره کرد :

- بررسی عملکرد الگوریتم‌های مختلف طبقه‌بندی در شرایط مختلف
- بررسی تأثیر چالش عدم تعادل کلاس ها بر عملکرد طبقه‌بندان
- مقایسه عملکرد شاخص‌های ارزیابی مختلف

سوال 1

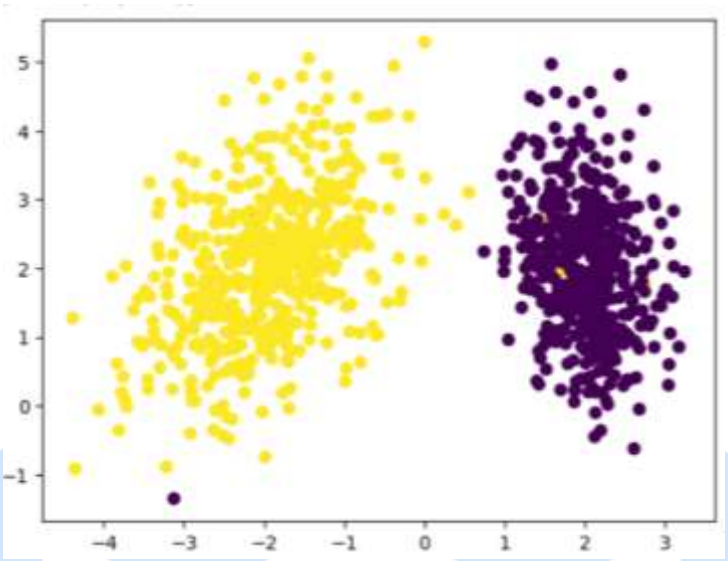
۱. با استفاده از sklearn.datasets یک دیتاست با ۱۰۰۰ نمونه، ۲ کلاس و ۲ ویژگی تولید کنید.

```
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
```

در ابتدا کتابخانه های مورد نظر خود را ایمپورت میکنیم. سپس با استفاده از دستور make_classification اطلاعات داده های مورد نظرمان را وارد می کنیم تا با توجه به ویژگی های داده شده دیتاست ما را تولید کند :

```
X , y = make_classification(n_samples = 1000 , n_features = 2 , n_redundant = 0, n_clusters_per_class= 1 , class_sep = 2 , random_state= 93 , n_classes = 2 )
plt.scatter(X[: , 0] , X[: , 1] , c = y)
X.shape , y.shape
```

در کد بالا دقت کنید که n_sample همان تعداد کل دیتاست های ما می باشد که طبق صورت سوال آن را برابر با 1000 قرار داده ایم. در ادامه n_feature نیز تعداد ویژگی های ما می باشد که آن را هم مطابق چیزی گفته شده است باید برابر با 2 قرار دهیم. n_classes تعداد کلاس های ما می باشد که آن را نیز برابر با 2 می گذاریم. n_clusters_per_classe تعداد خوشه ها در هر کلاس را نشان می دهد که مقدار آن را برابر با 1 قرار می دهیم. این بدین معنا می باشد که خوشه ها با یکدیگر همپوشانی ندارند. n_redundant تعداد ویژگی های اضافی را بیان می کند که با توجه به عدم خواسته سوال مقدار آن را برابر با صفر قرار می دهیم. و در نهایت نیز مقدار random_state را برابر با دو رقم آخر شماره دانشجویی قرار می دهیم. مقدار این مورد خیلی اهمیت ندارد اما ثابت بود آن باعث تکرار پذیری دیتاهای ما می شود. در ادامه دو ستون دارای ویژگی هامان را رسم میکنیم.



هر چقدر عدد class_sep کوچکتر باشد در هم امیختگی داده های ما بیشتر است. در کد بالا c = y برای تمایز میان داده های دو کلاس نوشته شده است. در غیر این صورت داده های دو کلاس با یک رنگ نمایش داده می شد که قابل تشخیص از یکدیگر نبوده است. در این مورد اطلاعات کافی در مورد هر یک از کلاس ها به طور دقیق گفته نشده است و تنها تعداد آن ها را مشخص کرده ایم.

۲. با استفاده از حداقل دو طبقه بند آمادهٔ پایتون و در نظر گرفتن فراپارامترهای مناسب، دو کلاس موجود در دیتاست ضمن توضیح روند انتخاب فراپارامترها (مانند تعداد دورهٔ آموزش و نرخ قسمت قبلی را از هم تفکیک کنید. یادگیری) ، نتیجهٔ دقت آموزش و ارزیابی را نمایش دهید. برای بهبود نتیجه از چه تکنیکهایی استفاده کردید؟

```
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
```

```
((800, 2), (200, 2), (800,), (200,))
```

ما از دو روش logisticregression و sgdclassifier استفاده میکنیم. برای هر دو روش نیاز داریم تا در ابتدا داده هارا به دو دسته ترین و تست تقسیم کنیم که اینکار را با نسبت 80 به 20 انجام میدهیم و در نهایت شکل انهارا با دستور shape. چک میکنیم.

```
model = LogisticRegression()
model.fit(x_train , y_train)
model.predict(x_test) , y_test
```

مدل خود را به صورت logisticregression تعریف میکنیم و انرا با دستور fit با داده های ترین اموزش میدهیم و سپس از مدل اموزش دیده میخواهیم تا داده های x_test را برای ما پیش بینی کند. و انهارا نمایش میدهیم.

```
(array([1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0,
        1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
        1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1,
        1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,
        0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
        0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1,
        0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1,
        1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
        0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,
        1, 1]),
 array([1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0,
        1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
        1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1,
        1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,
        0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
        0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1,
        0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1,
        1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
        0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,
        1, 1]))
```

مقادیر پیش بینی شده را به y_hat1 اختصاص میدهیم. و سپس دقت مدل را بررسی میکنیم.

```
y_hat1 = model.predict(x_test)
```

```
model.score(x_train , y_train)
```

```
0.9925
```

```
[38] model.score(x_test , y_test)
```

```
0.99
```

مدل بعدی مورد استفاده ما sgdc classifier است. دوباره همان داده های ترین و تست را به مدل مد نظر میدهیم. و انرا بر همان اساس اموزش میدهیم.

```
model1 = SGDClassifier(loss = "log_loss" , random_state = 98)
model1.fit(x_train , y_train)
```

```
SGDClassifier(loss='log_loss', random_state=98)
```

پیش بینی مدل خود را به y_hat2 اختصاص داده و انهارا نمایش میدهیم و در ادامه به بررسی دقت اموزش و تست میپردازیم.

```
y_hat2 = model1.predict(x_test)
```

```
y_hat2 , y_test
```

```
(array([0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0,
        0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0,
        0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
        1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
        0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1,
        0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
        1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1,
        1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
        1, 0]),
 array([0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0,
        0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
        1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0,
        0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
        1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
        0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
        0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
        1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1,
        1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
        1, 0]))
```

```
model1.score(x_train , y_train)
```

```
0.99125
```

```
model1.score(x_test , y_test)
```

```
0.99
```

برای بهبود نتایج، می توانیم تکنیک های زیر را در نظر بگیریم:

Hyperparameter Tuning : با مقادیر مختلف هایپرپارامترها آزمایش می کنیم تا ترکیب بهینه را پیدا کنیم. می توانیم از تکنیک هایی مانند جستجوی شبکه ای یا جستجوی تصادفی استفاده کنیم.

Feature Engineering: اگر مجموعه داده اجازه می دهد، سعی می کنیم ویژگی های جدید ایجاد کنیم یا ویژگی های موجود را تغییر دهیم تا اطلاعات بیشتری به مدل ارائه دهیم.

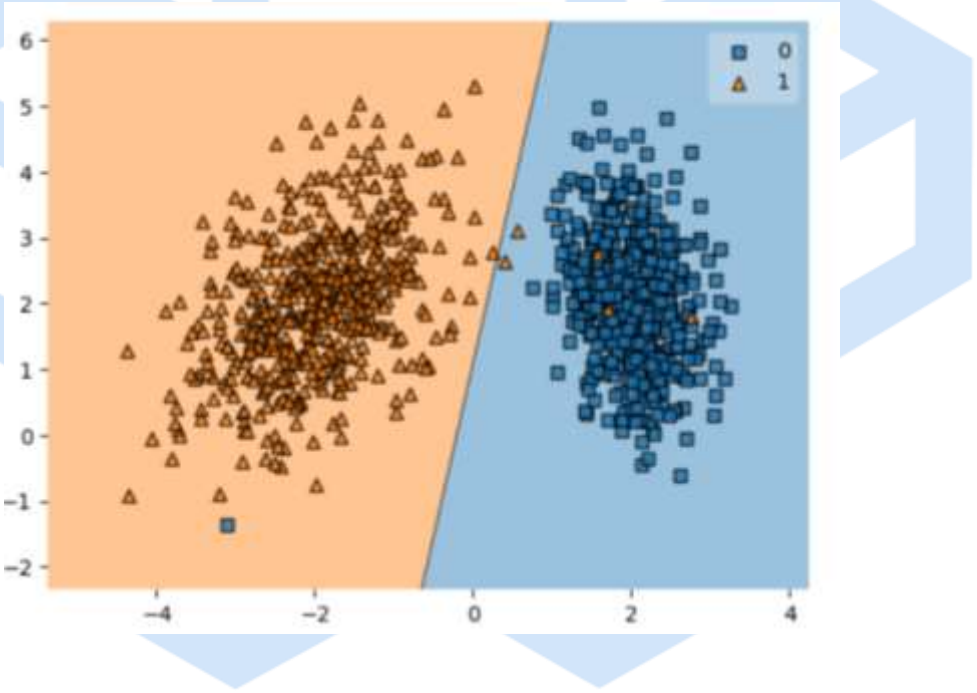
Ensemble Methods: چندین مدل را برای ایجاد یک مجموعه ترکیب می کنیم. این اغلب می تواند عملکرد را با کاهش بیش از حد برازش یا گرفتن الگوهای مختلف در داده ها بهبود بخشد.

Cross-Validation: از اعتبارسنجی متقاطع برای به دست آوردن تخمین بهتری از عملکرد مدل استفاده می کنیم. این کمک می کند تا اطمینان حاصل شود که مدل به خوبی به داده های دیده نشده تعمیم می یابد.

۳. مرز و نواحی تصمیم گیری برآمده از مدل آموزش دیدهٔ خود را به همراه نمونه ها در یک نمودار نشان دهید. اگر میتوانید نمونه هایی که اشتباه طبقه بندی شده اند را با شکل متفاوت نمایش دهید.

```
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(X , y , clf = model)
```

از روش رسم اماده ی scikit learn استفاده میکنیم در ابتدا کتابخانه ی مورد نظر را ایمپورت کرده و سپس مدل را به پلات میدهیم تا رسم کند.

