دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

مبانی سیستم‌های هوشمند

پروژه اول – بخش ۱

استاد درس: دکتر علیاری

نام و نام خانوادگی: زهرا ایران‌پور مبارکه

شماره دانشجویی: ۹۸۱۹۸۹۳

پاییز ۱۴۰۲

فهرست

چکیده

این گزارش شامل پاسخ کدنویسی و تشریحی (تحلیلی) به ۳ سوال کلی می‌باشد.

در سوال اول، با تولید یک دیتاست ۲کلاسه، با استفاده از ۲ طبقه‌بندی آماده پایتون کلاس‌ها از هم جدا شده و ضمن بدست آمدن خطا و درصد پیش‌بینی این دو طبقه‌بندی، و بدست آمدن مرز و نواحی تصمیم‌گیری، سوال مجددا برای حالت چالش برانگیزتر حل خواهد شد و در نهایت، یک کلاس به دیتاست اضافه شده و در مورد تغییرات آن توضیح داده خواهد شد.

در سوال دوم، از یک دیتاست مربوط به حوزه بانکی استفاده شده و با انجام اعمال preprocessing و train-test-split، آماده جداسازی کلاس‌ها می‌شود. در ادامه ضمن تحلیل اطلاعات بدست آمده، داده‌ها به روش مناسب نرمال‌سازی شده و پیش‌بینی مجددا انجام می‌شود تا تفاوت آشکار شود. در نهایت، وضعیت تعادل کلاس‌ها بررسی می‌شود و با استفاده از یک طبقه‌بندی آماده فرایند آموزش و ارزیابی مدل انجام می‌شود.

در سوال سوم، ، از یک دیتاست مربوط به بیماری قلبی استفاده شده و به دیتافریم تبدیل می‌شود. سپس، با مدنظر قرار دادن دوکلاس، دیتافریم جدیدی تشکیل می‌شود. در ادامه، با استفاده از دو طبقه‌بندی آماده پایتون، فرایند آموزش و ارزیابی مدل انجام می‌شود و دوکلاس از هم تفکیک می‌شوند. نهایتا یک شاخصه ارزیابی تعریف و بررسی می‌شود.

سوال ۱

ابتدا کتابخانه‌های مورد نیاز در این سوال اضافه می‌شود:

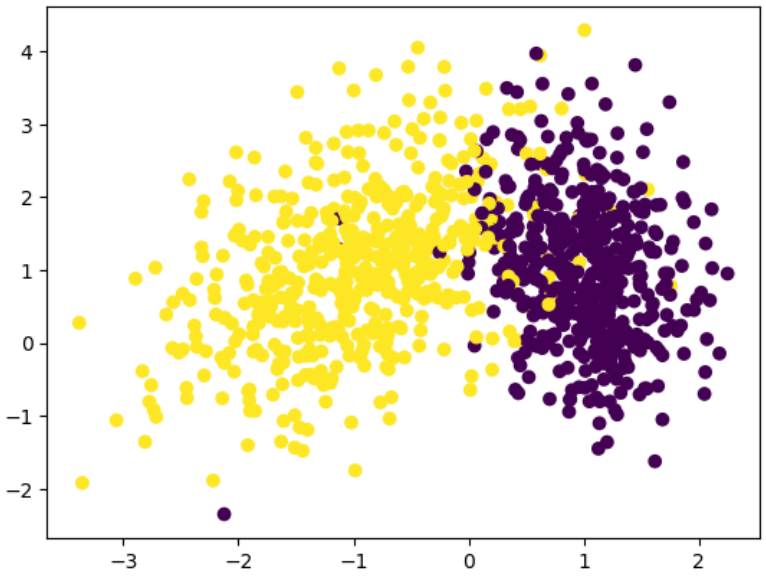
```
from math import sqrt
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, SGDClassifier
import sklearn.metrics as met
from sklearn.metrics import mean_absolute_error, mean_squared_error, confusion_matrix
from mlxtend.plotting import plot_decision_regions
```

۱. در این قسمت با استفاده از sklearn.datasets یک دیتاست با ۱۰۰۰ نمونه، ۲ کلاس و ۲ ویژگی تولید می‌شود:

```
X, y = make_classification(n_samples=1000, n_features=2, n_redundant=0, n_classes=2, class_sep=1 , n_clusters_per_class=1, random_state=93)
```

سپس با استفاده از دستور plt.scatter، نموداری جهت نمایش داده‌ها رسم می‌شود:

```
plt.scatter(X[:, 0], X[:, 1], c=y)
```



۲. در این قسمت ابتدا ۲۰ درصد داده‌ها برای تست و ۸۰٪ آن‌ها برای آموزش در نظر گرفته می‌شود:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=93)
X_train.shape, X_test.shape, y_train.shape, y_test.shape = ((800, 2), (200, 2), (800,), (200,))
```

برای بهبود نتیجه لازم بود فراپارامترهای مناسب که با داده‌های ما متناسب هستند انتخاب شوند و تاثیر تغییر آن‌ها در مقدار خطا و خروجی در نظر گرفته شود.

۱-۲) با استفاده از logistic regression به عنوان یکی از طبقه‌بندی‌های آماده پایتون، فرآیند آموزش مدل شروع می‌شود و مدل روی داده‌های آموزش فیت می‌شود:

```
model1 = LogisticRegression(solver='lbfgs', max_iter=2000, multi_class='ovr', penalty='l2', random_state=93)
model1.fit(X_train, y_train);
```

در قسمت انتخاب solver، گزینه‌های متفاوتی در دست است ولی چون lbfgs برای داده‌های کوچک تا متوسط که تعداد ویژگی‌های زیادی ندارد کاربردی است، این گزینه انتخاب شد.

حداکثر تکرار روی ۲۰۰۰ قرار داده می‌شود تا هم به حدی زیاد نباشد که ران شدن برنامه را کند کند، و هم به قدری پایین نباشد که مدل به خوبی عمل نکند.

از آن‌جایی که داده فقط ۲ کلاس دارد، در بخش multi_class گزینه one-vs-rest که دیفالت هم هست انتخاب می‌شود. اگر بیش از دو کلاس داشت، به جای آن multinomial قرار داده می‌شود.

در ادامه در قسمت penalty، مدل منظم‌سازی انتخاب می‌شود و lbfgs فقط l2 را قبول می‌کند و l2 به جلوگیری از overfit شدن کمک می‌کند. لذا این گزینه انتخاب می‌شود.

همچنین، random state برابر با دو رقم آخر شماره دانشجویی بنده قرار داده شده است.

در ادامه به ۳ روش می‌توان پیش‌بینی را انجام داد. روش اول به سادگی خود کلاس را پیش‌بینی می‌کند، روش دوم احتمال را و روش سوم لگاریتم احتمال را.

```
ypred1 = model1.predict(X_test)
ypred1_2 = model1.predict_proba(X_test)
ypred1_3 = model1.predict_log_proba(X_test)
```

روش اول برای ما در اینجا کاربرد بیشتری دارد. لذا با دستور ypred1, y_test به صورت چشمی مشاهده می‌شود که چه تفاوتی بین خروجی پیش‌بینی شده و خروجی اصلی در داده‌های تست وجود دارد.

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

درنهایت، accuracy برای دادهای تست و آموزش به صورت جداگانه بدست آورده می‌شود:

```
a = model1.score(X_train, y_train)
b = model1.score(X_test, y_test)
[a, b]
```

نتیجه:

```
[0.94625, 0.95]
```

نهایتا خطاهای مختلف نیز برای داده تست محاسبه می‌شود:

```
a = met.mean_squared_error(y_test, ypred1)     # MSE
b = mean_absolute_error(y_test, ypred1)        # MAE
c = sqrt(mean_squared_error(y_test, ypred1))   # SMSE
[a, b, c]
```

نتیجه:

```
[0.05, 0.05, 0.22360679774997896]
```

و برای داشتن دید بهتر، ماتریس درهم ریختگی بدست آورده می‌شود:

```
confusion_matrix(y_test, ypred1)
```

نتیجه:

```
array([[104,   3],
       [  7,  86]], dtype=int64)
```

می‌دانیم هرچه قطرهای این ماتریس بزرگتر باشد، یعنی مدل بهتر است و از ماتریس بدست آورده شده همین موضوع برآورد می‌شود.

۲-۲) با استفاده از SGD Classifier به عنوان یکی از طبقه‌بندی‌های آماده پایتون، فرآیند آموزش مدل انجام می‌شود و مدل روی داده‌های آموزش فیت می‌شود:

```
model2 = SGDClassifier(loss='log_loss', random_state=93)
model2.fit(X_train, y_train);
```

در قسمت انتخاب loss، چون log loss برای کلاس‌های ۰ و ۱ اعمال می‌شود، و داده هم فقط ۲ کلاس دارد، این گزینه انتخاب می‌شود.

همچنین، random state برابر با دو رقم آخر شماره دانشجویی بنده قرار داده شده است.

در ادامه مانند قسمت قبل به ۳ روش می‌توان پیش‌بینی را انجام داد. که در اینجا جهت جلوگیری از تکرار، فقط پاسخ روش اول آورده می‌شود که آرایه اول پیش‌بینی و آرایه دوم کلاس‌بندی درست داده تست است:

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

همچنین، accuracy برای دادهای تست و آموزش به صورت جداگانه مانند قسمت قبل بدست آورده می‌شود که داده اول مربوط به داده آموزش و داده دوم مربوط به داده تست است:

```
[0.93, 0.895]
```

نهایتا خطاهای مختلف نیز برای داده تست محاسبه می‌شود:

```
[MSE, MAE, SMSE] = [0.105, 0.105, 0.324037034920393]
```

و برای داشتن دید بهتر، ماتریس درهم ریختگی بدست آورده می‌شود:

```
array([[92, 15],
       [ 6, 87]], dtype=int64)
```

می‌دانیم هرچه قطرهای این ماتریس بزرگتر باشد، یعنی مدل بهتر است و از ماتریس بدست آورده شده همین موضوع برآورد می‌شود.

۳. مرز و نواحی تصمیم‌گیری برای دو مدل نشان داده می‌شود و نمونه‌هایی که اشتباه طبقه‌بندی شده‌اند در شکل متفاوت نشان داده می‌شود:

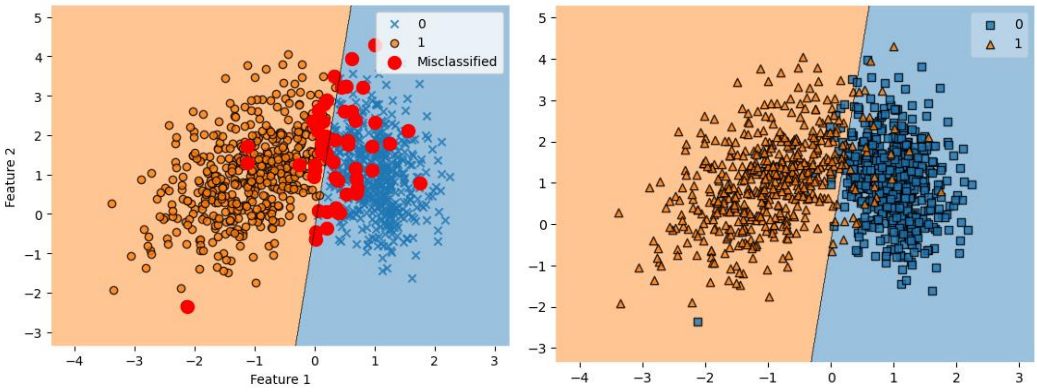
کد رسم شکل اصلی:

```
plot_decision_regions(X, y , clf=model2)
```

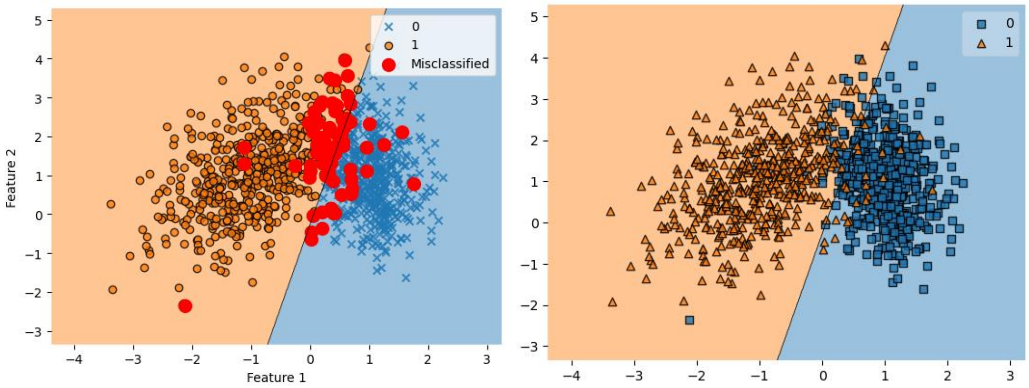
کد رسم نمونه‌های اشتباه:

```
ypred22 = model2.predict(X)
misclassified2 = ypred22 != y
plot_decision_regions(X, y, clf=model2, markers='xo', X_highlight=X[misclassified2])
plt.scatter(X[misclassified2][:, 0], X[misclassified2][:, 1], c='red', marker='o', s=100, label='Misclassified')
plt.show()
```

مدل logistic regression:

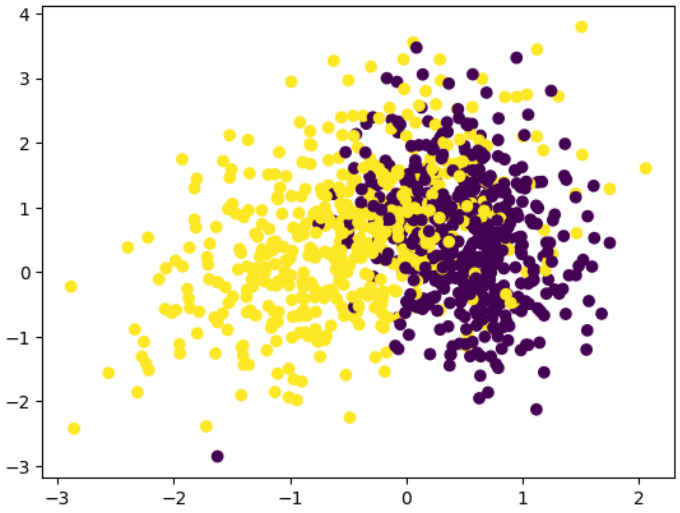


مدل SGD Classifier: مشاهده می‌شود که این مدل تعداد خطای بیشتری دارد



۴. با استفاده از تغییر class_sep و کمتر کردن آن از مقدار ۱، داده‌ها پیچیدگی پیدا کرده و درهم تنیده می‌شوند.

برای مثال آن را روی ۰.۵ قرار می‌دهیم و شکل داده‌ها به این صورت در می‌آید:



حال مراحل قبلی را برای داده‌های جدید تکرار می‌کنیم و برای جلوگیری از تکرار کدها، فقط خروجی آن‌ها نمایش داده می‌شود.

۴-۲) در این قسمت مجددا ۲۰ درصد داده‌ها برای تست و ۸۰٪ آن‌ها برای آموزش در نظر گرفته می‌شود.

X_train.shape, X_test.shape, y_train.shape, y_test.shape = ((800, 2), (200, 2), (800,), (200,))

۴-۲-۱) logistic regression

خروجی پیش‌بینی شده و خروجی اصلی:

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

درصد درستی:

accuracy score for train and test respectively = [0.7925, 0.78]

خطاها:

[MSE, MAE, SMSE] = [0.105, 0.105, 0.324037034920393]

ماتریس درهم ریختگی:

confusion_matrix = array([[92, 15],
[6, 87]], dtype=int64)

۴-۲-۲) SGD Classifier

خروجی پیش‌بینی شده و خروجی اصلی:

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

درصد درستی:

accuracy score for train and test respectively = [0.77625, 0.775]

خطاها:

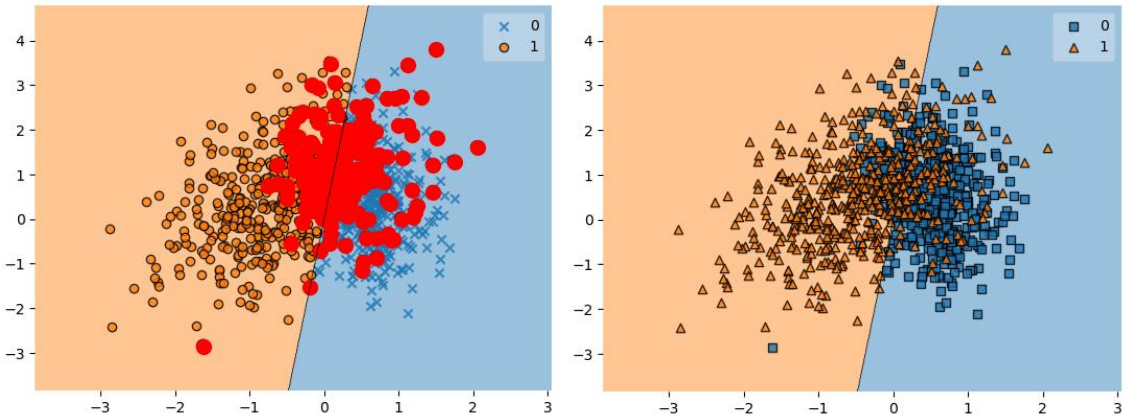
[MSE, MAE, SMSE] = [0.225, 0.225, 0.4743416490252569]

ماتریس درهم ریختگی:

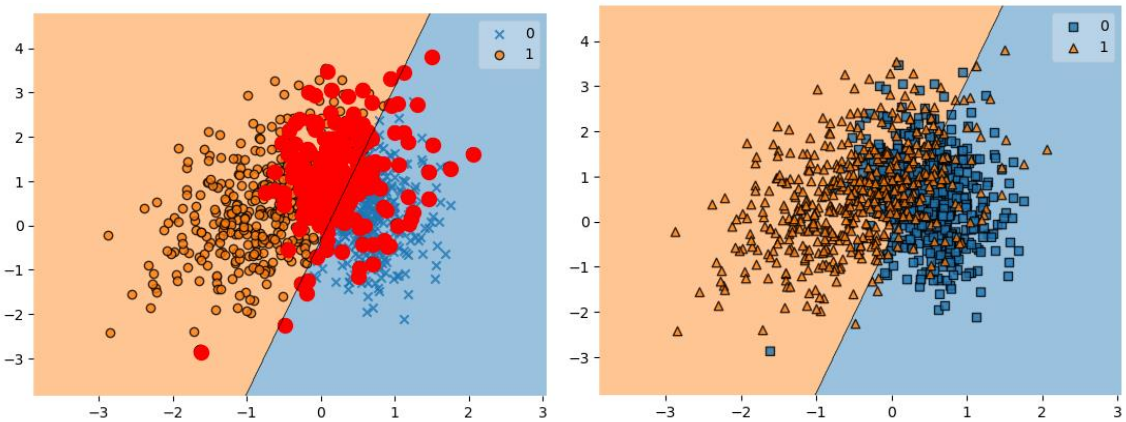
confusion_matrix = array([[75, 32],
[13, 80]], dtype=int64)

۳-۴) مرز و نواحی تصمیم‌گیری

مدل logistic regression:



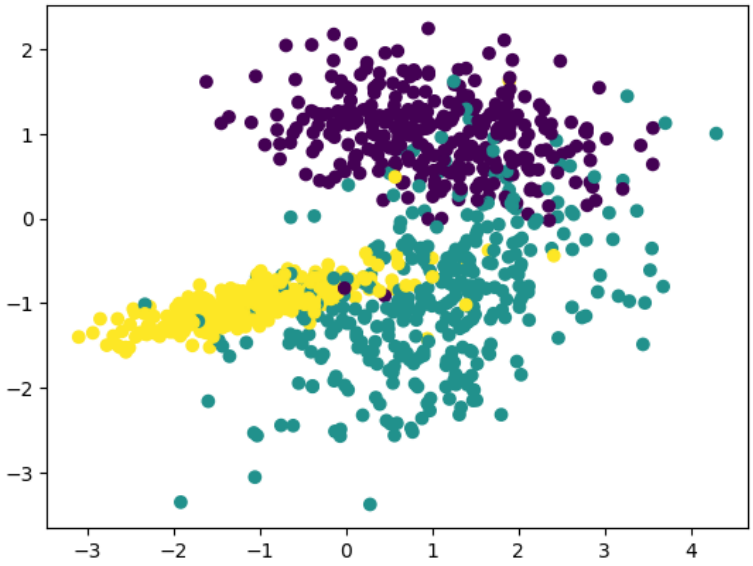
مدل SGD Classifier:



مقایسه: همان‌طور که مشاهده می‌شود هم تعداد نمونه‌هایی که اشتباه کلاس‌بندی شده‌اند بیشتر است و هم مقدار خطا افزایش یافته و score کاهش یافته است.

۵. در این قسمت یک کلاس به داده‌های تولید شده در قسمت ۱ اضافه می‌شود.

نمودار داده‌ها:



به این دلیل دیگر کلاس‌ها به صورت ۰ و ۱ نیست، باید تغییراتی در مدلسازی صورت گیرد. ولی هم‌چنان می‌توان با استفاده از کتابخانه‌های آماده پایتون این تغییرات را اعمال کرد.

- مدلسازی Logistic Regression:

```
model1 = LogisticRegression(solver='lbfgs', max_iter=2000, multi_class='multinomial', penalty='l2', random_state=93)
```

از آن‌جایی که داده اکنون بیش از ۲ کلاس دارد، در بخش multi_class گزینه one-vs-rest که دیفالت هم هست انتخاب نمی‌شود. به جای آن multinomial قرار داده می‌شود.

بقیه پارامترها مشکلی ندارند.

- مدلسازی SGD Classifier:

```
model2 = SGDClassifier(loss='log', random_state=93)
```

در قسمت انتخاب loss، چون log loss برای کلاس‌های ۰ و ۱ اعمال می‌شود، و داده اکنون بیش از ۲ کلاس دارد، این گزینه انتخاب نمی‌شود. به جای آن از log استفاده می‌شود که روش کار آن به این صورت بوده که ابتدا یک کلاس را از باقی کلاس ها جدا میکند و سپس برای کلاس دیگر همین کار را انجام میدهد و در نهایت با استفاده از تابع softmax خروجی‌ها را با هم ترکیب می‌کند.

خروجی‌ها:

- logistic regression:

خروجی پیش‌بینی شده و خروجی اصلی:

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

درصد درستی:

accuracy score for train and test respectively = [0.885, 0.895]

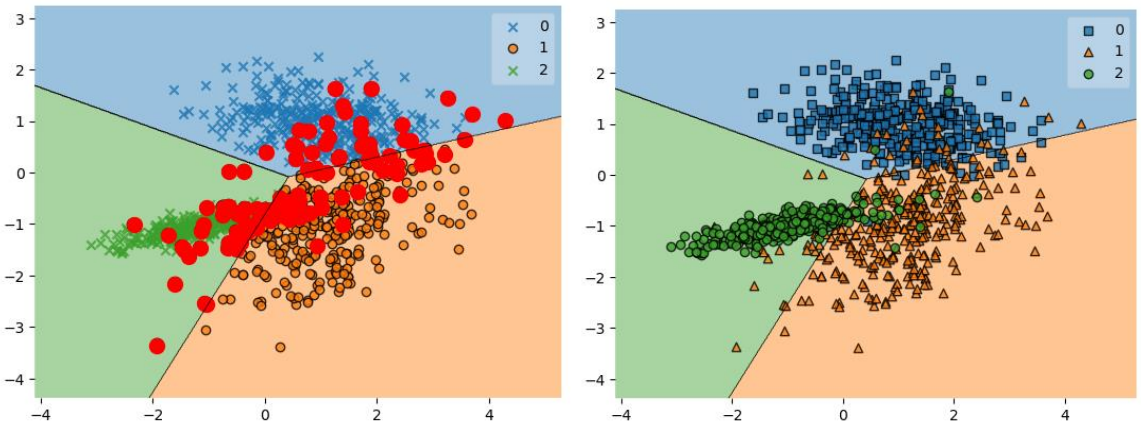
خطاها:

[MSE, MAE, SMSE] = [0.12, 0.11, 0.34641016151377546]

ماتریس درهم ریختگی:

confusion_matrix =
array([[61, 3, 0],
       [3, 58, 4],
       [1, 10, 60]], dtype=int64)

مرز و نواحی تصمیم‌گیری:



- SGD Classifier:

خروجی پیش‌بینی شده و خروجی اصلی:

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

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

درصد درستی:

accuracy score for train and test respectively = [0.84125, 0.845]

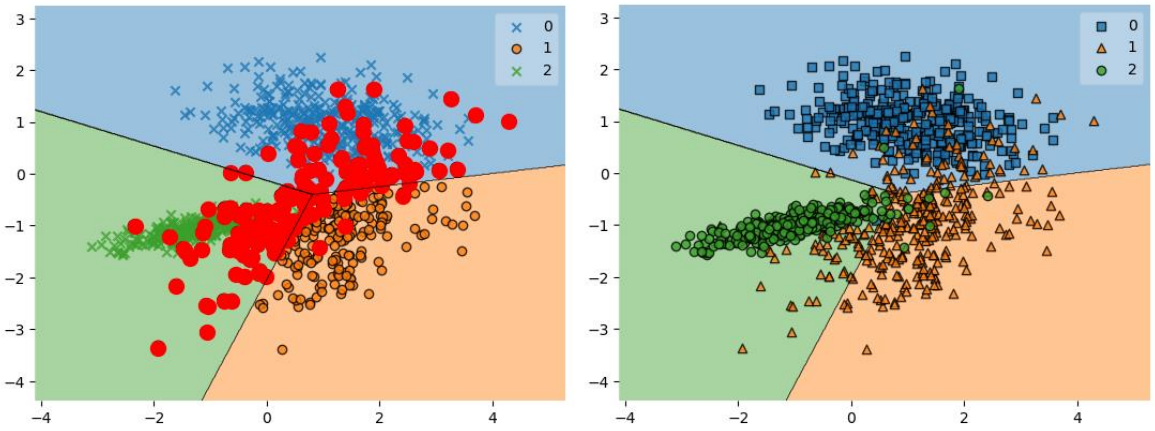
خطاها:

[MSE, MAE, SMSE] = [0.12, 0.11, 0.34641016151377546]

ماتریس درهم ریختگی:

confusion_matrix =
array([[64, 0, 0],
       [14, 38, 13],
       [1, 3, 67]], dtype=int64)

- مرز و نواحی تصمیم‌گیری:



سوال ۲

ابتدا کتابخانه‌های مورد نیاز در این سوال اضافه می‌شود:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.metrics as met
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, mean_absolute_error, mean_squared_error, confusion_matrix
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from math import sqrt
```