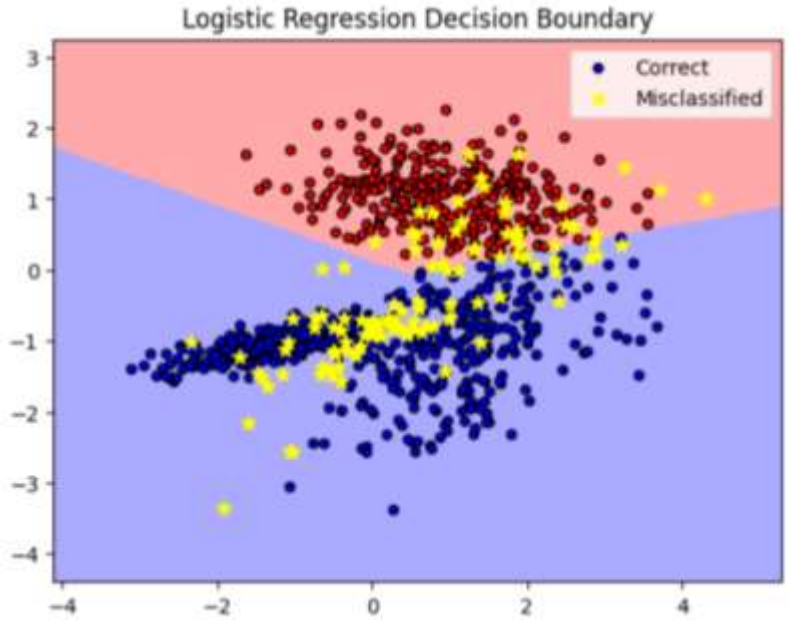


برای نمایش متفاوت داده های اشتباه طبقه بندی شده از کد زیر استفاده میکنیم در اینجا برای نمایش ملموس تر class_sep را از 2 به 1 تغییر دادیم دیتاهای بخش های بالاتر بر اساس class_sep 2 بود:

```
# Function to plot decision boundaries and areas
import numpy as np
def plot_decision_boundary(model, X_test, y_test, title):
    h = .02  # Step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

    # Plot the decision boundary
    x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
    y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 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)

    # Plot the training points
    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='*', color='yellow', s=50, label='Misclassified')

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

# Plot decision boundary for Logistic Regression
plot_decision_boundary(model,X, y, 'Logistic Regression Decision Boundary')
```

آرایه های correct_predictions و incorrect_predictions برای جداسازی نقاط صحیح و اشتباه طبقه بندی شده استفاده می شوند.

marker='o'برای نقاط صحیح استفاده می شود و marker='*' برای نقاط اشتباه طبقه بندی شده در تابع scatter استفاده می شود.
در نهایت شکل نهایی به صورت زیر خواهد بود :



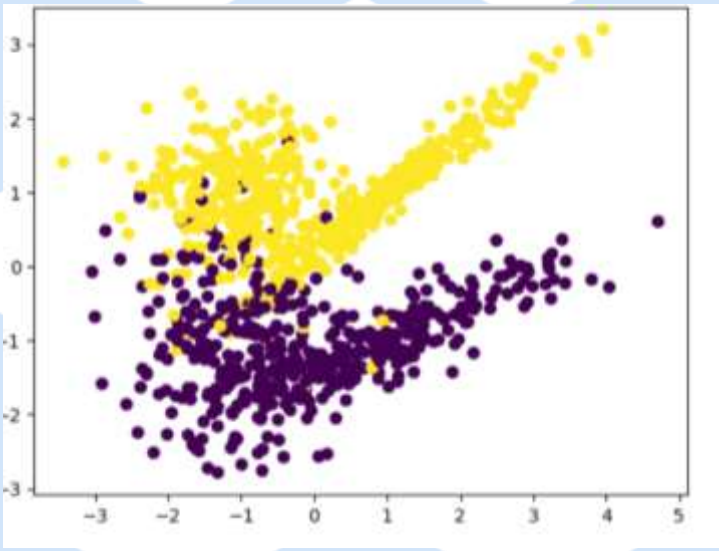
۴. از چه طریقی میتوان دیتاست تولید شده در قسمت »۱« را چالش برانگیزتر و سخت تر کرد؟ این کار را انجام داده و قسمتهای «۲» و «۳ »را برای این داده های جدید تکرار و نتایج را مقایسه کنید.

یکی از مهم ترین روش هایی که می تواند شرایط را برای ما در طبقه بندی کردن داده ها به مشکل بیاندازد این است که مقدار عددی پارامتر n_clusters_per_class را افزایش دهیم. این مورد باعث می شود تا داده های دو کلاس بیشتر در هم قاطی شوند و اختلاط بیش از حد آن ها می تواند کار classification را سخت تر کند. برای این که این موضوع را بهتر متوجه شویم می توانیم آن را به صورت زیر نمایش دهیم :

```
X , y = make_classification(n_samples = 1000 , n_features = 2 , n_redundant = 0, n_clusters_per_class= 2 , class_sep = 1 , random_state= 93 , n_classes = 2 )
plt.scatter(X[: , 0] , X[: , 1] , c = y)
X.shape , y.shape
```

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

دیتا ها به صورت زیر درمیاید و دقیقا همان متد قبل را اجرا میکنیم نتایج و کد ها به صورت زیرند:




```
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
```

```
((800, 2), (200, 2), (800,), (200,))
```

```
model = LogisticRegression()
model.fit(x_train , y_train)
model.predict(x_test) , y_test
```

```
(array([1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
        0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
        0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
        1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1,
        1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
        0, 1]),
 array([1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
        0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
        0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
        0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1,
        0, 1]))
```

```
167] y_hat1 = model.predict(x_test)
```

```
168] model.score(x_train , y_train)
```

```
0.91875
```

```
169] model.score(x_test , y_test)
```

```
0.955
```

```
model1 = SGDClassifier(loss = "log_loss" , random_state = 98)
model1.fit(x_train , y_train)
```

```
SGDClassifier
SGDClassifier(loss='log_loss', random_state=98)
```

```
y_hat2 = model1.predict(x_test)
```

```
y_hat2 , y_test
```

```
model1.score(x_train , y_train)
```

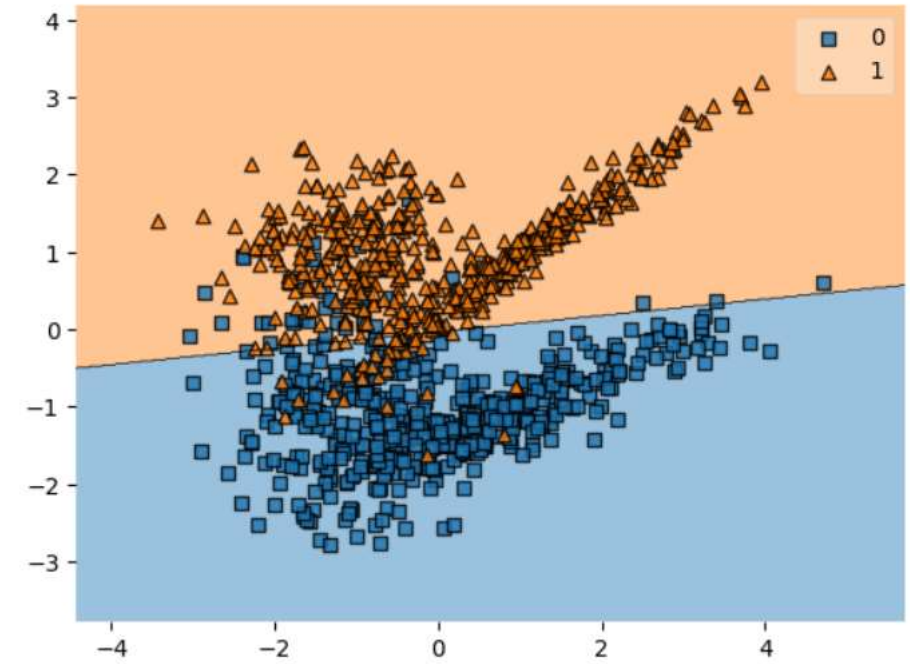
```
0.9225
```

```
model1.score(x_test , y_test)
```

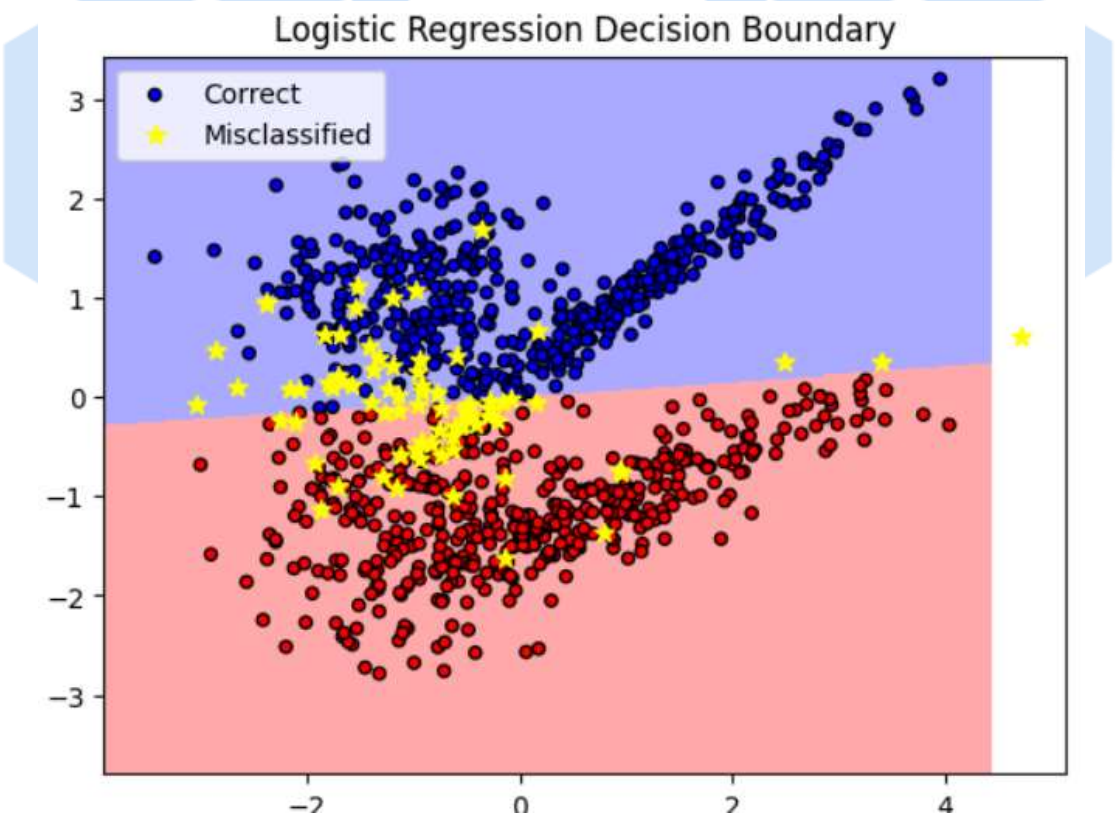
```
0.955
```

```
from mlxtend.plotting import plot_decision_regions


plot_decision_regions(X , y , clf = model)
```

```
<Axes: >
```

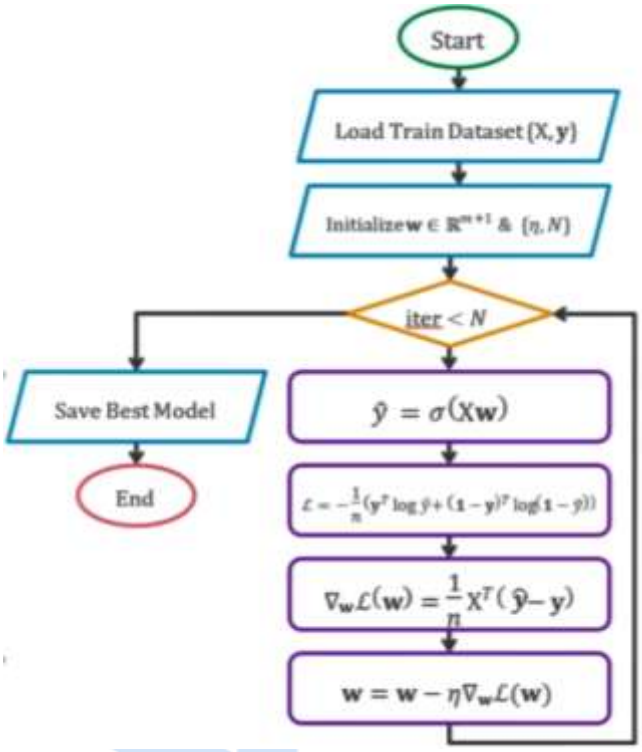


```python
# Function to plot decision boundaries and areas
import numpy as np
def plot_decision_boundary(model, x_test, y_test, title):
    h = .02  # Step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

    # Plot the decision boundary
    x_min, x_max = x_test[:, 0].min() - 1, x_test[:, 0].max() + 1
    y_min, y_max = x_test[:, 1].min() - 1, x_test[:, 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)

    # Plot the training points
    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='*', color='yellow', s=50, label='Misclassified')

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

# Plot decision boundary for Logistic Regression
plot_decision_boundary(model,x_test, y_test, 'Logistic Regression Decision Boundary')
```



۵. اگر یک کلاس به داده های تولید شده در قسمت «۱» اضافه شود، در کدام قسمتها از بلوک دیاگرام آموزش و ارزیابی تغییراتی ایجاد میشود؟ در مورد این تغییرات توضیح دهید. آیا میتوانید در این حالت پیاده سازی را به راحتی و با استفاده از کتابخانه ها و کدهای آمادهٔ پایتونی انجام دهید؟ پیاده سازی کنید.

همان طور که می دانیم الگوریتم آموزش طراحی مشخصی دارد و می توانیم هر قسمت آن را تغییر دهیم و تغییر در هر قسمت آن می تواند تغییراتی در ارزیابی سیستم ما ایجاد کند. بلوک دیاگرام آن به صورت زیر خواهد بود :



تولید داده:

هنگام تولید مجموعه داده، باید تعداد کلاس ها (n_classes) را مشخص کنیم و مطمئن شویم که در این مورد روی 3 تنظیم شده است.

تعریف مدل:

اگر از طبقه‌بندی‌کننده‌ای استفاده می کنیم که از طبقه‌بندی چند کلاسه پشتیبانی می‌کند (به عنوان مثال، رگرسیون لجستیک و غیره)، ممکن است نیازی به ایجاد تغییرات مهم نداشته باشیم. با این حال، اگر از یک طبقه‌بندی‌کننده باینری استفاده می کنیم، باید به یک طبقه‌بندی کننده چند کلاسه مانند LogisticRegression با پارامتر multi_class='multinomial'برویم.

آموزش:

هنگام تقسیم مجموعه داده و آموزش مدل، اطمینان حاصل کنیم که مدل با طبقه بندی چند کلاسه سازگار است. اگر از scikit-learn استفاده می کنیم، بسیاری از طبقه‌بندی‌کننده‌ها به‌طور خودکار دسته‌بندی چند کلاسه را انجام می‌دهند.

ارزیابی:

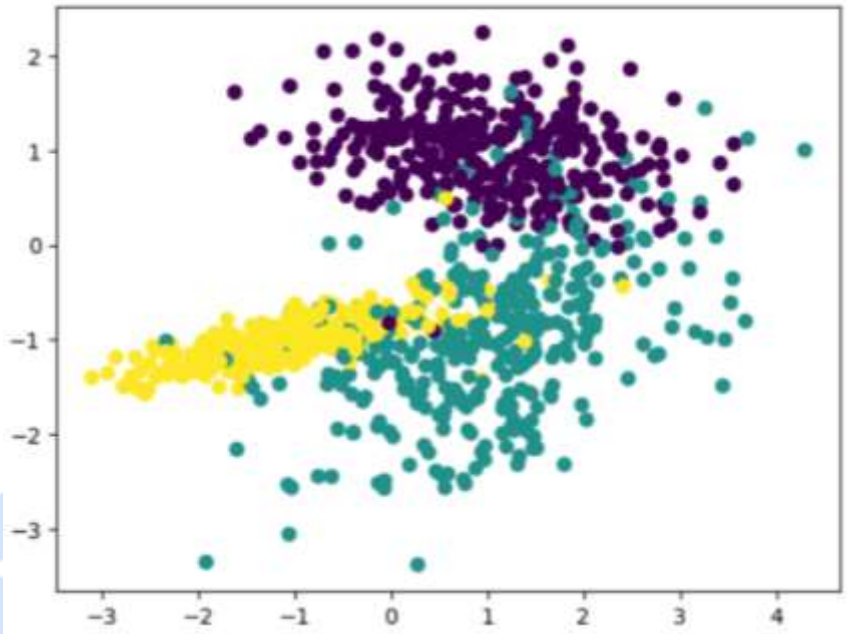
برای در نظر گرفتن کلاس جدید، معیارهای ارزیابی را به روز می کنیم. به عنوان مثال، دقت، برای هر سه کلاس محاسبه می شود.

ادامه کار دقیقا به همان صورتی است که در قسمت های دیده شد. فقط در اینجا با سه کلاس سر و کار داریم و باید از الگوریتم هایی استفاده کنیم که توانایی طبقه بندی سه کلاس را داشته باشند.

در اینجا یک نمونه پیاده سازی با استفاده از کتابخانه scikit-learn آورده شده است. در این مورد، من از LogisticRegression با پارامتر multi_class='multinomial' استفاده می کنم:

در ابتدا باید دیتا ها را با همان ساختاری که در قسمت یک خواسته شده بود اما این بار با سه کلاس تولید کنیم. پس به صورت زیر پیاده سازی می کنیم عدد n_class را برابر با 3 قرار میدهیم و باقی مراحل را تکرار میکنیم:

```
X , y = make_classification(n_samples = 1000 , n_features = 2 , n_redundant = 0, n_clusters_per_class= 1 , class_sep = 1 , random_state= 93 , n_classes = 3 )
plt.scatter(X[: , 0] , X[: , 1] , c = y)
X.shape , y.shape
```




```
] 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
```

```
((800, 2), (200, 2), (800,), (200,))
```

```
model = LogisticRegression()
model.fit(x_train , y_train)
model.predict(x_test) , y_test
```

```
(array([2, 0, 0, 1, 2, 0, 0, 0, 1, 1, 0, 2, 2, 0, 1, 0, 1, 0, 1, 0, 0, 1,
        2, 0, 2, 0, 2, 0, 1, 0, 2, 2, 0, 2, 0, 2, 0, 1, 0, 0, 0, 0, 0, 2,
        0, 2, 1, 1, 0, 1, 0, 1, 2, 2, 0, 2, 2, 1, 1, 0, 0, 1, 1, 2, 0, 0,
        2, 1, 1, 2, 2, 1, 2, 0, 2, 1, 2, 0, 2, 2, 1, 1, 2, 1, 2, 1, 1, 0,
        1, 2, 0, 0, 2, 1, 1, 0, 0, 2, 1, 0, 0, 1, 1, 1, 2, 2, 0, 2, 2, 1,
        2, 0, 2, 0, 2, 1, 0, 0, 1, 0, 0, 2, 2, 0, 0, 1, 2, 2, 1, 2, 0, 0,
        2, 2, 1, 2, 1, 0, 0, 1, 2, 2, 1, 1, 2, 0, 2, 2, 2, 1, 0, 2, 0, 1,
        1, 1, 1, 2, 0, 2, 2, 2, 2, 2, 0, 1, 0, 0, 1, 1, 0, 1, 1, 2, 0, 0,
        1, 1, 0, 1, 1, 2, 2, 2, 1, 0, 2, 2, 0, 1, 2, 0, 0, 1, 2, 2, 2, 2,
        1, 2]),
 array([1, 0, 0, 1, 2, 1, 1, 0, 1, 1, 0, 2, 2, 0, 1, 0, 2, 0, 1, 0, 0, 1,
        2, 0, 2, 0, 2, 0, 1, 0, 2, 2, 0, 2, 0, 2, 0, 1, 0, 0, 0, 0, 1, 2,
        1, 1, 1, 1, 0, 1, 1, 1, 2, 2, 0, 2, 2, 1, 1, 0, 0, 0, 1, 2, 1, 0,
        2, 1, 1, 2, 2, 1, 2, 0, 2, 1, 2, 0, 2, 2, 1, 1, 2, 1, 2, 1, 1, 0,
        1, 1, 0, 0, 1, 1, 1, 1, 0, 2, 1, 1, 0, 1, 1, 1, 2, 2, 0, 2, 2, 2,
        2, 0, 2, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0, 0, 1, 1, 2, 1, 2, 1, 0,
        2, 2, 1, 2, 1, 0, 0, 2, 2, 2, 1, 1, 2, 0, 2, 2, 2, 1, 0, 2, 0, 1,
        1, 1, 0, 2, 0, 2, 2, 2, 2, 2, 0, 1, 0, 0, 1, 1, 0, 1, 2, 2, 0, 1,
        1, 1, 0, 1, 1, 2, 2, 2, 1, 0, 2, 2, 0, 1, 2, 0, 0, 1, 2, 2, 2, 2,
        1, 2]))
```

```
y_hat1 = model.predict(x_test)
```

```
model.score(x_train , y_train)
```

```
0.89
```

```
model.score(x_test , y_test)
```

```
0.88
```

```
model1 = SGDClassifier(loss = "log_loss" , random_state = 98)
model1.fit(x_train , y_train)
```

```
SGDClassifier(loss='log_loss', random_state=98)
```

```
y_hat2 = model1.predict(x_test)
```

```
y_hat2 , y_test
```

```
(array([2, 0, 0, 1, 2, 0, 0, 0, 2, 1, 0, 2, 2, 0, 1, 0, 1, 0, 1, 0, 0, 2,
        2, 0, 2, 0, 2, 0, 1, 0, 2, 2, 0, 2, 0, 2, 0, 1, 0, 0, 0, 0, 0, 2,
        0, 2, 1, 1, 0, 1, 0, 1, 2, 2, 0, 2, 2, 1, 1, 0, 0, 1, 1, 2, 0, 0,
        2, 0, 1, 2, 2, 1, 2, 0, 2, 1, 2, 0, 2, 2, 0, 1, 2, 2, 2, 1, 1, 0,
        1, 2, 0, 0, 2, 1, 1, 0, 0, 2, 1, 0, 0, 1, 1, 1, 2, 2, 0, 2, 2, 1,
        2, 0, 2, 0, 2, 1, 0, 0, 1, 0, 0, 2, 2, 0, 0, 1, 2, 2, 1, 2, 0, 0,
        2, 2, 1, 2, 1, 0, 0, 2, 2, 2, 1, 1, 2, 0, 2, 2, 2, 1, 0, 2, 0, 2,
        2, 1, 1, 2, 0, 2, 2, 2, 2, 2, 0, 1, 0, 0, 1, 1, 0, 1, 1, 2, 0, 0,
        1, 1, 0, 1, 2, 2, 2, 2, 2, 0, 2, 2, 0, 1, 2, 0, 0, 1, 2, 2, 2, 2,
        1, 2]),
 array([1, 0, 0, 1, 2, 1, 1, 0, 1, 1, 0, 2, 2, 0, 1, 0, 2, 0, 1, 0, 0, 1,
        2, 0, 2, 0, 2, 0, 1, 0, 2, 2, 0, 2, 0, 2, 0, 1, 0, 0, 0, 0, 1, 2,
        1, 1, 1, 1, 0, 1, 1, 1, 2, 2, 0, 2, 2, 1, 1, 0, 0, 0, 1, 2, 1, 0,
        2, 1, 1, 2, 2, 1, 2, 0, 2, 1, 2, 0, 2, 2, 1, 1, 2, 1, 2, 1, 1, 0,
        1, 1, 0, 0, 1, 1, 1, 1, 0, 2, 1, 1, 0, 1, 1, 1, 2, 2, 0, 2, 2, 2,
        2, 0, 2, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0, 0, 1, 1, 2, 1, 2, 1, 0,
        2, 2, 1, 2, 1, 0, 0, 2, 2, 2, 1, 1, 2, 0, 2, 2, 2, 1, 0, 2, 0, 1,
        1, 1, 0, 2, 0, 2, 2, 2, 2, 2, 0, 1, 0, 0, 1, 1, 0, 1, 2, 2, 0, 1,
        1, 1, 0, 1, 1, 2, 2, 2, 1, 0, 2, 2, 0, 1, 2, 0, 0, 1, 2, 2, 2, 2,
        1, 2]))
```