۱. در این سوال با یک دیتاست حوزه بانکی آشنایی به عمل می‌آید.

اهداف و ویژگی‌های دیتاست:

این داده‌ها، دارای ۱۳۷۲ نمونه و چندمتغیره هستند و از تصاویری که برای ارزیابی یک روش احراز هویت برای اسکناس‌های واقعی و جعلی گرفته شده بود، استخراج شده‌اند. برای دیجیتالی کردن، از یک دوربین صنعتی که معمولاً برای بازرسی چاپ است، استفاده شده‌است. تصاویر نهایی دارای ۴۰۰*۴۰۰ پیکسل هستند. با توجه به لنز شی و فاصله تا جسم مورد بررسی، تصاویری در مقیاس خاکستری با وضوح حدود ۶۶۰ نقطه در اینچ بدست آمده‌است. ابزار تبدیل موجک برای استخراج ویژگی‌ها از تصاویر استفاده شده.

- واریانس تصویر تبدیل شده موجک (پیوسته)
- چولگی تصویر تبدیل شده موجک (پیوسته)
- کورتوز تصویر تبدیل شده موجک (پیوسته)
- آنتروپی تصویر (پیوسته)
- کلاس (عدد صحیح)

سپس دیتا با دستور gdown در محیط گوگل کلب وارد می‌شود. Id مد نظر با آپلود دیتا در گوگل درایو و برداشتن قسمتی از لینک آن بدست می‌آید.

```
https://drive.google.com/file/d/1V1QyoKtRkK6Ln2OJjtnfvQ67II7-VG3T/view?usp=sharing
pip install --no-cache-dir gdown
!gdown 1V1QyoKtRkK6Ln2OJjtnfvQ67II7-VG3T
```

با مشاهده داده مشخص می‌شود که نمونه اول به عنوان سرتیتر درنظر گرفته شده. به همین دلیل، هنگام تبدیل دیتا، دیتافریم را بدون هدر تنظیم می‌کنیم تا مشکل حل شود.

```
df = pd.read_csv('data_banknote_authentication.txt' ,header=None)
```

۲. در این قسمت از سوال به فرآیند برزدن داده‌ها و در ادامه train-test-split می‌پردازیم.

بر زدن (Shuffling) داده‌ها در مسائل یادگیری ماشین اهمیت بسیاری دارد و در مراحل آموزش الگوریتم‌های یادگیری ماشین انجام می‌شود. برخی از اهمیت‌های بر زدن داده‌ها عبارتند از:

- پیشگیری از تغییرات ناخواسته: اگر داده‌ها در ترتیب مرتب شوند، ممکن است الگوریتم به سرعت یاد بگیرد اما نتواسته الگوهای کلی و عمومی را به خوبی یاد بگیرد. در واقع، الگوریتم ممکن است به جزئیات خاص داده‌ها علاقه‌مند شود و نتواسته به خوبی عمومی‌ترین الگوها را تشخیص دهد.
- پیشگیری از برازش زیاد (Overfitting): اگر الگوریتم بر روی داده‌هایی که به هم وابسته هستند آموزش ببیند، ممکن است از رفتارهای غیر عمومی و به خصوصیت‌هایی که فقط در داده‌های خاص وجود دارند برازش زیاد (overfitting) کند. بر زدن داده‌ها کمک می‌کند تا این مشکل پیشگیری شود.
- بهبود تعمیم‌پذیری: بر زدن داده‌ها باعث می‌شود الگوریتم بیشترین تعداد اطلاعات را از داده‌ها به دست آورده و به خوبی تعمیم‌پذیر شود. این بهبود تعمیم‌پذیری مدل در مقابل داده‌های جدید و ناشناخته را افزایش می‌دهد.
- پیشگیری از بایاس ناخواسته: اگر داده‌ها به ترتیب خاصی آموزش داده شوند، الگوریتم ممکن است با برخی خصوصیت‌های ناخواسته داده‌ها آشنا شود و این باعث بایاس شود. با بر زدن داده‌ها، این امکان را کاهش می‌دهیم.

عملیات بر زدن داده‌ها (اینکس‌ها هم در این عملیات به هم می‌ریزند که ممکن است در ادامه مشکل‌آفرین باشد. به همین دلیل پس از بر زدن، آن‌ها را ریست می‌کنیم):

```
df_shuffled = df.sample(frac=1, random_state=93)
df_shuffled.reset_index(inplace = True, drop = True)
df_shuffled
```

تقسیم به دو بخش آموزش و ارزیابی (تست) با نسبت تبدیل ۲۰٪:

```
X = df_shuffled.iloc[:, 0:4]
y = df_shuffled.iloc[:, 4:5]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=93)
X_train.shape, X_test.shape, y_train.shape, y_test.shape = ((1097, 4), (275, 4), (1097, 1), (275, 1))
```

۳. اگر بخواهیم بدون استفاده از کتابخانه آماده پایتون، مدل، تابع اتلاف و الگوریتم یادگیری و ارزیابی را کدنویسی کنیم، از گرادیان کاهشی استفاده کرده و کد آن را می‌نویسیم تا دو کلاس دیتاست به خوبی از هم جدا شوند:

تعریف تابع سیگموئید برای رگرسیون لجستیک:

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

تعریف تابع تلفات متقابل آنتروپی (افزودن اپسیلون برای جلوگیری از مشکلات لگاریتم صفر):

```
def cross_entropy_loss(y_true, y_pred):
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    return - (y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
```

محاسبه گرادیان برای رگرسیون لجستیک:

```
def compute_gradient(X, y, weights):
    m = len(y)
    predictions = sigmoid(np.dot(X, weights))
    gradient = np.dot(X.T, predictions - y) / m
    return gradient
```

آموزش مدل رگرسیون لجستیک با استفاده از گرادیان نزولی (مقدار دهی اولیه وزن ها با صفر):

```
def train(X, y, learning_rate, epochs):
    m, n = X.shape
    weights = np.zeros((n, 1))
    losses = []
    for epoch in range(epochs):
        gradient = compute_gradient(X, y, weights)
        weights -= learning_rate * gradient
        loss = np.mean(cross_entropy_loss(y, sigmoid(np.dot(X, weights))))
        losses.append(loss)
    return weights, losses
```

ارزیابی مدل رگرسیون لجستیک بر روی داده‌های جدید (بر اساس آستانه، احتمالات به برچسب‌های باینری تبدیل می‌شوند):

```
def evaluate(X, weights, threshold=0.5):
    predictions = sigmoid(np.dot(X, weights))
    labels = (predictions >= threshold).astype(int)
    return labels
```

مدلسازی:

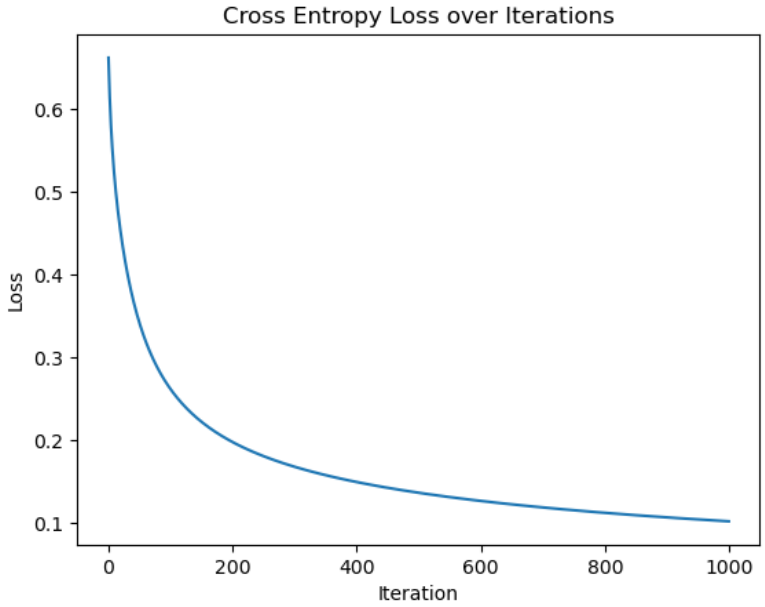
```
learning_rate = 0.01
epochs = 1000
trained_weights, losses = train(X_train, y_train, learning_rate, epochs)
```

رسم نمودار:

```
plt.plot(losses)
plt.title('Cross Entropy Loss over Iterations')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.show()
```

بررسی مدل:

```
test_predictions = evaluate(X_test, trained_weights)
accuracy = np.mean(test_predictions == y_test)
accuracy = 0.970909
```



تحلیل نمودار تابع اتلاف: نمودار تابع اتلاف که در بالا رسم شد، نشان‌دهنده تغییرات مقدار تابع هزینه (تابع اتلاف) در طول فرآیند آموزش است. در زیر تحلیلی از این نمودار ارائه شده است:

- کاهش تدریجی تابع اتلاف: همانطور که انتظار می‌رود، با پیشرفت آموزش، تابع اتلاف به تدریج کاهش پیدا می‌کند. این نشان‌دهنده بهینه‌سازی و بهبود وزن‌ها در جهتی که تابع هزینه کمینه شود، است.
- همگرایی: نمودار نشان می‌دهد که تابع اتلاف همگراست و به یک مقدار پایدار همگرا می‌شود. این نشان‌دهنده این است که فرآیند آموزش به یک وزن بهینه رسیده است.
- تغییرات تابع اتلاف در مراحل آخر آموزش: مراحل آخر آموزش، تغییرات در تابع اتلاف کمتر می‌شود و نمودار به یک خط میل می‌کند. این نشان‌دهنده استقرار مدل در یک وضعیت که دیگر بهبود قابل توجهی ندارد.

تحلیل نمودار تابع اتلاف مهم است تا مشاهده کنیم که آیا مدل به اندازه کافی بهینه شده است یا نیاز به افزایش تعداد ایپاک‌ها (epochs) داریم. از آن‌جایی که تابع اتلاف به سرعت کاهش پیدا کرده و سپس به یک مقدار ثابت رسیده، نشان‌دهنده برازش خوب مدل است.

نظر در مورد عملکرد مدل قبل از ارزیابی از روی تابع اتلاف:

نمودار تابع اتلاف می‌تواند به ما اطلاعات مهمی درباره عملکرد مدل بدهد، اما این تا حدی است که به علتی به نام "برازش بر روی داده‌های آموزش" (overfitting)، ممکن است مدل به داده‌های آموزش بسیار خوب برازش یافته باشد ولی بر روی داده‌های تست یا داده‌های جدید عملکرد مناسبی نداشته باشد. به عبارت دیگر، این ممکن است نشان‌دهنده این باشد که مدل به داده‌های آموزش "خاص" شده و توانمندی عمومی برای تفکیک موارد جدید را نداشته باشد.

به همین دلیل، نمودار تابع اتلاف به تنهایی کافی نیست و نیاز به ارزیابی روی داده‌های جدید (تست) داریم. اگر تابع اتلاف بر روی داده‌های آموزش به خوبی کاهش پیدا کرده باشد اما عملکرد مدل بر روی داده‌های تست به اندازه مطلوب نباشد، ممکن است مدل ما با مشکلاتی مثل برازش بر روی داده‌های آموزش روبرو شده باشد.

برای ارزیابی بهتر عملکرد مدل، می‌توان از معیارهایی مانند دقت (accuracy)، بازخوانی (recall)، دقت (precision) و اندازه F1 (F1-score) بر روی داده‌های تست استفاده کرد. این معیارها به ما اطلاعاتی درباره توانمندی مدل در تشخیص مثبت‌ها و منفی‌ها ارائه می‌دهند و می‌توانند بهتر از تابع اتلاف به ما کمک کنند تا عملکرد مدل را به صورت جامع‌تر ارزیابی کنیم.

برای این کار:

```
test_predictions = evaluate(X_test, trained_weights)
accuracy = accuracy_score(y_test, test_predictions)
precision = precision_score(y_test, test_predictions)
recall = recall_score(y_test, test_predictions)
f1 = f1_score(y_test, test_predictions)
```

خروجی:

```
Accuracy: 0.9709090909090909
Precision: 0.9734513274336283
Recall: 0.9565217391304348
F1 Score: 0.9649122807017544
```

۴. نرمال‌سازی داده‌ها

اهمیت فرایند نرمال‌سازی داده‌ها: فرایند نرمال‌سازی داده‌ها یک مرحله مهم در پیش‌پردازش داده‌ها است که می‌تواند بهبودی چشمگیر در عملکرد مدل‌های یادگیری ماشین داشته باشد. در زیر تعدادی از اهمیت‌های فرایند نرمال‌سازی آورده شده است:

- تسهیل آموزش مدل: نرمال‌سازی داده‌ها می‌تواند فرآیند آموزش مدل را تسهیل کند. زمانی که داده‌ها در مقیاس‌ها واحدهای مختلفی باشند، ممکن است الگوریتم‌های یادگیری ماشین به مشکل بخورند و به سرعت به تناوب‌ها متمایل شوند. با نرمال‌سازی، مقیاس داده‌ها به یک مقیاس مشخص تغییر می‌کند و فرآیند یادگیری مدل پایدارتر می‌شود.
- پیشگیری از برازش بر روی داده‌های آموزش: نرمال‌سازی کمک می‌کند از برازش بر روی داده‌های آموزش جلوگیری کنیم. برازش بر روی داده‌های آموزش ممکن است منجر به یادگیری مدل از نویزهای غیرضروری یا مشکلات دیگر در داده‌ها شود.
- بهبود عملکرد در مسائل بهینه‌سازی: در الگوریتم‌های بهینه‌سازی مانند گرادیان کاهشی، نرمال‌سازی می‌تواند بهبودی در سرعت همگرایی و کارایی داشته باشد.

- پیشگیری از مشکل برخورداری از مقادیر پرت (Outliers): نرمال‌سازی به ما کمک می‌کند از مشکل برخورداری از مقادیر پرت در داده‌ها جلوگیری کنیم.
- بهبود توزیع داده‌ها: با نرمال‌سازی، توزیع داده‌ها به شکلی مرکزی تر و بهتر توزیع می‌شود که می‌تواند در بهبود عملکرد مدل‌ها مؤثر باشد.
- مقایسه قابلیت‌ها: با نرمال‌سازی، می‌توانیم بر اساس ویژگی‌های مشترک واحدی داده‌ها را مقایسه کنیم.

در کل، نرمال‌سازی به ما کمک می‌کند داده‌ها را به یک وضعیت مناسب تر برسانیم که باعث بهبود کارایی و پایداری مدل‌های یادگیری ماشین می‌شود.