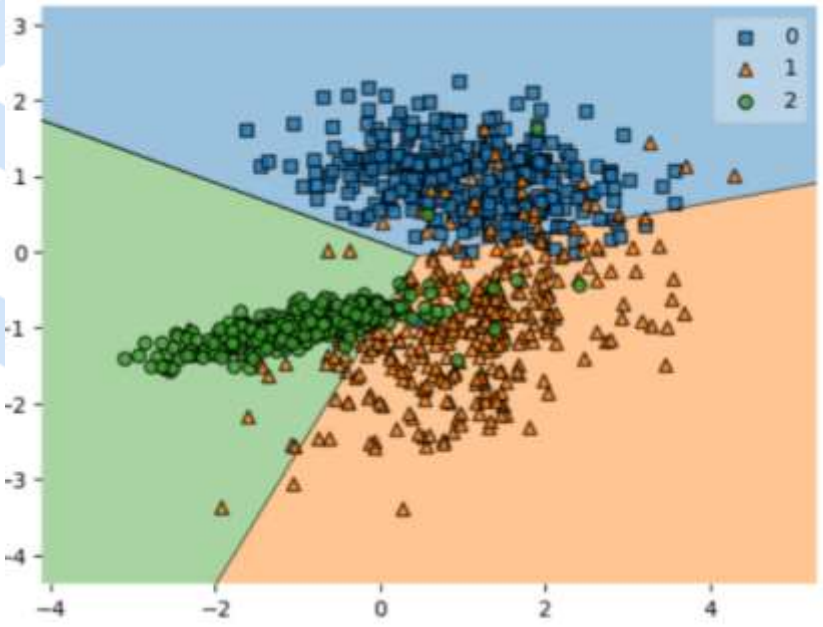
```
model1.score(x_train , y_train)
```

```
0.875
```

```
model1.score(x_test , y_test)
```

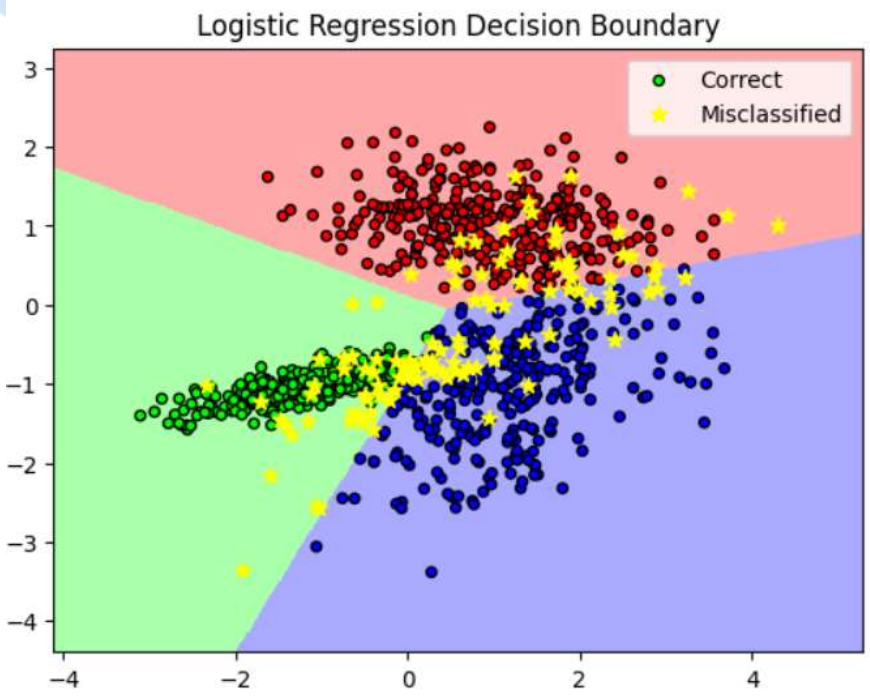
```
0.84
```

```
from mlxtend.plotting import plot_decision_regions


plot_decision_regions(X , y , clf = model)
```



```python
# Function to plot decision boundaries and areas
def plot_decision_boundary(model, X, y, title):
    h = .02  # Step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF', '#AAFFAA'])
    cmap_bold = ListedColormap(['#FF0000', '#0000FF', '#00FF00'])

    # Plot the decision boundary
    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)

    # Plot the training points
    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='*', color='yellow', s=50, label='Misclassified')

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

# Plot decision boundary for Logistic Regression
plot_decision_boundary(model, X, y, 'Logistic Regression Decision Boundary')
```

در این حالت دیگر نمی توانیم سه کلاس را با هر تعداد دیتایی از یکدیگر جدا کنیم. بنابراین این کار را کتابخانه های آماده پایتون به صورت خودکار انجام می دهند. شکل طبقه بندی شده به صورت روبرو نمایش داده می شود و مانند قبل باز داده هایی که از کلاس های دیگر در طبقه های دیگری به اشتباه قرار گرفته اند و دارای خطا می باشند را با علامت متفاوت نمایش می دهیم. در شکل هر رنگ نشان دهنده یک کلاس در دیتاست پایین می باشد :



سوال 2

۱ .با مراجعه به این پیوند با یک دیتاست مربوط به حوزهٔ «بانکی» آشنا شوید و ضمن توضیح کوتاه اهداف و ویژگیهایش، در محیط گوگل کولب قرار gdown فایل آن را دانلود کرده و پس از بارگذاری در گوگلدرایو خود، آن را با دستور دهید. اگر تغییر فرمتی برای فایل این دیتاست نیاز میبینید، این کار را با دستورهای پایتونی انجام دهید.

```
!gdown 1uYthYdsdveWJD7dLSQbAPxiy1bVEXhX0
```

```
Downloading...
From: https://drive.google.com/uc?id=1uYthYdsdveWJD7dLSQbAPxiy1bVEXhX0
To: /content/data_banknote_authentication.txt
100% 46.4k/46.4k [00:00<00:00, 47.7MB/s]
```

```
import pandas as pd
import numpy as np

read_file = pd.read_csv (r'/content/data_banknote_authentication.txt')
read_file.to_csv (r'/content/data_banknote_authentication.csv')
df = pd.read_csv("/content/data_banknote_authentication.csv")

df
```

	Unnamed: 0	3.6216	8.6661	-2.8073	-0.44699	0
0	0	4.54590	8.16740	-2.4586	-1.46210	0
1	1	3.86600	-2.63830	1.9242	0.10645	0
2	2	3.45660	9.52280	-4.0112	-3.59440	0
3	3	0.32924	-4.45520	4.5718	-0.98880	0
4	4	4.36840	9.67180	-3.9606	-3.16250	0
...	...	...	...	...	...	...
1366	1366	0.40614	1.34920	-1.4501	-0.55949	1
1367	1367	-1.38870	-4.87730	6.4774	0.34179	1

دیتا ست را از لینک مورد نظر دانود کرده و با دستور gdown و مراحل گفته شده در ویدیوی اموزشی کلاس در گوگل کولب اپلود میکنیم. فایل به فرمت txt اپلود میشود که به تغییر فرمت نیاز دارد با استفاده از دستورات بالا اول فایل را خوانده و سپس فرمت انرا تغییر میدهیم. به دلیل نداشتن اسم ستون، برای هر کدام نام مشخصی تعیین میکنیم تا در ادامه کار تسهیل شود.

```
import csv
from sklearn.utils import shuffle
# field names
# fields = ['index', 'price1', 'price2', 'price3' , 'price4' , 'price5']
# importing python package
import pandas as pd
# read contents of csv file
file = pd.read_csv("/content/data_banknote_authentication.csv")
# adding header
headerList = ['index','price1', 'price2', 'price3' , 'price4' , 'price5']
# converting data frame to csv
file.to_csv("/content/data_banknote_authentication.csv",
            header=headerList, index=False)
# display modified csv file
file2 = pd.read_csv("/content/data_banknote_authentication.csv")
print('\nModified file:')
```

محتوای فایل تغییر فرمت داده شده را در یک متغیر ریخته و برای ان با استفاده از headerlist نام های مد نظر را قرار میدهیم .

۲ .ضمن توضیح اهمیت فرآیند بر زدن (مخلوط کردن) دادهها را مخلوط کرده و با نسبت تقسیم دلخواه و معقول به دو بخش «آموزش» و «ارزیابی» تقسیم کنید.

اهمیت شافل کردن دیتا ها در یادگیری ماشین به شرح زیر است:

- جلوگیری از بروز الگوهای کاذب : شافل کردن دیتا ها باعث می شود که الگوریتم یادگیری ماشین الگوهای
- کاذبی را در دیتا ها تشخیص ندهد . این الگوهای کاذب می توانند ناشی از ترتیب خاصی از داده ها در
- مجموعه داده باشند.
- بهبود عملکرد الگوریتم : شافل کردن دیتا ها می تواند به بهبود عملکرد الگوریتم یادگیری ماشین کمک
- کند . این به این دلیل است که شافل کردن باعث می شود که الگوریتم به طور مساوی از تمام داده ها
- استفاده کند و از تأثیر داده های نادرست یا غیرعادی جلوگیری کند.
- افزایش سرعت یادگیری : شافل کردن دیتا ها می تواند به افزایش سرعت یادگیری الگوریتم یادگیری
- ماشین کمک کند . این به این دلیل است که شافل کردن باعث می شود که الگوریتم به طور مساوی از
- تمام داده ها استفاده کند و از تکرار داده ها جلوگیری کند.

```
file2=shuffle(file2)
print(file2)
```

با اضافه کردن کد دو خط بالا به ادامه ی کد داده های شافل شده و انها را مشاهده میکنیم(کتابخانه مربوط به شافل در عکس بالا در کد ایمپورت شده.)

```
Modified file:
      index   price1    price2    price3    price4  price5
463     463  5.74030  -0.44284   0.38015  1.376300       0
889     889 -1.66370   3.28810  -2.27010 -2.222400       1
143     143  3.84810  10.15390  -3.85610 -4.222800       0
937     937 -4.37730  -5.51670  10.93900 -0.408200       1
1359   1359 -0.24745   1.93680  -2.46970 -0.805180       1
...     ...      ...       ...       ...       ...     ...
700     700  5.59100  10.46430  -4.38390 -4.337900       0
482     482  0.96788   7.19070   1.27980 -2.456500       0
1276   1276  0.56232   1.00150  -2.27260 -0.006049       1
957     957 -0.36372   3.04390  -3.48160 -2.783600       1
646     646  3.77910   2.57620   1.30980  0.565500       0

[1371 rows x 6 columns]
```

```
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
```

```
[5] X = file2[["price1" ,"price2" , "price3" , "price4"]].values

    y = file2[["price5"]].values
    X ,y
```

```
(array([[ 3.5829  ,  1.4423  ,  1.0219  ,  1.4008  ],
        [ 1.6349  ,  3.286   ,  2.8753  ,  0.087054],
        [-2.565   , -5.7899  ,  6.0122  ,  0.046968],
        ...,
        [ 0.22432 , -0.52147 , -0.40386 ,  1.2017  ],
        [-1.2528  , 10.2036  ,  2.1787  , -5.6038  ],
        [ 1.2309  ,  3.8923  , -4.8277  , -4.0069  ]]),
 array([[0],
        [0],
        [1],
        ...,
        [1],
        [0],
        [1]]))
```

```
[6] 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
```

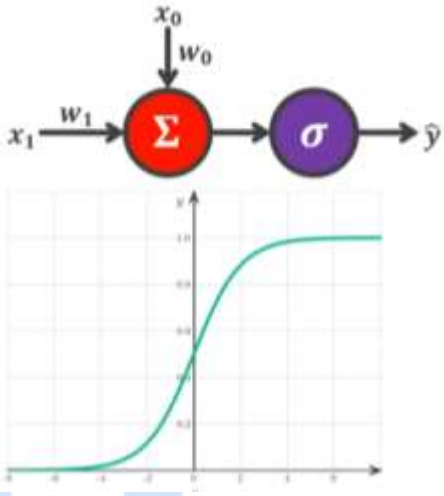
```
((1096, 4), (275, 4), (1096, 1), (275, 1))
```

4 خط اول مربوط مربوط به این بخش از سوال نیست و برای ادامه ی کار است. در ابتدا ستون هایی که شامل دیتا بوده و نامگذاری کردیم را به X داده و همین کار را برای ستون target برای y میکنیم.

داده هارا به بخش ترین و تست برای x و y ها تقسیم میکنیم و نسبت اموزش را 80 درصد و تست را 20 درصد میگذاریم.

۳. بدون استفاده از کتابخانه های آمادهٔ پایتون، مدل، تابع اتلاف و الگوریتم یادگیری و ارزیابی را کدنویسی کنید تا دو کلاس موجود در دیتاست به خوبی از یکدیگر تفکیک شوند. نمودار تابع اتلاف را رسم کنید و نتیجهٔ دقت ارزیابی روی دادههای تست را محاسبه کنید. نمودار تابع اتلاف را تحلیل کنید. آیا میتوان از روی نمودار تابع اتلاف و قبل از مرحلهٔ ارزیابی با قطعیت در مورد عملکرد مدل نظر داد؟ چرا و اگر نمیتوان، راه حل چیست؟

ابتدا به مرور مدل یادگیری مد نظر میپردازیم.



$$\mathbf{x} = [x_0 \quad x_1]^T$$
$$\mathbf{w} = [w_0 \quad w_1]^T$$
$$\boxed{\hat{y} = \sigma(\mathbf{w}^T\mathbf{x})}$$
$$\boxed{\sigma(x) = \frac{1}{1+e^{-x}}}$$

هدف اموزش مدل بالا به ماشین است. ایکس بخش ورودی و w مربوط به وزن های ماست سیگموید نیز به عنوان transfer function بکار رفته.

$$acc = \frac{\text{sum}(y == \text{round}(y_{hat}))}{\text{len}(y)}$$

$$L = \begin{cases} -\log(\hat{y}) & y = 1 \\ -\log(1-\hat{y}) & y = 0 \end{cases}$$
$$L = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$$

از فرمول های تصویر سمت راست برای تابع اتلاف و از تصویر سمت چپ برای سنجش دقت استفاده میکنیم.

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

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

```
[54] y_hat = logistic_regression(np.random.rand(5 ,3) ,np.random.rand(3 , 1))
     y_hat
```

```
array([[0.73978081],
       [0.66231037],
       [0.64742472],
       [0.63915085],
       [0.72236284]])
```

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

```
bce(np.ones((5 , 1)) , y_hat)
```

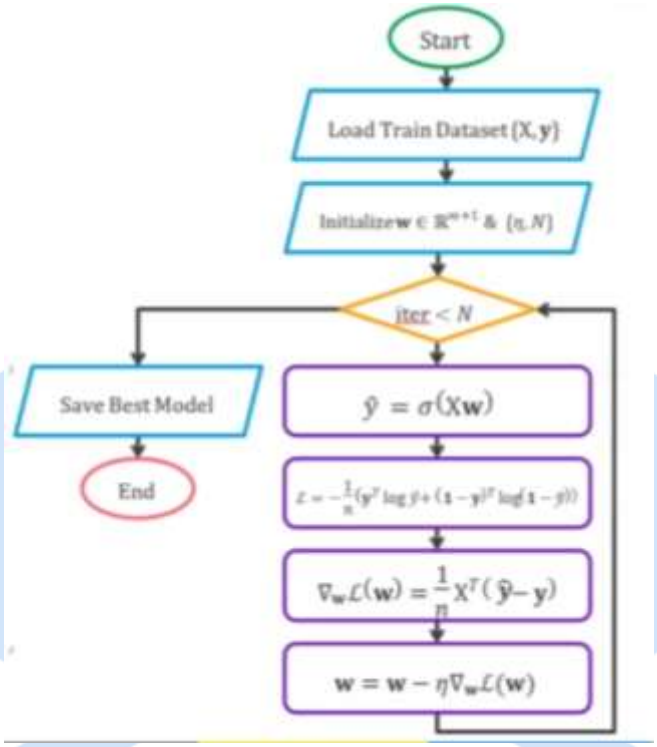
```
0.38420351871113617
```

تابع سیگموید را به کمک نامپای و فرمول های اورده شده تعریف میکنیم. سپس مدل خود را با نام logisticregression تعریف کرده و خروجی را تحت عنوان y_hat نام گذاری میکنیم. در نهایت تابع اتلاف خود را تعریف میکنیم و تمامی موارد گفته شده را تست میکنیم تا از درست کار کردن کد مطمئن شویم. در نظر داشته باشید به دلیل ضرب ماتریسی در بخش logisticregression به تعداد سطر و ستون ها دقت شود تا کد دچار ارور نشود.در اینجا از اعداد رندوم استفاده کردیم همچنین برای تابع اتلاف در حالت برداری به میانگین گیری نیاز است که اعمال شده است.

$$\mathbf{w} = \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \frac{1}{n} X^T (\hat{y} - \mathbf{y})$$

در ادامه به تعریف گرادیان (فرمول های شکل بالا) و استفاده از الگوریتم زیر میپردازیم.



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

```
[13] gradient(np.random.randn(5 , 2) , np.ones((5 , 1)) , y_hat)

     array([[ 0.07123339],
            [-0.15188796]])
```

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

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

```
[16] accuracy(np.array([1 , 0 , 1]) , np.array([0.7 , 0.5 , 0.3]))

     0.6666666666666666
```

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

     (1096, 5)
```

```
[18] m = 4
     w = np.random.randn(m+1 , 1)
     print(w.shape)
     eta = 0.01
     n_epochs = 2000

     (5, 1)
```

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

       e = bce(y_train , y_hat)
       error_hist.append(e)
       grads = gradient(x_train , y_train , y_hat)
       w = gradient_descent(w , eta , grads)
       if(epoch + 1) % 100 == 0:
         print(f"Epoch = {epoch} , \t E = {e:.4} \t w={w.T[0]}")
```

گرادیان، گرادیان نزولی و تابع محاسبه ی دقت را طبق فرمول های گفته شده تعریف میکنیم. M به تعداد فیچر های اولیه ما تعریف میشود که در ابتدا 4 تا بود در ادامه یک ستون به ان به دلیل داشتن بایاس در w اضافه میشود. این کار را برای x_train هم انجام میدهیم.

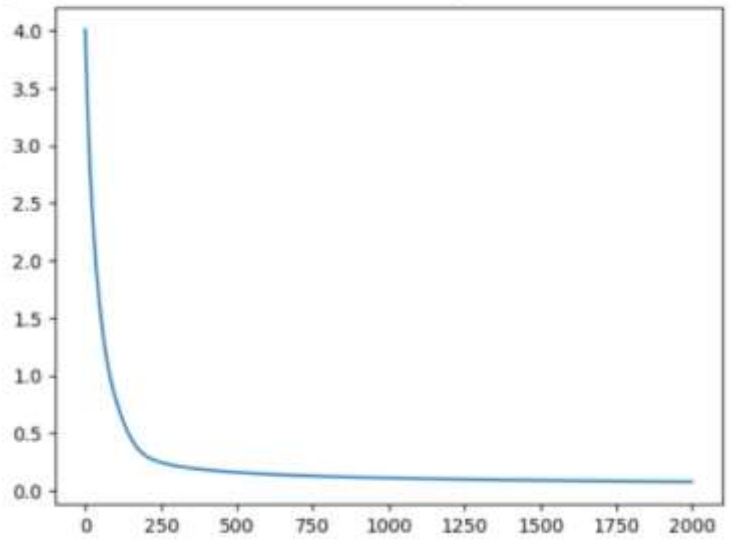
تعداد ایپاک ها و ضریب یادگیری برای مدل تعریف میکنیم. و در ادامه یک لیست خالی برای ارور ها به منظور ذخیره سازی آن در هر ایپاک درست میکنیم. سپس به تعداد ایپاک ها و با داده های ترین خود y_hat را حساب میکنیم.

سپس ارور را حساب کرده و به لیست آن اضافه میکنیم و سپس w هارا به کمک گرادیان نزولی تصحیح میکنیم. از خط بعدی برای نمایش ارور و ماتریس وزن استفاده شده است. در اینجا هر 100 ایپاک یک بار نتیجه را نمایش دادیم.

```
Epoch = 99 ,    E = 0.8081    w=[-0.39890031 -0.35166132 -0.37894875 -0.33466601  0.83423392]
Epoch = 199 ,   E = 0.3037    w=[-0.20271889 -0.79170238 -0.22896575 -0.26703812  0.37445421]
Epoch = 299 ,   E = 0.216     w=[-0.08046183 -0.94515066 -0.25859868 -0.30514623  0.17128197]
Epoch = 399 ,   E = 0.1807    w=[ 0.01402635 -1.02453996 -0.31899282 -0.3647368   0.05984929]
Epoch = 499 ,   E = 0.1588    w=[ 0.09570547 -1.07963574 -0.37656051 -0.42101524 -0.01424108]
Epoch = 599 ,   E = 0.1436    w=[ 0.16908863 -1.12298498 -0.42737758 -0.47098221 -0.06742964]
Epoch = 699 ,   E = 0.1322    w=[ 0.23629609 -1.1594489  -0.47187659 -0.51526154 -0.10725612]
Epoch = 799 ,   E = 0.1232    w=[ 0.2985888  -1.19137746 -0.51109865 -0.55487077 -0.13790804]
Epoch = 899 ,   E = 0.1159    w=[ 0.35681358 -1.22007326 -0.5460034  -0.59067907 -0.16196246]
Epoch = 999 ,   E = 0.1098    w=[ 0.41158068 -1.24632998 -0.57736608 -0.62336103 -0.18111156]
Epoch = 1099 ,  E = 0.1045    w=[ 0.46335099 -1.27066642 -0.60579431 -0.65343361 -0.19651895]
Epoch = 1199 ,  E = 0.1       w=[ 0.51248473 -1.29344078 -0.63176352 -0.6812967  -0.20901343]
Epoch = 1299 ,  E = 0.09599   w=[ 0.55927111 -1.31491144 -0.65564865 -0.70726438 -0.21920223]
Epoch = 1399 ,  E = 0.09243   w=[ 0.6039473  -1.33527161 -0.67774866 -0.73158745 -0.22754061]
Epoch = 1499 ,  E = 0.08923   w=[ 0.64671132 -1.35467014 -0.69830486 -0.75446923 -0.23437658]
Epoch = 1599 ,  E = 0.08633   w=[ 0.68773094 -1.37322452 -0.71751425 -0.77607693 -0.23998054]
Epoch = 1699 ,  E = 0.08369   w=[ 0.72715008 -1.39102954 -0.7355395  -0.79654982 -0.24456562]
Epoch = 1799 ,  E = 0.08128   w=[ 0.76509356 -1.40816299 -0.75251629 -0.8160052  -0.24830186]
Epoch = 1899 ,  E = 0.07905   w=[ 0.80167062 -1.42468973 -0.76855889 -0.83454285 -0.2513264 ]
Epoch = 1999 ,  E = 0.077     w=[ 0.83697764 -1.44066456 -0.78376432 -0.85224844 -0.2537509 ]
```

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

```
[<matplotlib.lines.Line2D at 0x7eab38cd2890>]
```



و تابع اتلاف را رسم میکنیم.

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

```
0.9781818181818182
```

در ادامه دقت را محاسبه میکنیم.