روش اول: min max scaler

این روش بازه‌ای از مقادیر را به بازه‌ای معین می‌نماید، مثلاً از ۰ تا ۱. رابطه محاسبه برای این روش به صورت زیر است:

$$\text{Normalized Value} = \frac{\text{Original Value} - \text{Min}}{\text{Max} - \text{Min}}$$

```
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(X)
```

روش دوم: Z-score normalization

این روش مقادیر را به نحوی نرمال‌سازی می‌کند که میانگین مقدار صفر شود و انحراف معیار یک. رابطه محاسبه برای این روش به صورت زیر است:

$$\text{Standardized Value} = \frac{\text{Original Value} - \text{Mean}}{\text{Standard Deviation}}$$

```
scaler = StandardScaler()
standardized_data = scaler.fit_transform(X)
```

اطلاعات بخش ارزیابی نیز می‌تواند در فرآیند نرمال‌سازی داده‌ها مورد استفاده قرار گیرد. در اینجا چند نکته مربوط به استفاده از اطلاعات ارزیابی در فرآیند نرمال‌سازی آورده شده است:

- استفاده از آماره‌های ارزیابی برای تشخیص پرت‌ها: آماره‌های ارزیابی مانند میانگین (mean) و انحراف معیار (standard deviation) می‌توانند به شناسایی داده‌های پرت یا نقص در داده‌ها کمک کنند. اگر مشاهده شود که داده‌ها دارای انحراف معیار زیادی هستند یا میانگین غیرمعقولی دارند، ممکن است نیاز به پردازش و نرمال‌سازی دارند.

- تطبیق فرآیند نرمال‌سازی با توزیع داده‌ها: اطلاعات بخش ارزیابی ممکن است نشان دهد که توزیع داده‌ها به صورت غیرعادی (non-Gaussian) است. در چنین شرایطی، ممکن است استفاده از روش‌های نرمال‌سازی مبتنی بر میانگین و انحراف معیار موثر نباشد و نیاز به روش‌های دیگر مانند تبدیل‌های بازه‌ای (box-cox) یا تبدیل‌های یکسان‌سازی (quantile) باشد.
- تطبیق نحوه نرمال‌سازی با ماهیت داده‌ها: اطلاعات بخش ارزیابی می‌توانند نشان دهند که چه نوع نرمال‌سازی برای داده‌های مورد نظر مناسب‌تر است. به عنوان مثال، در صورتی که داده‌ها توزیع گسسته دارند، ممکن است استفاده از روش Min-Max Scaling کمک کند.

در کل، ارزیابی دقیق و شناخت دقیق از ویژگی‌ها و توزیع داده‌ها می‌تواند در انتخاب و اعمال بهترین فرآیند نرمال‌سازی بر اساس ویژگی‌ها و خصوصیات خاص داده‌ها کمک کند.

۵. تکرار قسمت‌های ۱ تا ۳ برای داده‌های نرمال‌شده

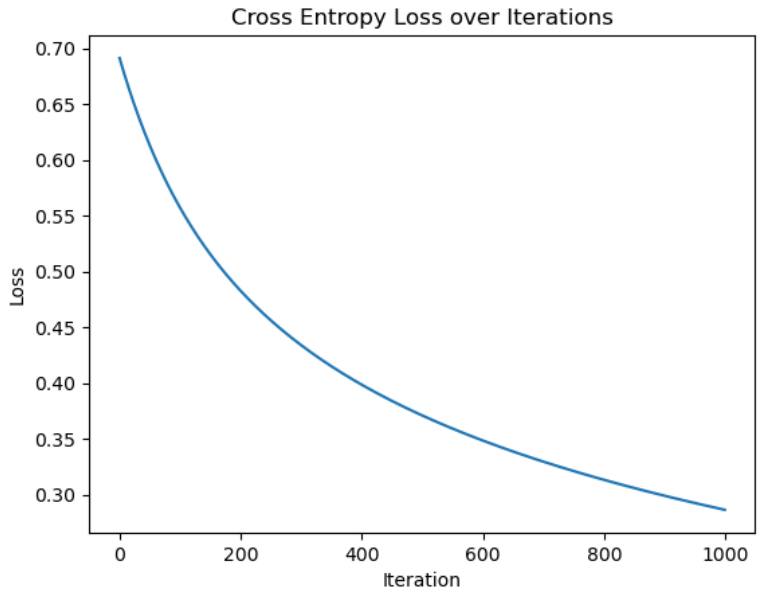
دیتای نرمال‌شده را به دیتافریم تبدیل کرده و ستون لیبل را به آن اضافه می کنیم.

```
df2 = pd.DataFrame(standardized_data)
dfnew = pd.concat([df2, y], axis=1)
```

سپس دیتاها را بر می‌زنیم و به دو قسمت آموزش و ارزیابی تقسیم می‌کنیم.

در بخش بعدی نیازی به تعریف مجدد توابع نیست چراکه قبلا تعریف شده اند. فقط استفاده از آن‌ها را مجددا انجام می‌دهیم. یعنی وارد قسمت مدل‌سازی می‌شویم:

نمودار تابع تلفات را رسم می‌کنیم:



در ادامه accuracy score و همچنین باقی معیارهای ارزیابی را نیز محاسبه می‌کنیم:

```
Accuracy: 0.9090909090909091
Precision: 0.8760330578512396
Recall: 0.9137931034482759
F1 Score: 0.8945147679324894
```

نتایج پیش‌بینی مدل برای چند نمونه:

ستون لیبل اصلی و لیبل پیش‌بینی شده را کنار هم قرار می‌دهیم:

```
ehem = pd.DataFrame(test_predictions)
ehem.reset_index(inplace = True, drop = True)
test = pd.DataFrame(y_test)
test.reset_index(inplace = True, drop = True)
dfshow = pd.concat([test, ehem], axis=1)
```

سپس ۵ سطر اول را مشاهده می‌کنیم:

```
dfshow.head()
```

	4	0
0	0	0
1	0	0
2	1	1
3	1	1
4	0	1

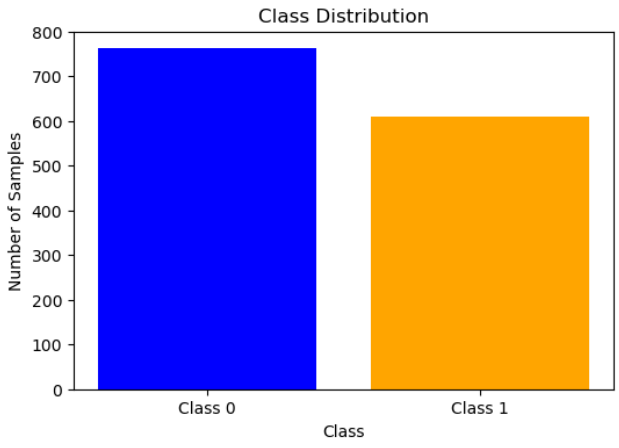
۶. وضعیت تعادل داده‌ها در دو کلاس موجود در دیتاست

شمارش تعداد هر کلاس:

```
class_counts = dfnew.iloc[:, 4:5].value_counts()
class_counts_index = [0, 1]
```

رسم نمودار:

```
plt.figure(figsize=(6, 4))
plt.bar(class_counts_index, class_counts.values, color=['blue', 'orange'])
plt.xlabel('Class')
plt.ylabel('Number of Samples')
plt.title('Class Distribution')
plt.xticks(class_counts_index, labels=['Class 0', 'Class 1'])
plt.show()
```



نمایش تعداد:

```
print('Class 0 samples:', class_counts[0])
print('Class 1 samples:', class_counts[1])
```

```
Class 0 samples: 762
Class 1 samples: 610
```

مشاهده شد که تعداد نمونه‌های کلاس‌ها با هم برابر نیستند.

عدم تعادل کلاس‌ها در دیتاست می‌تواند منجر به مشکلات مختلف در مسائل یادگیری ماشین شود. برخی از مشکلات ممکن عبارتند از:

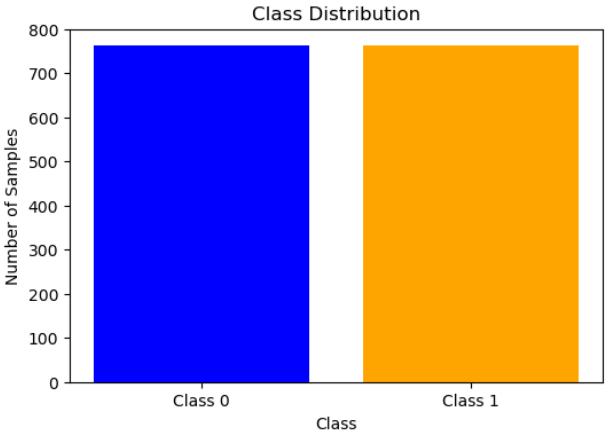
- تحت‌تخمین مدل: مدل‌های یادگیری ماشین معمولاً به تعداد نمونه‌های زیادی از هر کلاس نیاز دارند تا بتوانند به خوبی یاد بگیرند. در صورتی که یک کلاس تعداد نمونه کمتری داشته باشد، ممکن است مدل به نحوی تحت‌تخمین شده و دقت پایینی در پیش‌بینی‌های مربوط به آن کلاس داشته باشد.
- درخت‌های تصمیم و الگوریتم‌های مبتنی بر گرادیان: الگوریتم‌هایی مانند درخت تصمیم و الگوریتم‌های مبتنی بر گرادیان معمولاً حساس به توازن کلاس‌ها هستند. این الگوریتم‌ها ممکن است به سمت کلاس با تعداد نمونه بیشتر جلوگیری از تعادلی در کلاس‌ها نمایش دهند.
- درمان نادرست از مقادیر ارزیابی: در صورتی که دیتاست ناتوان در تعادل کلاس‌ها باشد، مقادیر ارزیابی مانند دقت (accuracy) تنها به خود کارآیی الگوریتم بر روی کل داده نگاه می‌کنند و این می‌تواند به نتایج نادرستی منجر شود. به عبارت دیگر، یک مدل می‌تواند با پیش‌بینی همیشه به یک کلاس (کلاس اکثریت)، دقت بالایی داشته باشد، اما این ممکن است توانایی پیش‌بینی کلاس‌های کمتری را نشان ندهد.
- سرریز (Overfitting) در کلاس اکثریت: در صورتی که کلاس اکثریت تعداد نمونه زیادی داشته باشد و کلاس کمیت اقلیت، ممکن است الگوریتم یادگیری به سمت کلاس اکثریت بیش‌اندازه برازش شود و باعث شود که مدل براحتی داده‌های کلاس اکثریت را پیش‌بینی کند اما نتواسته الگوهای عمومی را یاد بگیرد.

برای مقابله با این مشکلات، ممکن است اقداماتی نظیر افزایش تعداد نمونه‌های کلاس کمتر، استفاده از تکنیک‌های نمونه‌برداری متوازن، یا استفاده از معیارهای ارزیابی مناسب برای داده‌های ناتوازن انجام شود. یکی از تکنیک‌های معروف نمونه‌برداری متوازن به نام SMOTE (Synthetic Minority Over-sampling Technique) است. این تکنیک نمونه‌های جدید برای کلاس اقلیت (کمترین کلاس) ایجاد می‌کند تا تعادل در دیتاست ایجاد شود.

```
smote = SMOTE(sampling_strategy='auto')
X = dfnew.iloc[:, 0:4]
y = dfnew.iloc[:, 4:5]
X_resampled, y_resampled = smote.fit_resample(X, y)
df_resampled = pd.concat([pd.DataFrame(X_resampled), pd.DataFrame(y_resampled)], axis=1)
```

و در ادامه همان کدهارا برای بررسی تعادل میزنیم و نتیجه این می‌شود:

```
Class 0 samples: 762
Class 1 samples: 762
```



مشاهده می‌شود که داده‌ها متعادل شده‌اند.

۷. ارزیابی به وسیله یک طبقه بندی آماده پایتون: logistic regression

کدهای همانند سوال یک می‌باشد و برای جلوگیری از تکرار در گزارش آورده نمی‌شود.

معیارهای ارزیابی:

Accuracy: 0.8945454545454545
Precision: 0.9509803921568627
Recall: 0.8016528925619835
F1 Score: 0.8699551569506727

در این حالت هم مانند قبل چالش عدم تعادل داده‌ها با استفاده از کتابخانه آماده smote حل می‌شود.

سوال ۳

ابتدا کتابخانه‌های مورد نیاز در این سوال اضافه می‌شود:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression, SGDClassifier
import sklearn.metrics as met
from sklearn.metrics import hinge_loss, log_loss, accuracy_score, precision_score, recall_score, f1_score, mean_absolute_error, mean_squared_error, confusion_matrix
from sklearn.model_selection import train_test_split
from math import sqrt
```

۱. در این سوال با یک دیتاست مربوط به بیماری قلبی آشنایی به عمل می‌آید.

هدف و ویژگی‌های دیتاست: این مجموعه داده شامل شاخص‌های مختلف مرتبط با سلامت برای نمونه‌ای 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: ۰ = خیر، ۱ = بله).
- HighBP: وضعیت فشار خون بالا (باینری: ۰ = خیر، ۱ = بله).
- HighChol: وضعیت کلسترول بالا (دودویی: ۰ = خیر، ۱ = بله).
- CholCheck: دفعات بررسی کلسترول (طبقه ای).
- BMI: شاخص توده بدن (مستمر).
- سیگاری: وضعیت سیگار کشیدن (دودویی: ۰ = خیر، ۱ = بله).
- سکته مغزی: سابقه سکته مغزی (باینری: ۰ = خیر، ۱ = بله).
- دیابت: وضعیت دیابت (دودویی: ۰ = خیر، ۱ = بله).
- PhysActivity: سطح فعالیت بدنی (طبقه ای).
- میوه ها: فراوانی مصرف میوه (قسمتی).
- سبزیجات: فراوانی مصرف سبزیجات (قسمتی).
- HvyAlcoholConsump: وضعیت مصرف الکل سنگین (باینری: ۰ = خیر، ۱ = بله).
- AnyHealthcare: دسترسی به هر مراقبت بهداشتی (باینری: ۰ = خیر، ۱ = بله).
- NoDocbcCost: بدون پزشک به دلیل هزینه (باینری: ۰ = خیر، ۱ = 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: ۰ = خیر، ۱ = بله).
- جنسیت: جنسیت فرد (دودویی: ۰ = زن، ۱ = مرد).
- سن: سن فرد (مستمر).

- تحصیلات: مقطع تحصیلی (قسمتی).
- درآمد: سطح درآمد (مقوله ای).

این مجموعه داده حاوی انواع اطلاعات مرتبط با سلامت، عوامل سبک زندگی و اطلاعات جمعیتی برای گروهی از افراد است که آن را برای بررسی همبستگی‌ها و عوامل خطر بالقوه بیماری قلبی و سایر شرایط سلامتی مناسب می‌کند.

سپس دیتا با دستور gdown در محیط گوگل کلب وارد می‌شود. Id مد نظر با آپلود دیتا در گوگل درایو و برداشتن قسمتی از لینک آن بدست می‌آید.

https://drive.google.com/file/d/1N1FFek9iESUTxW3uIJTvYM55QKRunJZO/view?usp=sharing

```
pip install --no-cache-dir gdown
!gdown 1N1FFek9iESUTxW3uIJTvYM55QKRunJZO
```

۲. دیتاست به صورت یک دیتافریم دراورده می‌شود:

```
df = pd.read_csv('heart_disease_health_indicators.csv')
```

۱۰۰ نمونه داده مربوط به کلاس ۱ و ۱۰۰ نمونه مربوط به کلاس ۰ را جدا کرده و در دیتا فریم جدید قرار می‌دهیم:

```
class1 = df[df['HeartDiseaseorAttack'] == 1].sample(100 , replace=False)
class0 = df[df['HeartDiseaseorAttack'] == 0].sample(100 , replace=False)
data = pd.concat([class1, class0])
```

۳. تفکیک کلاس‌ها با استفاده از دو طبقه‌بندی آماده پایتون و در نظرگرفتن فراپارامترهای مناسب جداسازی داده آموزش و ارزیابی:

```
X = data.iloc[:, 1:22]
y = data.iloc[:, 0:1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=93)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

مدلسازی Logistic Regression:

از آن‌جایی که مدلی با تعداد نمونه محدود (۲۰۰) ولی ویژگی‌های زیاد (۲۲) داریم، گزینه‌های متفاوتی را می‌توان برای solver انتخاب کرد. مانند 'liblinear', 'newton-cg', 'lbfgs'. ما در اینجا برای نمونه lbfgs را انتخاب می‌کنیم.

حداکثر تکرار روی ۲۰۰۰ قرار داده می‌شود تا هم به حدی زیاد نباشد که ران شدن برنامه را کند کند، و هم به قدری پایین نباشد که مدل به خوبی عمل نکند.

از آن‌جایی که داده فقط ۲ کلاس دارد، در بخش multi_class گزینه one-vs-rest که دیفالت هم هست انتخاب می‌شود. اگر بیش از دو کلاس داشت، به جای آن multinomial قرار داده می‌شود.

در ادامه در قسمت penalty، مدل منظم‌سازی انتخاب می‌شود و lbfgs فقط l2 را قبول می‌کند و l2 به جلوگیری از overfit شدن کمک می‌کند. لذا این گزینه انتخاب می‌شود.

همچنین، random state برابر با دو رقم آخر شماره دانشجویی بنده قرار داده شده است.

مدلسازی:

```
model1 = LogisticRegression(solver='lbfgs', max_iter=2000, multi_class='ovr', penalty='l2', random_state=93)
model1.fit(X_train, y_train);
```

پیش‌بینی:

```
ypred1 = model1.predict(X_test)
ypred1_2 = model1.predict_proba(X_test)
ypred1_3 = model1.predict_log_proba(X_test)
```

درصد درستی:

```
a = model1.score(X_train, y_train)
b = model1.score(X_test, y_test)
```

خطاها:

```
c = met.mean_squared_error(y_test, ypred1)     # MSE
d = mean_absolute_error(y_test, ypred1)        # MAE
e = sqrt(mean_squared_error(y_test, ypred1))   # SMSE
```

ارزیابی:

```
accuracy = accuracy_score(y_test, ypred1)
precision = precision_score(y_test, ypred1)
recall = recall_score(y_test, ypred1)
f1 = f1_score(y_test, ypred1)
```

ماتریس درهم ریختگی:

```
f=confusion_matrix(y_test, ypred1)
```

چاپ خروجی‌ها:

```
print(f'Accuracy for train: {a}')
print(f'Accuracy for test: {b}')
print(f'mean squared error: {c}')
print(f'mean absolute error: {d}')
print(f'sqrt mean squared error: {e}')
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print(f'confusion matrix: {f}')
```

نتایج:

```
Accuracy for train: 0.8
Accuracy for test: 0.675
mean squared error: 0.325
mean absolute error: 0.325
sqrt mean squared error: 0.570087712549569
Accuracy: 0.675
Precision: 0.75
Recall: 0.6521739130434783
F1 Score: 0.6976744186046512
confusion matrix: [[12  5]
                   [ 8 15]]
```

مدلسازی SGD Classifier:

در قسمت انتخاب loss، چون log loss برای کلاس‌های ۰ و ۱ اعمال می‌شود، و داده هم فقط ۲ کلاس دارد، این گزینه انتخاب می‌شود.

همچنین، random state برابر با دو رقم آخر شماره دانشجویی بنده قرار داده شده است.

در اینجا تنها مدلسازی با رگرسیون لجستیک متفاوت است و باقی کد به همان صورت است.

```
model2 = SGDClassifier(loss='log_loss', random_state=93)
```

نتایج:

```
Accuracy for train: 0.675
Accuracy for test: 0.5
mean squared error: 0.5
mean absolute error: 0.5
sqrt mean squared error: 0.7071067811865476
Accuracy: 0.5
Precision: 0.6363636363636364
Recall: 0.30434782608695654
F1 Score: 0.411764705882353
confusion matrix: [[13  4[
                   [16  7]]
```

۴. بدست آوردن تابع اتلاف با استفاده از دستورات آماده سایکیت‌لرن

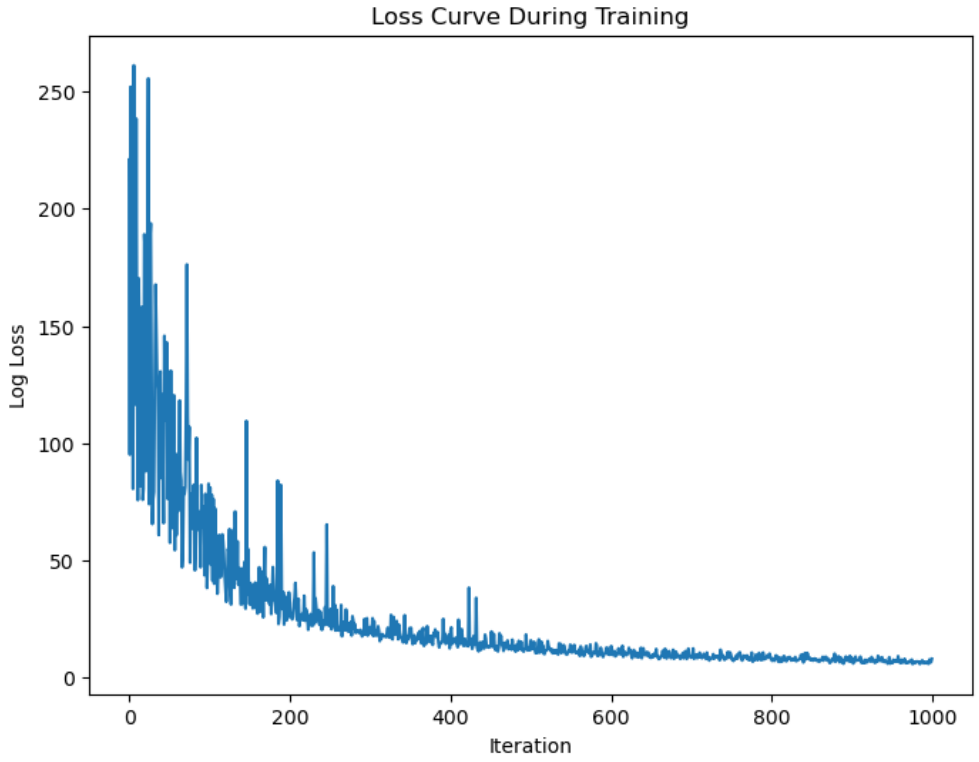
در مدل رگرسیون لجستیک اتلاف را می‌توان با این کد بدست آورد:

```
y_prob1 = model1.predict_proba(X_test)
loss = log_loss(y_test, y_prob1)
print("Log Loss:", loss)
Log Loss: 0.6011398384500823
```

در مدل SGD Classifier نمودار اتلاف را می‌توان با این کد بدست آورد:

```
loss_history = []
for epoch in range(1000):
   model2.partial_fit(X_train, y_train, classes=np.unique(y))
   decision_values = model2.decision_function(X_train)
   loss = hinge_loss(y_train, decision_values)
   loss_history.append(loss)

plt.figure(figsize=(8, 6))
plt.plot(loss_history)
plt.title('Loss Curve During Training')
plt.xlabel('Iteration')
plt.ylabel('Log Loss')
plt.show()
```



۵. محاسبه شاخصه‌های ارزیابی دیگر غیر از accuracy

در اینجا می‌توان همان شاخص‌هایی را که در قسمت قبل بدست آمد را نام برد خروجی‌شان پیشتر آورده شد و در ادامه در مورد هر یک از آن‌ها توضیحاتی داده می‌شود.

- ماتریس درهم‌ریختیگی (confusion matrix): این ماتریس یک روش اندازه گیری عملکرد برای مساله طبقه بندی یادگیری ماشین است که در آن خروجی می تواند دو یا چند کلاس باشد. این جدول با ۴ ترکیب مختلف از مقادیر پیش بینی شده و واقعی ساخته می شود.

Predicted Values	Actual Values: Positive (1)	Actual Values: Negative (0)
Positive (1)	TP	FP
Negative (0)	FN	TN

مثبت صحیح (True Positives) : پیش بینی کردیم که مثبت بوده و درست پیش بنی کردیم

منفی صحیح (True Negatives): پیش بینی کردیم منفی باشد و درست پیش بینی کردیم.

مثبت کاذب (False Positives): پیش بینی کردیم مثبت باشد ولی غلط پیش بینی کردیم

منفی کاذب (False Negatives): پیش بینی کردیم غلط باشد ولی درست بود.

- فراخوانی (Recall): به این مفهوم اشاره دارد که از بین همه کلاسهای مثبت ، چقدر درست پیش بینی کردیم. باید تا حد ممکن بالا باشد.

$$Recall = \frac{TP}{TP + FN}$$

- صحت (Precision): یعنی از بین تمام کلاسهای مثبتی که به طور صحیح پیش بینی کرده ایم ، چند نفر در واقع مثبت هستند.

$$Precision = \frac{TP}{TP + FP}$$

- معیار F1 (f1 score): اگر بخواهیم همزمان هر دو معیار صحت و فراخوانی در ارزیابی مدل دخیل باشند از این معیار استفاده می کنیم.

$$F - measure = \frac{2*Recall*Precision}{Recall + Precision}$$