همان طور که می توان فهمید با زیاد شدن ایپاک ها میزان خطای الگوریتم استفاده شده در این قسمت کاهش می یابد. می توانیم تعداد ایپاک ها را زیاد یا کم کنیم ولی ممکن است الگوریتمی که در این قسمت استفاده کرده ایم دچار under modeling و یا over modelingشود . تا به اینجای داده ها را در پایتون import کردیم

و با توجه به فیلم درس الگوریتم را به صورت دستی وارد کرده ایم. در این قسمت در ابتدا داده ها را مخلوط کردیم

و پس از مخلوط کردن داده ها را به بخش های ارزیابی و آموزش تقسیم بندی کرده ایم. سپس با استفاده از الگوریتم گرادیان نزولی داده ها را تفکیک کرده ایم و دو کلاس را از یکدیگر جدا کرده ایم. در نهایت هم نمودار تابع اتلاف را نمایش داده ایم.

در حالی که نمودار تابع ضرر در طول آموزش می تواند بینش ارزشمندی در مورد همگرایی و پویایی یادگیری مدل شما ارائه دهد، اما همیشه برای نتیجه گیری قطعی در مورد عملکرد آن کافی نیست. در اینجا به چند دلیل اشاره میکنیم:

- تطبیق بیش از حد و تعمیم: یک مدل ممکن است در مجموعه آموزشی عملکرد خوبی داشته باشد)که منجر به کاهش تلفات آموزشی می شود(، اما آزمون واقعی آن با ارزیابی در یک اعتبارسنجی یا مجموعه تست جداگانه همراه است. تطبیق بیش از حد زمانی اتفاق میافتد که یک مدل داده های آموزشی را به خوبی یاد میگیرد، از جمله نویز و نقاط پرت آن، اما نمیتواند به داده های جدید و دیده نشده تعمیم یابد. بررسی ضرر در یک مجموعه اعتبار سنجی جداگانه برای ارزیابی عملکرد تعمیم بسیار مهم است.

معیارهای ارزیابی: علاوه بر تابع ضرر، سایر معیارهای ارزیابی مانند دقت، غیره برای درک جامع عملکرد مدل ضروری هستند. تابع ضرر ممکن است تمام جنبه های رفتار مدل را در بر نگیرد، بهویژه زمانی که با مجموعه دادهه ای نامتعادل یا الزامات خاص سروکار داریم.

- مشکلات نرخ یادگیری: نرخ یادگیری می تواند به طور قابل توجهی بر همگرایی مدل شما تأثیر بگذارد. گاهی اوقات، منحنی ضرر ممکن است به دلیل نرخ یادگیری خیلی بالا یا خیلی پایین، رفتار نامنظمی از خود نشان دهد. تنظیم نرخ یادگیری در طول آموزش یا استفاده از زمان بندی نرخ یادگیری می تواند کمک کننده باشد.
- دوران و توقف اولیه: منحنی ضرر ممکن است در چند دوره تثبیت نشود. نظارت بر منحنی در تعداد مناسبی از دوره ها ضروری است. علاوه بر این، استفاده از تکنیکهایی مانند توقف زود هنگام میتواند از تطبیق بیش از حد جلوگیری کرده و منابع محاسباتی را ذخیره کند.
- کیفیت داده و پیش پردازش: کیفیت داده های شما و اثربخشی مراحل پیش پردازش می تواند بر عملکرد مدل تأثیر بگذارد. تجسم توزیع داده ها و درک تأثیر پیش پردازش بر روی منحنی ضرر می تواند آموزنده باشد.

برای ارزیابی آگاهانه تر از عملکرد مدل خود، توصیه میشود:

- ارزیابی بر روی یک مجموعه داده جداگانه: از یک مجموعه اعتبارسنجی یا یک مجموعه آزمایشی استفاده کنید که مدل در طول آموزش ندیده است.
- نظارت بر معیارهای چندگانه: بسته به ماهیت کار خود، معیارهای ارزیابی مختلفی را در نظر بگیرید.

- اجرای اعتبارسنجی متقابل: در صورت امکان، از تکنیک هایی مانند اعتبار سنجی متقابل برای ارزیابی استحکام مدل در زیر مجموعه های مختلف داده ها استفاده کنید.

۴. حداقل دو روش برای نرمالسازی داده ها را با ذکر اهمیت این فرآیند توضیح دهید و با استفاده از یکی از این روشها، داده ها را نرمال کنید. آیا از اطلاعات بخش «ارزیابی» در فرآیند نرمالسازی استفاده کردید؟ چرا؟

به طور کلی روش های زیادی برای نرمال سازی داده ها داریم که در این قسمت می خواهیم به دو روش از آن ها اشاره کنیم:

نرمالسازی داده ها یک مرحله مهم در پردازش و تحلیل دادههاست 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، دادهها به گونهای تغییر میکنند که حداقل و حداکثر آنها به ترتیب به یک مقدار نگاشته میشوند. فرمول نرمالسازی Min-Max برای یک داده X به صورت زیر است :

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

Standardization score-Z: در این روش، دادهها به گونهای تغییر میکنند که میانگین آنها صفر و انحراف معیاری آنها یک شود. فرم نرمالسازی Z-score برای یک داده X به صورت زیر است:

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

اطلاعات بخش "ارزیابی" در فرآیند نرمالسازی به تنهایی معمولاً استفاده نمیشود. بخش ارزیابی معمولاً برای ارزیابی عملکرد مدل یا سیستم پس از اعمال تغییرات)مانند نرمالسازی(استفاده میشود. انتخاب یک روش نرمالسازی باید بر اساس نیازها و خصوصیات دادهها انجام شود. اگر توزیع دادهها نسبت به هم مهم است، ممکن است از Z-score Standardization استفاده کنید. اگر میخواهید دادهها را به یک بازه خاص نگاشت کنید، Min-Max Scalingمناسب تر است.

در مورد بخش "ارزیابی" در فرآیند عادی سازی، بستگی به زمینه دارد. اگر بخش ارزیابی حاوی اطلاعاتی در مورد دامنه و توزیع ویژگی ها باشد، می تواند در انتخاب روش نرمال سازی مناسب سودمند باشد. به عنوان مثال، اگر ویژگی ها دارای نقاط پرت باشند، عادی سازی امتیاز Z ممکن است قوی تر باشد. اگر ویژگیها محدوده مشخصی دارند، ممکن است مقیاسبندی Min-Max ترجیح داده شود. درک ویژگی های داده ها و انتخاب روش عادی سازی بر این اساس ضروری است.

در اینجا از روش اول استفاده شده است.

```
# find maximum & minimum values of a list of columns
maxValues = file2[['price1', 'price2' , 'price3' , 'price4']].max()

print("Maximum value in column 'price1', 'price2' , 'price3' , 'price4': ")
print(maxValues)

minValues = file2[['price1', 'price2' , 'price3' , 'price4']].min()
print("Minimum value in column 'price1', 'price2' , 'price3' , 'price4': ")
print(minValues)
```

```
Maximum value in column 'price1', 'price2' , 'price3' , 'price4':
price1     6.8248
price2    12.9516
price3    17.9274
price4     2.4495
dtype: float64
Minimum value in column 'price1', 'price2' , 'price3' , 'price4':
price1    -7.0421
price2   -13.7731
price3    -5.2861
price4    -8.5482
dtype: float64
```

در ابتدا مقدار مینیمم و ماکسیمم داده های هر ستون را پیدا میکنیم.

```
[ ]  #bakhshe 4 standard sazi ravesh 1
     for i in range(4):
       file2[f'price{i+1}'] = (file2[f'price{i+1}']-minValues[i])/(maxValues[i]- minValues[i])
       print(file2[f'price{i+1}'])
```

سپس با نوشتن یک حلقه نرمالسازی برای تمام داده های همه ستون هارا انجام میدهیم. و انهارا چک میکنیم.

```
X = file2[["price1" ,"price2" , "price3" , "price4"]].values

y = file2[["price5"]].values
X ,y
```

```
(array([[0.5310776 , 0.48176481, 0.5761475 , 0.84021386],
        [0.15125226, 0.63886966, 0.27350895, 0.64597143],
        [0.44169353, 0.8891213 , 0.27856635, 0.30205407],
        ...,
        [0.33404005, 0.6554386 , 0.21039266, 0.65949244],
        [0.63919838, 0.27309193, 0.57455791, 0.73921365],
        [0.47757249, 0.62423526, 0.15078295, 0.57660238]]),
 array([[0],
        [1],
        [0],
        ...,
        [1],
        [0],
        [1]]))
```

مشاهده میشود تمامی داده ها نرمال سازی شدند.

۵. تمام قسمتهای »۱« تا »۳« را با استفاده از داده های نرمال شده تکرار کنید و نتایج پیشبینی مدل را برای پنج نمونه داده نشان دهید.

```
[ ]  m = 4
     w = np.random.randn(m+1 , 1)
     print(w.shape)
     eta = 0.01
     n_epochs = 4000
```

```
(5, 1)
```

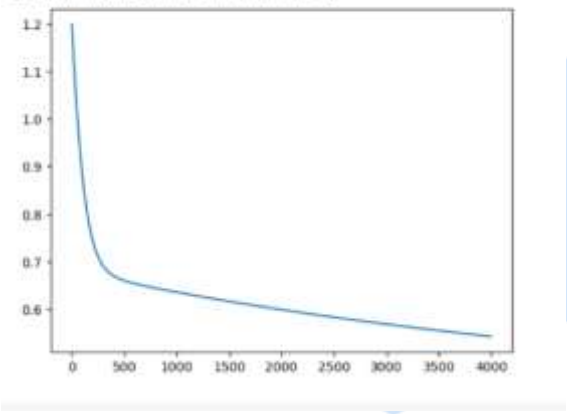
```
[ ]  error_hist = []
     for epoch in range(n_epochs):
       y_hat = logistic_regression(x_train , w)

       e = bce(y_train , y_hat)
       error_hist.append(e)
       grads = gradient(x_train , y_train , y_hat)
       w = gradient_descent(w , eta , grads)
       if(epoch + 1) % 100 == 0:
         print(f"Epoch - {epoch} , \t E - {e:.4} \t w={w.T[0]}")
```

دوباره فرایند اموزش را تکرار میکنیم و دقت را بررسی میکنیم.

5 داده اول از هرکدام نشان داده شده است.

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

```
[<matplotlib.lines.Line2D at 0x7eab1b8bcfa0>]
```




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

```
0.7054545454545454
```

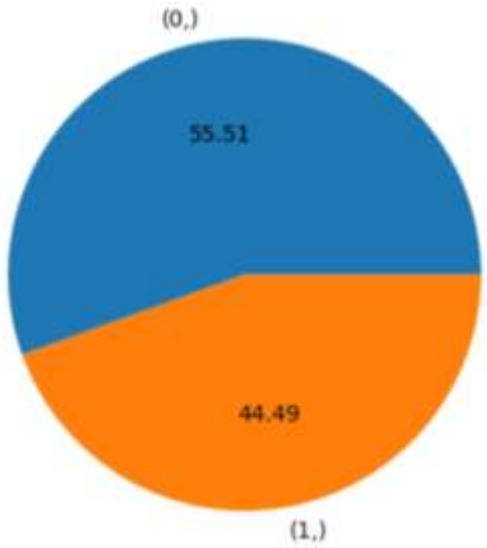
```
y_hat[0:5], y_test[0:5]
```

```
(array([1, 1, 0, 1, 0]),
 989  1
 682  1
 3    0
 739  1
 291  0)
```

۶-با استفاده از کدنویسی پایتون وضعیت تعادل داده ها در دو کلاس موجود در دیتاست را نشان دهید .آیا تعداد نمونه های کلاس ها با هم برابر است؟ عدم تعادل در دیتاست می تواند منجر به چه مشکلاتی شود؟ برای حل این موضوع چه اقداماتی می توان انجام داد؟ پیاده سازی کرده و نتیجه را مقایسه و گزارش کنید.

```
#barresi taadol dade ha
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

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

در ابتدا کتابخانه های مورد نیاز را ایمپورت میکنیم اولین کتابخانه ی موجود برای رسم استفاده میشود. سپس y را از نامپای به فرمت دیتا فریم تبدیل میکنیم تا بتوانیم از plot استفاده کنیم درنهایت تعداد داده هارا محاسبه کرده و رسم میکنیم.



مشاهده میشود که کلاس داده های ما از نظر تعدادی متعادل نیستند.

عدم تعادل در دیتاست، به معنای عدم توازن در توزیع کلاسها یا دستههای مختلف دادهها، میتواند به مشکلات مختلفی منجر شود .در زیر چند مشکل اصلی آورده شده است:

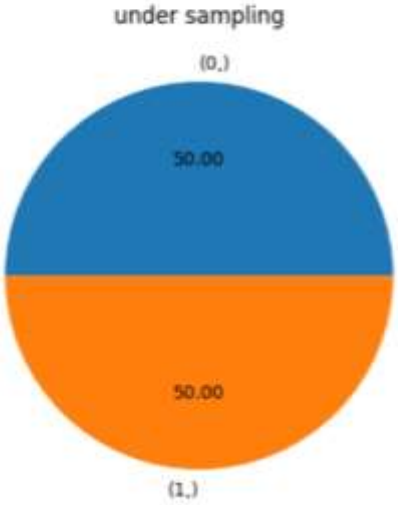
- مشکل در آموزش مدل:
  - در دیتاست های ناتوانمند به تعداد نمونه های هر کلاس، مدل ممکن است با مشکل مواجه شود .این موضوع میتواند منجر به یادگیری ناکافی برای کلاسهای کمنمونه شود.
- تاثیرات انحرافی:

- در صورتی که تعداد نمونههای یک کلاس زیاد باشد و برای کلاسهای دیگر کم باشد، مدل ممکن است به سمتی خاص بیفتد و به کلاسهای کمنمونه کمتر توجه کند .این موضوع میتواند به انحراف(bias) درپیش بینی ها منجر شود.

- دقت تخمینگر:
  - در صورت عدم تعادل، دقت تخمینگر برای کلاسهای با تعداد نمونه بیشتر بالا میرود، اما برای کلاسهای کم نمونه کاهش می یابد .این ممکن است باعث نادرست فهم شود که مدل بهترین عملکرد را ارائه میدهد.
- افزایش هزینه آموزش:
  - برای مدلهایی که با دیتاستهای ناتوانمند آموزش داده میشوند، احتمالاً نیاز به تلاش و هزینه زیادتری برای دستیابی به عملکرد خوب دارند.
- اهمال کلاس های کم نمونه:
  - ممکن است مدل به خاطر تعداد کم نمونهها به صورت اشتباهی کلاسهای کمنمونه را اهمال کند یا به اشتباه آنها را به عنوان نمونه های کلاس اکثریت (majority class) در نظر بگیرد.
- پایداری نتایج:
  - دقت و کارایی مدل ممکن است در مواجهه با دادههای جدید تحت تأثیر قرار گیرد، زیرا مدل ممکن است بر اساس نمونه های زیاد یک کلاس و بی توجه به کلاسهای کم نمونه باشد.

برای حل مشکلات مربوط به عدم تعادل دیتاست، روشهایی مانند oversampling افزایش نمونه های کم نمونه، Undersampling کاهش نمونه های بیشتری، یا استفاده از الگوریتمهای خاص مانند class weight مورد استفاده قرار میگیرند.

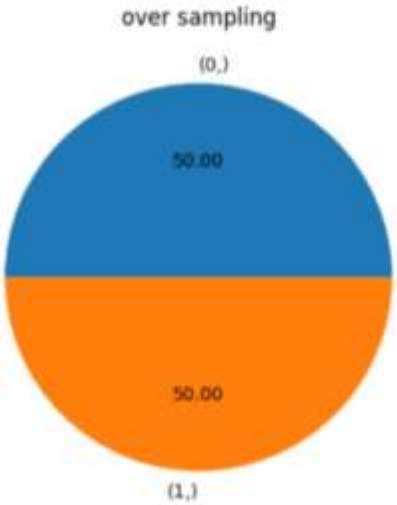
```
#class balancing random undersampling
from imblearn.under_sampling import RandomUnderSampler
y = pd.DataFrame(y, columns=['']) 
rus = RandomUnderSampler(sampling_strategy=1)
x_res_undersampling , y_res_undersampling = rus.fit_resample(X , y)
ax = y_res_undersampling.value_counts().plot.pie(autopct = '%.2f')
_ = ax.set_title("under sampling")
```



در ابتدا ستون تارگت هایمان را از نامپای به دیتا فریم تغییر میدهیم و سپس از کتابخانه اورده شده استفاده میکنیم و تعداد ایکس و ایگرگ هارا اپدیت میکنیم.

نمودار تعداد داده هارا رسم میکنیم. برای روش بعدی همین مراحل را تکرار میکنیم.

```
#class balancing random oversampling
from imblearn.over_sampling import RandomOverSampler
y = pd.DataFrame(y, columns=['']) 
rus = RandomOverSampler(sampling_strategy=1)
x_res_oversampling , y_res_oversampling = rus.fit_resample(X , y)
ax = y_res_oversampling.value_counts().plot.pie(autopct = '%.2f')
_ = ax.set_title("over sampling")
```



۷. فرآیند آموزش و ارزیابی مدل را با استفاده از یک طبقه بند آمادهٔ پایتونی انجام داده و اینبار در این حالت چالش عدم تعادل داده های کلاس ها را حل کنید.

یک بار دیگر مدل را با کتابخانه آماده در حالت عدم تعادل و بدون تعادل حل میکنیم.

```
[ ]  #bakhshe akhar
     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 = file2[["price1" ,"price2" , "price3" , "price4"]].values
     y = file2[["price5"]].values
     x_train , x_test , y_train , y_test = train_test_split(X , y , test_size = 0.2)
```

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

```
model = LogisticRegression(random_state = 93, solver='sag', max_iter=200)
model.fit(X , y)
```

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

LogisticRegression

```
LogisticRegression(max_iter=200, random_state=93, solver='sag')
```

```
model.score(x_test , y_test)
```

```
0.9745454545454545
```

```
model.score(x_train , y_train)
```

```
0.9708029197080292
```

```
y = pd.DataFrame(y , columns = [""])
y.value_counts()
```

```
0    761
1    610
dtype: int64
```

```
[48] #hal kardan ba rafee adam taadol
```

```
[49] 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 = 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: DataConversio
  y = column_or_1d(y, warn=True)
▾ LogisticRegression
LogisticRegression()
```

```
[51] y_hat = model.predict(x_test)
     model.score(x_test , y_test)

     1.0
```

```
[52] model.score(x_train , y_train)

     0.9918032786885246
```

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

      1.0
```

مشاهده میکنیم در حالت متعادل دقت بالاتری داریم.

سوال ۳

1 . به این پیوند مراجعه کرده و یک دیتاست مربوط به «بیماری قلبی» را دریافت کرده و توضیحات مختصری در مورد هدف و ویژگیهای آن بنویسید. فایل دانلودشدهٔ دیتاست را روی گوگل درایو خود قرار داده و با استفاده از دستور gdown آن را در محیط گوگل کولب بارگذاری کنید.

```
!gdown 1KfBK5V59MtQ-CrEEZK0YMfEDDryANhXs
```

```
Downloading...
From: https://drive.google.com/uc?id=1KfBK5V59MtQ-CrEEZK0YMfEDDryANhXs
To: /content/heart_disease_health_indicators.csv
100% 11.8M/11.8M [00:00<00:00, 207MB/s]
```

با استفاده از دستور gdown دیتاست مورد نظر را در گوگل کولب اپلود میکنیم.

با توجه به اطلاعاتی که در خود سایت این دیتاست نوشته شده است می توان فهمید که این مجموعه داده شامل شاخص های مختلف مرتبط با سلامت برای نمونه ای از افراد است. در اینجا توضیح مختصری از هر ستون آورده شده است:

Heart Diseaseor Attack:نشان می دهد که آیا فرد دچار بیماری قلبی یا حمله قلبی شده است (دودویی :0=خیر ، 1 = بله.)

HighBP: وضعیت فشار خون بالا (باینری: 0 = خیر، 1 = بله.)

HighChol : وضعیت کلسترول بالا (دودویی: 0 = خیر، 1 = بله).

CholCheck : دفعات بررسی کلسترول (طبقه ای).

BMI : شاخص توده بدن (مستمر).

Smoker: وضعیت سیگار کشیدن (دودویی: 0 = خیر، 1 = بله).

Stroke: سابقه سکته مغزی (باینری: 0 = خیر، 1 = بله).

Diabetes: وضعیت دیابت (دودویی: 0 = خیر، 1 = بله).

PhysActivity : سطح فعالیت بدنی (طبقه ای).

Fruits: فراوانی مصرف میوه (قسمتی).

Veggies: فراوانی مصرف سبزیجات (قسمتی).

HvyAlcoholConsump : وضعیت مصرف الکل سنگین (باینری: 0 = خیر، 1 = بله).

AnyHealthcare : دسترسی به هر مراقبت بهداشتی (باینری: 0 = خیر، 1 = بله).

NoDocbcCost : بدون پزشک به دلیل هزینه (باینری: 0 = خیر، 1 = بله).

GenHlth : ارزیابی سلامت عمومی (طبقه ای).

MentHlth : ارزیابی سلامت روان (مقوله ای).

PhysHlth : ارزیابی سلامت جسمانی (طبقه ای).

DiffWalk : وضعیت دشواری راه رفتن (باینری: 0 = خیر، 1 = بله).

Sex: جنسیت فرد (دودویی: 0 = زن، 1 = مرد).

Age: سن فرد (مستمر).

Education: مقطع تحصیلی (قسمتی).

Income: سطح درآمد (مقوله ای).

این مجموعه داده حاوی انواع اطلاعات مرتبط با سلامتی، 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.

2.ضمن توجه به محل قرارگیری هدف و ویژگیها، دیتاست را به صورت یک دیتافریم درآورده و با استفاده از دستورات پایتونی، ۱۰۰ نمونه داده مربوط به کلاس «۱» و ۱۰۰ نمونهداده مربوط به کلاس «۰» را در یک دیتافریم جدید قرار دهید و در قسمتهای بعدی با این دیتافریم جدید کار کنید.

```
import pandas as pd
import numpy as np
from sklearn.utils import shuffle
df = pd.read_csv("/content/heart_disease_health_indicators.csv")
df.keys()

df1 = df[df["HeartDiseaseorAttack"] == 1]
df1 =  df1.iloc[0:100 , :]

df2 = df[df["HeartDiseaseorAttack"] == 0]
df2 = df2.iloc[0:100 , :]

# df=[df1 , df2]
df = pd.concat([df1, df2] , ignore_index=True)

df = shuffle(df)
df = df.reset_index(drop=True)
df
```

در ابتدا دیتاست خود را با استفاده از دستور pd.read_csv میخوانیم و با دستور keys() میفهمیم چه فیچر هایی داخل دیتاست داریم و سپس 100 داده از کلاس 1 و 100 داده از کلاس 1 برمیداریم در داخل یک دیتافریم ریخته و باهم کانکت میکنیم به دلیل اینکه کلاس ها به ترتیب چیده میشوند(100 تای اول 1 100 تا دوم 0) داده هارا شافل میکنیم.
و سپس دوباره ایندکس هارا ریست میکنیم.

	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex
0	0	1	0	1	27	1	0	0	1	0	...	1	0	3	0	0	0	0
1	1	0	1	1	29	0	0	0	0	1	...	1	0	5	0	30	1	1
2	0	0	0	1	28	1	1	0	0	1	...	1	1	4	15	30	1	0
3	1	1	1	1	28	1	0	2	0	0	...	1	0	4	0	0	0	1
4	0	0	0	1	22	0	0	0	1	0	...	1	0	1	5	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
195	1	1	0	1	24	1	0	0	0	0	...	1	0	4	5	30	1	1
196	1	1	1	1	28	0	0	0	1	0	...	1	0	4	0	10	1	0
197	0	1	0	1	27	0	0	0	1	1	...	1	0	3	0	0	0	1
198	1	1	1	1	25	1	1	0	0	0	...	1	0	5	0	30	0	0
199	0	0	0	1	21	0	0	0	1	0	...	1	0	2	0	0	0	0

200 rows × 22 columns

۳. با استفاده از حداقل دو طبقهبند آمادهٔ پایتون و در نظر گرفتن فراپارامترهای مناسب، دو کلاس موجود در دیتاست را از هم تفکیک کنید. نتیجهٔ دقت آموزش و ارزیابی را نمایش دهید.

```
X = df[df.columns[1:]].values

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

y = df[["HeartDiseaseorAttack"]].values
X ,y
```

در ابتدا کتابخانه های لازم را در برنامه ایمپورت میکنیم و سپس کلیه ستون ها بجز ستون اول که تارگت است را به X اختصاص داده و ستون تارگت را به y میدهیم. و نتیجه را تست میکنیم.

```
(array([[ 1,  0,  1, ...,  9,  6,  8],
        [ 0,  1,  1, ..., 13,  5,  6],
        [ 0,  0,  1, ...,  7,  4,  3],
        ...,
        [ 1,  0,  1, ...,  8,  6,  8],
        [ 1,  1,  1, ..., 11,  4,  4],
        [ 0,  0,  1, ..., 10,  5,  6]]),
 array([[0],
        [1],
        [0],
        [1],
        [0],
        [0],
        [1],
        [1],
        [0],
        [1],
        [1],
        [0],
        [1],
        [0],
        [0],
        [1],
        [1],
        [1],
        [1],
        [0],
        [0],
        [0],
        [0],
        [0],
        [1],
        [0],
        [1],
        [1],
```

مشاهده میشود عملیات به درستی صورت گرفته است.

```
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
```

```
((160, 21), (40, 21), (160, 1), (40, 1))
```

سپس داده هارا به دو دسته ترین و تست با نسبت 80 به 20 مثل سوالات قبل تعریف میکنیم و اندازه انهارا چک میکنیم.

```
model = LogisticRegression(solver = 'sag', max_iter = 4000 , random_state = 93)
model.fit(x_train , y_train)
y_hat = model.predict(x_test)
model.predict(x_test) , y_test
y_hat.shape
```

سپس از مدل logisticregression به عنوان مدل مد نظر آماده ی خود استفاده میکنیم و انرا با دستور fit و با داده های ترین اموزش میدهیم.

سپس از مدل میخواهیم تا در مرحله تست مقدار y های متناظر با x_test را پیش بینی کند و انرا به y_hat نسبت میدهیم. در نهایت داده هارا چک میکنیم.

```
model.score(x_test , y_test)
```

```
0.6
```

```
model.score(x_train , y_train)
```

```
0.75
```

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

```
0.6
```

روش SGDClassifier:

```
model1 = SGDClassifier(loss = 'log_loss' , random_state = 93)
model1.fit(x_train , y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation
  y = column_or_1d(y, warn=True)
SGDClassifier(loss='log_loss', random_state=93)
```

```
y_hat1 = model1.predict(x_test)
```

همین کار را برای مدل دیگر انجام میدهیم.

```
model1.score(x_test , y_test)
```

```
0.575
```

```
model1.score(x_train , y_train)
```

```
0.66875
```

```
from sklearn.metrics import accuracy_score
score1 = accuracy_score(y_test, y_hat1)
score1
```

```
0.575
```

به همین صورت برای روش perceptron:

```
from sklearn.linear_model import Perceptron
model2 = Perceptron(random_state = 93)
model2.fit(x_train , y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn
  y = column_or_1d(y, warn=True)
```

```
Perceptron(random_state=93)
```

```
model2.score(x_train , y_train)
```

```
0.525
```

```
model2.score(x_test , y_test)
```

```
0.55
```

```
y_hat2 = model2.predict(x_test)
```

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_hat2)
score
```

```
0.55
```

(ایمپورت کردن یکبار کتابخانه ها کافیست لازم نیست هر سری این امر تکرار شود.)

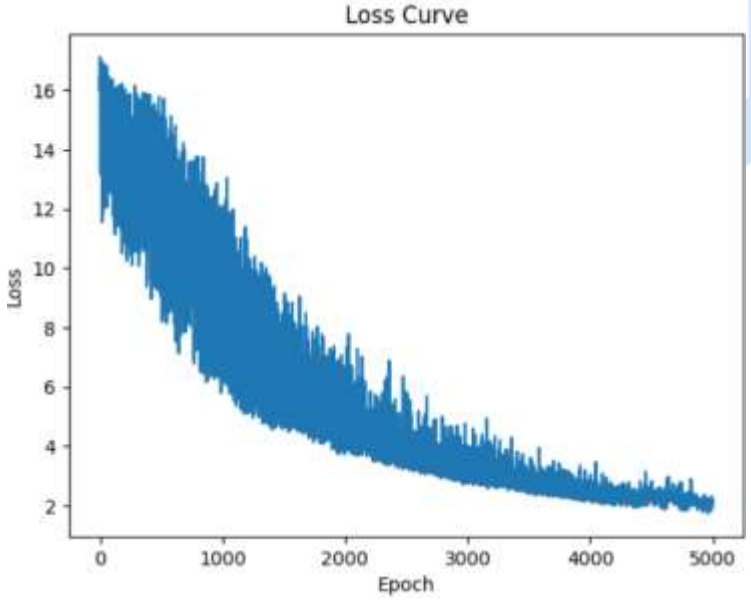
۴. در حالت استفاده از دستورات آمادهٔ سایکیتلرن، آیا راهی برای نمایش نمودار تابع اتلاف وجود دارد؟ پیادهسازی کنید.

بله. در واقع در این سوال می خواهیم با استفاده از کتابخانه های آماده در پایتون که در قسمت قبل داده ها را طبقه بندی کرده ایم در این حالت تابع اتلاف را بیابیم و رسم کنیم. طبق توضیحات گفته شده و خواسته شده باید هسته مرکزی مدل مورد استفاده در این سوال با استفاده از کتابخانه های آماده در پایتون زده شود و سپس به هر روشی که خواستیم و توانستیم تابع اتلاف را بیابیم و رسم کنیم. در این قسمت من سعی کردم از همان مدل هایی که در قسمت قبل استفاده کرده ایم در این قسمت نیز استفاده کنیم اما به طور مثال برخی از روش ها و مدل ها مانند LogisticRegression از دستورات مربوط به محاسبه تابع اتلاف به صورت عادی پیروی نمی

کردند و باید به صورت نقطه ای و iteration آن ها را مشخص می کردیم. برای همین منظور در این قسمت من از یک مدل آماده در سایکیت لرن استفاده کرده ام که دستورات اماده آن به صورت زیر می باشد :

```
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


model = SGDClassifier(loss='log', random_state=83)
losses = []
epochs = 5000

for i in range(epochs):

    model.partial_fit(x_train, y_train , [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()
```

در ابتدا تمامی کتابخانه های مورد نیاز را در محیط import می کنیم و با استفاده از کلاسSGDCeslassifier مدل مورد نظرمان را روی داده ها پیاده می کنیم و در داخل model می ریزیم. در ادامه تابعی با نام losses تعریف می کنیم که در ادامه مقادیر اتلافی را در داخل آن بریزیم. در این قسمت برای بهبودی عملکرد مقدار اپیاک ها را زیاد و روی 5000 قرار می دهیم اما این مدل برای اپیاک های کمتر از این هم جوابگو خواهد بود. در ادامه اپیاک ها را به مدل مورد نظرمان اعمال می کنیم و تابع losses را append می کنیم.در ادامه نیز با استفاده از دستورات و کتابخانه های پایتون تابع اتلافی مورد نظرمان را رسم می کنیم :



می بینیم که در این حالت نیز هر چه تعداد اپیاک ها بیشتر می شود مقدار تلاف ما نیز کمتر می شود تا به یک مقدار مشخصی میل کند. حتی می توانیم برای قشنگ تر شدن ظاهر نمودار تابع اتلافی سوال ، از دستورات curve fitting استفاده کنیم. اما باید به تعداد اپیاک ها توجه کنیم زیرا انتخاب آن ها خیلی تاثیری زیادی در خطای سیستم دارد. اگر مقدار آن ها کم باشد تابع خطای زیادی را به ما نشان می دهد و از آن طرف نیز اگر تعداد اپیاک ها را زیاد انتخاب کنیم ممکن است مدل ما دچار undermodeling شود، پس تعداد اپیاک ها مهم است.

۵ .یک شاخصهٔ ارزیابی جدید (غیر از accuracy) تعریف کنید و بررسی کنید که از چه طریقی میتوان این شاخص را در ارزیابی داده های تست نمایش داد. پیادهسازی کنید.

می توانیم از شاخص جدیدی که به تازگی در کلاس درس ارائه شده است ، استفاده کنیم :

ماتریس کانفیوژن (Confusion Matrix) یک ابزار آماری است که برای ارزیابی عملکرد یک مدل طبقه‌بندی استفاده می‌شود .این ماتریس به صورت یک جدول مربعی نمایش داده می‌شود که در آن هر سطر مربوط به یک کلاس واقعی و هر ستون مربوط به یک کلاس پیش‌بینی شده است.

در هر سطر از ماتریس، تعداد نمونه‌های واقعی آن کلاس که به درستی پیش‌بینی شده‌اند و تعداد نمونه‌های واقعی آن کلاس که به اشتباه پیش‌بینی شده‌اند، نشان داده می‌شود .

معیارهای مختلفی از ماتریس کانفیوژن برای ارزیابی عملکرد یک مدل طبقه‌بندی استفاده می‌شود .برخی از این معیارها عبارتند از:

- دقت :(Accuracy) نسبت کل نمونه‌های پیش‌بینی شده صحیح به کل نمونه‌ها.

$$Accuracy = (TP + TN) / (TP + FP + FN + TN)$$

- صحت :(Precision) نسبت نمونه‌های پیش‌بینی شده صحیح به کل نمونه‌هایی که به عنوان آن کلاس پیش‌بینی شده‌اند.

$$Precision = TP / (TP + FP)$$

- فراخوانی :(Recall) نسبت نمونه‌های پیش‌بینی شده صحیح به کل نمونه‌های واقعی آن کلاس.

$$Recall = TP / (TP + FN)$$

- F1-score: میانگین حسابی دقت و فراخوانی.

$$F1\text{-}score = 2 * Precision * Recall / (Precision + Recall)$$

انتخاب معیار مناسب برای ارزیابی عملکرد یک مدل طبقه‌بندی به عوامل مختلفی بستگی دارد، از جمله اینکه کدام کلاس‌ها از اهمیت بیشتری برخوردارند و اینکه خطاهای پیش‌بینی در کدام کلاس‌ها قابل قبول‌تر هستند. ماتریس کانفیوژن یک ابزار مفید برای ارزیابی عملکرد یک مدل طبقه‌بندی است .با استفاده از این ماتریس، می‌توان نقاط قوت و ضعف یک مدل را شناسایی کرد و برای بهبود عملکرد آن اقدامات لازم را انجام داد.

برای f1score داریم:

```
#bakhshe akhar
from sklearn.metrics import confusion_matrix , f1_score
import matplotlib.pyplot as plt
```

```
cm = confusion_matrix(y_test , y_hat)
F1 = f1_score(y_test , y_hat , average=None)
F1
```

```
array([0.55555556, 0.63636364])
```

برای confusion matrix داریم:

```
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

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

نتیجه: