به نام خدا

نام ونام خانوادگی:

محمد حسین بیاتی

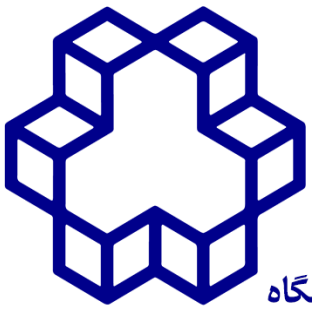
شماره دانشجویی:

۹۹۲۴۶۹۳

گزارش کار مینی پروژه شماره یک-بخش اول

استاد درس:

دکتر مهدی علیاری



دانشگاه
خواجه نصیرالدین طوسی
K. N. Toosi University
of Technology

فهرست مطالب

سوال اول

بخش ۱

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

دیتاست خواسته شده را به کمک make-classification ایجاد می کنیم.

همچنین برای نشان دادن داده ها نیاز به رسم نمودار داریم پس از کتابخانه Matplotlib استفاده می کنیم:

برای تولید این داده ها کافیست n_samples را برابر با ۱۰۰۰ قرار دهیم تا دیتاست مذکور دارای ۱۰۰۰ نمونه باشد، همچنین برای ایجاد ۲ کلاس مقدار n_classes را برابر ۲ قرار می دهیم و همچنین برای ایجاد ۲ ویژگی نیز n_features را برابر با ۲ قرار می دهیم، بنابراین کد ما به شکل زیر خواهد بود:

همچنین مقدار random_satate را برابر با ۹۳ قرار می دهیم که دو رقم آخر شماره دانشجویی است، زمانی که یک مقدار خاص به random_state اختصاص می دهیم، به این معنا خواهد بود که هر بار که از این کد استفاده می کنیم، دیتاها و مقادیر تصادفی مشخصی تولید می شوند، از این زمانی استفاده می شود که نیاز است تا داده های تصادفی یا نتایج آزمایشات تصادفی قابل تکرار باشد، زیرا زمانی که random_state تنظیم نشود، هر بار که کد اجرا شود، نتایج و داده ها تغییر می کنند واین موضوع زمانی تکرار پذیری نتایج مهم باشد مشکل ساز خواهد شد. بنابراین random_state امکان تکرار پذیری نتایج را فراهم خواهد کرد.

بنابراین کد ما به شکل زیر خواهد بود:

Dataset

```
import matplotlib.pyplot as plt

from sklearn.datasets import make_classification

# Create the dataset
X, y = make_classification(n_samples=1000, n_features=2, n_classes=2,
random_state=93)
```

با اجرا کردن این کد به ارور زیر بر خواهیم خورد:

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-e653a14a3a18> in <cell line: 4>()
      2
      3 # Create the dataset
----> 4 X, y = make_classification(n_samples=1000, n_features=2,
n_classes=2, random_state=42)
      5
      6 # Display the dimensions of the dataset

/usr/local/lib/python3.10/dist-
packages/sklearn/datasets/_samples_generator.py in
make_classification(n_samples, n_features, n_informative, n_redundant,
n_repeated, n_classes, n_clusters_per_class, weights, flip_y, class_sep,
hypercube, shift, scale, shuffle, random_state)
    176     # Count features, clusters and samples
    177     if n_informative + n_redundant + n_repeated > n_features:
--> 178         raise ValueError(
    179             "Number of informative, redundant and repeated "
    180             "features must sum to less than the number of total"

ValueError: Number of informative, redundant and repeated features must sum
to less than the number of total features
```

که طبق آن سه پارامتر دیگر به نام های n_informative و n_redundant و n_reapeted اهمیت پیدا خواهند کرد که نقش هر کدام به شکل زیر است:

- n_informative تعداد پارامترهای اطلاعات زا هستند که برای تفکیک بین کلاس ها استفاده می شوند، تنظیم این پارامتر به مقداری کمتر از n_features به این معناست که برخی از ویژگی ها اطلاعات زا نیستند و به تفکیک بین کلاس ها کمکی نمی کنند.
- n_redundant تعداد ویژگی های تکراری را مشخص می کند که این ویژگی ها به عنوان ترکیب خطی از ویژگی های دیگر هستند.
- n_repeated تعداد ویژگی های تکرار شده را نشان می دهد، این ویژگی ها کپی های اضافی از ویژگی های اطلاعات زا و یا ویژگی های تکراری هستند.

بنابراین مشخص است که باید:

$$n_{informative} + n_{redundant} + n_{repeated} \leq n_{features}$$

زمانی که هیچ اطلاعاتی راجع به n_informative و n_redundant و n_repeated ندهیم به طور default خواهیم داشت:

n_informative = n_redundant = 2

n_repeated = 0

پس از آنجا که تعداد ویژگی های ما یعنی n_features برابر با ۲ است پس رابطه

$$n_{informative} + n_{redundant} + n_{repeated} \leq n_{features}$$

برقرار نخواهد بود، پس نیاز است تا ما اطلاعات لازم را راجع به n_informative و n_redundant و n_repeated بدهیم.

ما در نظر می گیریم که هر دو ویژگی مذکور اطلاعات زا هستند و تکراری نیستند.

از طرفی می توانیم اطلاعات دیگر همانند تعداد شکل های داده های در هر کلاس را به کمک n_clusters_per_class تغییر بدهیم که برای سادگی در نظر می گیریم که داده های در هر کلاس تنها از یک نوع هستند پس n_clusters_per_class=1 ، همچنین برای تعیین میزان تفکیک پذیری کلاس ها از هم از پارامتر class_sep استفاده می کنیم که هر چه کمتر باشد کلاس ها بیشتر در هم فرو رفته اند، این پارامترها را نیز همان default یعنی برابر با ۱ قرار می دهیم.

پس کد ما به شکل زیر تغییر خواهد بود:

```
# Create the dataset
X, y = make_classification(
    n_samples=1000,
    n_features=2,
    n_informative=2,
    n_redundant=0,
    n_repeated=0,
    n_clusters_per_class=1,
    class_sep=1,
    n_classes=2,
    random_state=93
    )
```

که در آن X همان ویژگی های ما است که انتظار داریم یک آرایه ۱۰۰۰ در ۲ باشد زیرا ۱۰۰۰ نمونه داشتیم و هر نمونه دارای دو ویژگی است و y همان هدف های ما(target) است که به صورت label زده می باشد. بنابراین y باید یک آرایه ۱۰۰۰ در ۱ باشد.

برای بررسی این موضوع می توانیم ابعاد X و y به صورت زیر بررسی کنیم:

```
# Display the dimensions of the dataset
print(f'Dimensions of the features: {X.shape}')
print(f'Dimensions of the labels: {y.shape}')
```

نتایج به صورت زیر خواهد بود:

```
Dimensions of the features: (1000, 2)
Dimensions of the target: (1000,)
```

که همان نتایج مورد انتظار ما است.

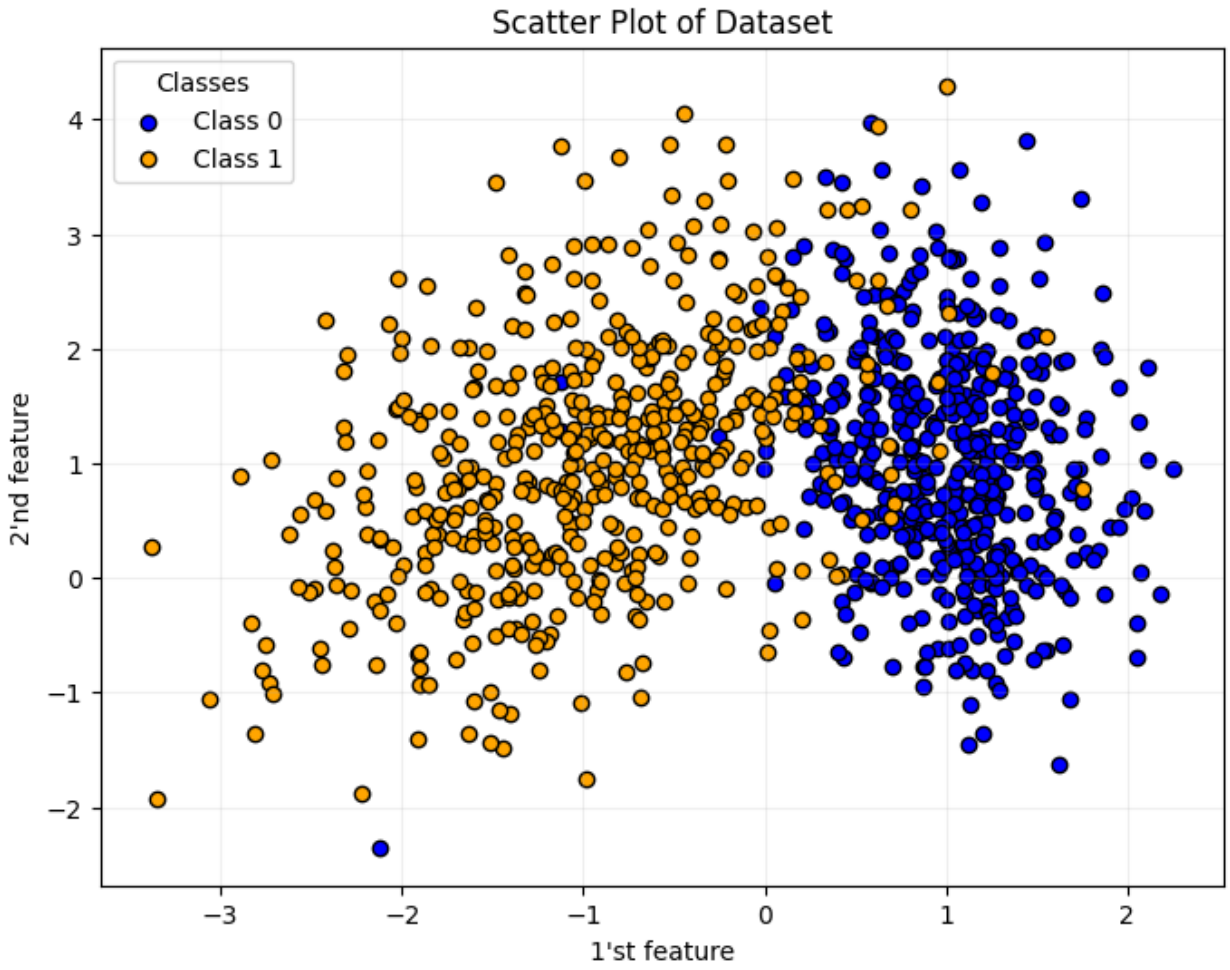
برای رسم نمودار scatter داده ها نیز به شکل زیر عمل می کنیم:

ابتدا ابعاد نمودار مورد نظر را تعیین می کنیم، سپس داده های مربوط به کلاس صفر یعنی داده هایی که target مورد نظر برای آنها صفر بوده را رسم می کنیم و سپس داده های کلاس ۱ را رسم می کنیم(در هر مورد شکل مناسب و رنگ مناسب در نظر می گیریم.)

همچنین محور ها را نامگذاری کرده و عنوان مناسبی برای محور ها وهمچنین خود نمودار در نظر می گیریم. سپس علامت خطوط grid را فعال کرده و همچنین legend را برای مشخص بودن داده های هر دسته فعال می کنیم، پس کد به شکل زیر خواهد بود:

```
# Create a scatter plot
plt.figure(figsize=(8, 6))  # Set the figure size
scatter_class_0 = plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue',
label='Class 0', edgecolors='k', marker='o')
scatter_class_1 = plt.scatter(X[y == 1, 0], X[y == 1, 1], color='orange',
label='Class 1', edgecolors='k', marker='o')
plt.xlabel("1'st feature")
plt.ylabel("2'nd feature")
plt.title('Scatter Plot of Dataset')  # Title for the plot
plt.legend(handles=[scatter_class_0,                    scatter_class_1],
title='Classes',loc="upper left")
plt.grid(alpha=0.2)  # Display grid lines
```

نتایج :



با هر بار اجرای کد باز هم به نتایج مشابهی می رسیم (به علت استفاده از random_state).

همچنین می توان X که همان ویژگی های ما است و همچنین y را با یکبار ران کردن کد دید.

بخش ۲

۲- با استفاده از حداقل دو طبقه بند آماده پایتون و در نظر گرفتن فراپارامترهای مناسب، دو کلاس موجود در دیتا ست قبلی را از هم تفکیک کنید. ضمن توضیح روند انتخاب فراپارامترها (مانند تعداد دوره آموزش و نرخ یادگیری)، نتیجه دقت آموزش و ارزیابی را نمایش دهید. برای بهبود از چه تکنیک هایی استفاده کردید.

برای طبقه بندی داده ها از linear classifier ها استفاده می کنیم که در درس نیز توضیح داده شد، استفاده می کنیم، در این بخش از SGD Classifier و Logistic Regression و Perceptron استفاده می کنیم.

همچنین نیاز است که داده های train و test را جدا کنیم، بنابراین کافیست از بخش model selection از train_test_split استفاده کنیم.

بنابراین برای فراخوانی این کتابخانه ها به شکل زیر عمل می کنیم:

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

همچنین باید داده های train و test را مشخص کنیم برای اینکار ۸۰ درصد داده ها را برای train انتخاب می کنیم و ۲۰ درصد باقی مانده برای test مورد استفاده قرار می گیرند پس باید test_size را برابر۰.۲ قرار دهیم تا ۲۰ درصد داده ها برای تست انتخاب شوند، از طرفی برای اینکه همانند بخش الف تکرار پذیری نتایج ایجاد شود باید یه عدد به random_state اختصاص داده شود که این عدد را برابر ۹۳(دو رقم آخر شماره دانشجویی) قرار می دهیم، پس:

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=93)
```

انتظار داریم X_train شامل ۸۰ درصد نمونه ها باشد یعنی یک آرایه ۸۰۰ در ۲ باشد y_train نیز یک آرایه ۸۰۰ در ۱ باشد. از طرفی باید X_test شامل ۲۰ درصد نمونه ها باشد یعنی یک آرایه ۲۰۰ در ۲ باشد y_test نیز یک آرایه ۲۰۰ در ۱ باشد.

برای اینکه ابعاد آرایه های ویژگی و همچنین target در داده های train و test مشخص شود، به صورت زیر عمل می کنیم:

```
# Display the dimensions of the training and testing sets
print(f'Dimensions of the training features: {X_train.shape}')
```

```
print(f'Dimensions of the training target: {y_train.shape}')
print(f'Dimensions of the testing features: {X_test.shape}')
print(f'Dimensions of the testing target: {y_test.shape}')
```

نتایج:

```
Dimensions of the training features: (800, 2)
Dimensions of the training target: (800,)
Dimensions of the testing features: (200, 2)
Dimensions of the testing target: (200,)
```

که همان نتایج مورد انتظار ماست.

در هر کدام از روش طبقه بندی نیاز است تا مدل را ابتدا مشخص کنیم و پارامترهای هر مدل را نیز مشخص کنیم، سپس این مدل را بر داده های train فیت کنیم و سپس accuracy را می توانیم هم برای دادهای test و هم برای داده های train محاسبه کنیم.

همچنین در هر مدل می توانیم پیش بینی سیستم را از داده های test ببینیم و با مقدار هدف مقایسه کنیم.

روش Logistic Regression:

برای این روش همانند توضیحات بالا عمل می کنیم، فقط کافیست پارامترهای آن را تعیین کنیم.

از بین تمام پارامتر های آن موارد زیر را تعیین خواهیم کرد:

- ❖ پارامتر C :

این پارامتر، یک پارامتر مثبت است. مقدار بزرگتر از C به آن معناست که تاثیر عامل مقدار مطلق وزن ها کم است، این موضوع سبب افزایش توانایی ما در مدل سازی نمونه های پیچیده شده و باعث یادگیری الگوهای نویزی خواهد شد. مقدار پیش فرض آن برابر با ۱ است.

- ❖ پارامتر penalty :

این پارامتر نوع تابع جریمه مورد استفاده در موضوع بهینه سازی را تعیین می کند که می تواند یکی از موارد "None","elasticnet","l2","l1" را اخذ کند. اگر 'l1' انتخاب شود، از تابع جریمه L1 (Lasso) برای کم کردن وزن‌ها استفاده می‌شود. اگر 'l2' انتخاب شود، از تابع جریمه L2 (Ridge) برای کم کردن وزن‌ها استفاده می‌شود. اگر 'elasticnet' انتخاب شود، ترکیبی از L1 و L2 به عنوان تابع جریمه مورد استفاده قرار می‌گیرد. اگر None انتخاب شود، تابع جریمه استفاده نمی‌شود. مقدار پیشفرض penalty برابر با 'l2' است.

- ❖ پارامتر solver :

این پارامتر الگوریتم بهینه سازی مورد استفاده برای یافتن وزن های مدل را مشخص می کند که شامل "newton-cg"،"lbfgs"،"liblinear"،"sag"،"saga" می باشد. "liblinear" مناسب برای مجموعه داده های کوچک و متوازن است، در حالی که "sag"و"saga" برای مجموعه داده های بزرگ مناسب هستند. پیش فرض این پارامتر "lbfgs" است.

با توجه به توضیحات بالا و مجموعه داده ی ما پارامترها را به صورت زیر تعیین می کنیم، از طرفی برای اینکه همانند بخش الف تکرار پذیری نتایج ایجاد شود باید یه عدد به random_state اختصاص داده شود که این عدد را برابر ۹۳(دو رقم آخر شماره دانشجویی) قرار می دهیم، پس:

```
LogReg_classifier = LogisticRegression(C=1.0, penalty='l2', solver='sag',
max_iter=20000, random_state=93)
LogReg_classifier.fit(X_train, y_train)
y_pred = LogReg_classifier.predict(X_test)
LogReg_train_accuracy = LogReg_classifier.score(X_train, y_train)
LogReg_test_accuracy = LogReg_classifier.score(X_test, y_test)
```

همچنین برای مشاهده prediction و هدف اصلی به صورت زیر عمل می کنیم:

```
y_pred, y_test
```

نتایج:

```
(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, 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]))
```

روش SGD Classifier:

برای این روش همانند توضیحات قبل عمل می کنیم، فقط کافیست پارامترهای آن را تعیین کنیم.

از بین تمام پارامتر های آن موارد زیر را تعیین خواهیم کرد:

- پارامتر max_iter :

این پارامتر، حداکثر تعداد دوره های آموزش را مشخص می کند و اگر پس از این تعداد دوره آموزش به همگرایی نرسیده باشیم این موضوع اعلام خواهد شد.

- پارامتر penalty :

این پارامتر نوع تابع جریمه مورد استفاده در موضوع بهینه سازی را تعیین می کند که می تواند یکی از موارد "None","elasticnet","l2","l1" را اخذ کند. اگر 'l1' انتخاب شود، از تابع جریمه L1 (Lasso) برای کم کردن وزن‌ها استفاده می‌شود. اگر 'l2' انتخاب شود، از تابع جریمه L2 (Ridge) برای کم کردن وزن‌ها استفاده می‌شود. اگر 'elasticnet' انتخاب شود، ترکیبی از L1 و L2 به عنوان تابع جریمه مورد استفاده قرار می‌گیرد. اگر None انتخاب شود، تابع جریمه استفاده نمی‌شود. مقدار پیشفرض penalty برابر با 'l2' است.

- پارامتر loss :

این پارامتر نوع تابع هدف که کمینه کردن آن در موضوع بهینه سازی مورد نظر است را مشخص می کند.

- پارامتر alpha :

این پارامتر نرخ یادگری را مشخص می کند.

با توجه به توضیحات بالا و مجموعه داده ی ما پارامترها را به صورت زیر تعیین می کنیم، از طرفی برای اینکه همانند بخش الف تکرار پذیری نتایج ایجاد شود باید یه عدد به random_state اختصاص داده شود که این عدد را برابر ۹۳(دو رقم آخر شماره دانشجویی) قرار می دهیم، پس:

```
SGD_classifier = SGDClassifier(alpha=0.01, loss='log_loss', max_iter=20000,
penalty='l2', random_state=93)
SGD_classifier.fit(X_train, y_train)
y_pred1 = SGD_classifier.predict(X_test)
SGD_train_accuracy = SGD_classifier.score(X_train, y_train)
SGD_test_accuracy = SGD_classifier.score(X_test, y_test)
```

مشاهده prediction و هدف اصلی به صورت زیر عمل می کنیم:

```
y_pred1, y_test
```

نتایج:

```
(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, 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]))
```

روش Perceptron:

برای این روش همانند توضیحات قبل عمل می کنیم، فقط کافیست پارامترهای آن را تعیین کنیم.

از بین تمام پارامتر های آن موارد زیر را تعیین خواهیم کرد:

- ❖ پارامتر max_iter :

این پارامتر، حداکثر تعداد دوره های آموزش را مشخص می کند و اگر پس از این تعداد دوره آموزش به همگرایی نرسیده باشیم این موضوع اعلام خواهد شد.

- ❖ پارامتر penalty :

این پارامتر نوع تابع جریمه مورد استفاده در موضوع بهینه سازی را تعیین می کند که می تواند یکی از موارد "None","elasticnet","l2","l1" را اخذ کند. اگر 'l1' انتخاب شود، از تابع جریمه L1 (Lasso) برای کم کردن وزن‌ها استفاده می‌شود. اگر 'l2' انتخاب شود، از تابع جریمه L2 (Ridge) برای کم کردن وزن‌ها استفاده می‌شود. اگر 'elasticnet' انتخاب شود، ترکیبی از L1 و L2 به عنوان تابع جریمه مورد استفاده قرار می‌گیرد. اگر None انتخاب شود، تابع جریمه استفاده نمی‌شود. مقدار پیشفرض penalty برابر با 'l2' است.

- ❖ پارامتر alpha :

این پارامتر نرخ یادگری را مشخص می کند.

با توجه به توضیحات بالا و مجموعه داده ی ما پارامترها را به صورت زیر تعیین می کنیم، از طرفی برای اینکه همانند بخش الف تکرار پذیری نتایج ایجاد شود باید یه عدد به random_state اختصاص داده شود که این عدد را برابر ۹۳(دو رقم آخر شماره دانشجویی) قرار می دهیم، پس:

```
perceptron_classifier = Perceptron(alpha=0.01, penalty='l2',
max_iter=20000, random_state=93)
perceptron_classifier.fit(X_train, y_train)
y_pred2 = perceptron_classifier.predict(X_test)
perceptron_train_accuracy = perceptron_classifier.score(X_train, y_train)
perceptron_test_accuracy = perceptron_classifier.score(X_test, y_test)
```

مشاهده prediction و هدف اصلی به صورت زیر عمل می کنیم:

```
y_pred2, y_test
```

نتایج:

```
(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, 0, 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, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1,
0, 0, 0, 0, 1, 0, 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, 0, 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, 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]))
```

بررسی دقت در دادهای test و train:

به منظور نمایش دقت در این داده ها به صورت زیر عمل می کنیم:

```
print(f"Logistic Regression Train Accuracy: {LogReg_train_accuracy:.2f}")
print(f"Logistic Regression Test Accuracy: {LogReg_test_accuracy:.2f}")

print(f"SGD Classifier Train Accuracy: {SGD_train_accuracy:.2f}")
print(f"SGD Classifier Test Accuracy: {SGD_test_accuracy:.2f}")

print(f"Perceptron Train Accuracy: {perceptron_train_accuracy:.2f}")
print(f"Perceptron Test Accuracy: {perceptron_test_accuracy:.2f}")
```

نتایج:

```
Logistic Regression Train Accuracy: 0.95
Logistic Regression Test Accuracy: 0.95
SGD Classifier Train Accuracy: 0.94
```

```
SGD Classifier Test Accuracy: 0.95
Perceptron Train Accuracy: 0.94
Perceptron Test Accuracy: 0.94
```

برای اینکه نتایج بهبود پیدا کند پارامتر ها را به صورت دقیق تعیین می کنیم و با تغییر آنها به بهترین نتیجه ممکن می رسیم(tune کردن فراپارمترها)، از طرفی می توان زمانی که آموزش به همگرایی نرسیده است حداکثر دوره های همگرایی را افزایش داد و همچنین روش های بهینه سازی را تغییر داد.

یکی دیگر از پارامترهای مهم نرخ آموزش است که تعیین آن اهمیت دارد، همچنین برای بهبود نتایج می توان پارامترهای بیشتری را دخیل کرد و یا از سایر classifier ها و حتی classifier های غیر خطی استفاده کرد.

همچنین یک روش دیگر برای افزایش دقت این است که تعداد داده های یکسانی از هر دو کلاس برای تست برداشته شود که این کار به کمک کد زیر قابل پیاده سازی است:

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=93, stratify=y)
```

نتایج پس از انجام این کار:

```
Logistic Regression Train Accuracy: 0.94
Logistic Regression Test Accuracy: 0.97
SGD Classifier Train Accuracy: 0.94
SGD Classifier Test Accuracy: 0.96
Perceptron Train Accuracy: 0.93
Perceptron Test Accuracy: 0.95
```

همانطور که مشخص است دقت پس از انجام این کار افزایش یافته است.

همچنین نرمال کردن داده ها نیز می تواند سودمند باشد که در بخش های بعد تاثیر آن را می بینیم.

بخش ۳

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

برای رسم نمودارهای خواسته شده از روش دوم معرفی شده در کلاس یعنی استفاده از plot_decision_regions استفاده می کنیم.

برای مشخص کردن داده هایی که به اشتباه طبقه بندی شده اند نیز داده های test که مقدار prediction آنها با target آنها متفاوت است را در دسته داده هایی که اشتباه طبقه بندی شده اند قرار می دهیم، همچنین داده های train که در هر روش درست دسته بندی نشده اند را در این دسته داده ها قرار می دهیم.

سپس نمودار دسته بندی داده ها را به کمک plot_decision_regions همانطور که در کلاس گفته شد رسم می کنیم.

```
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
```

همچنین محور ها را نامگذاری کرده و عنوان مناسبی برای محور ها و همچنین خود نمودار در نظر می گیریم. سپس legend را برای مشخص بودن داده های هر دسته فعال می کنیم، پس کد به شکل زیر خواهد بود:

دز این روش همه ی داده ها چه test و چه train نشان داده خواهند شد و اگر داده ای به اشتباه در یک دسته قرار گرفته باشد یک علامت بر روی زده خواهد شد.

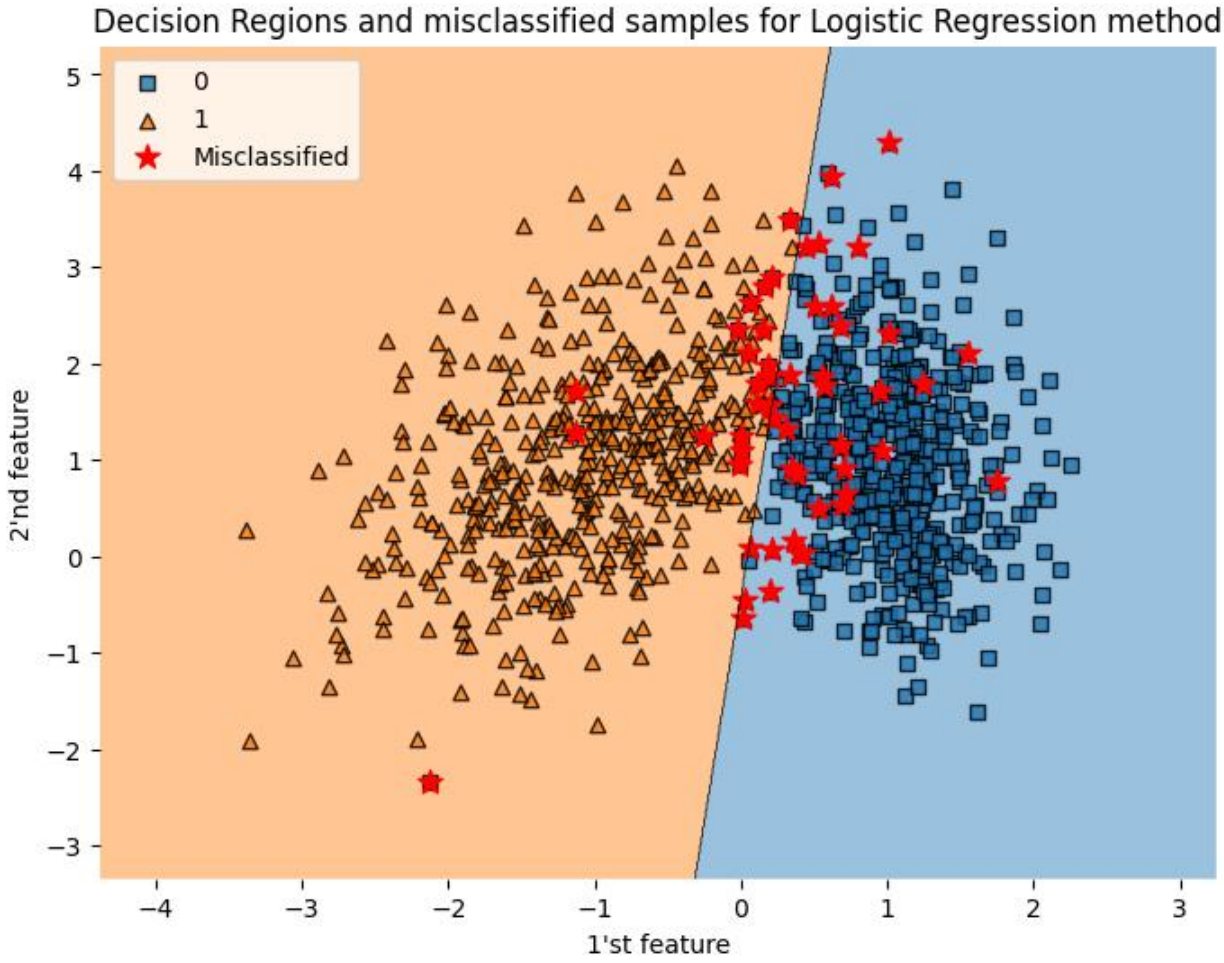
طبقه بندی به روش Logistic Regression :

```
plt.figure(figsize=(8,6))
plot_decision_regions(X, y, clf=LogReg_classifier, legend=2)

# misclassified samples
misclassified = X[y != LogReg_classifier.predict(X)]
plt.scatter(misclassified[:, 0], misclassified[:, 1], c='red', marker='*',
s=100, label='Misclassified')

# label and title
plt.xlabel("1'st feature")
plt.ylabel("2'nd feature")
plt.title('Decision Regions and misclassified samples for Logistic
Regression method')
plt.legend(loc="upper left")
```

نتایج:



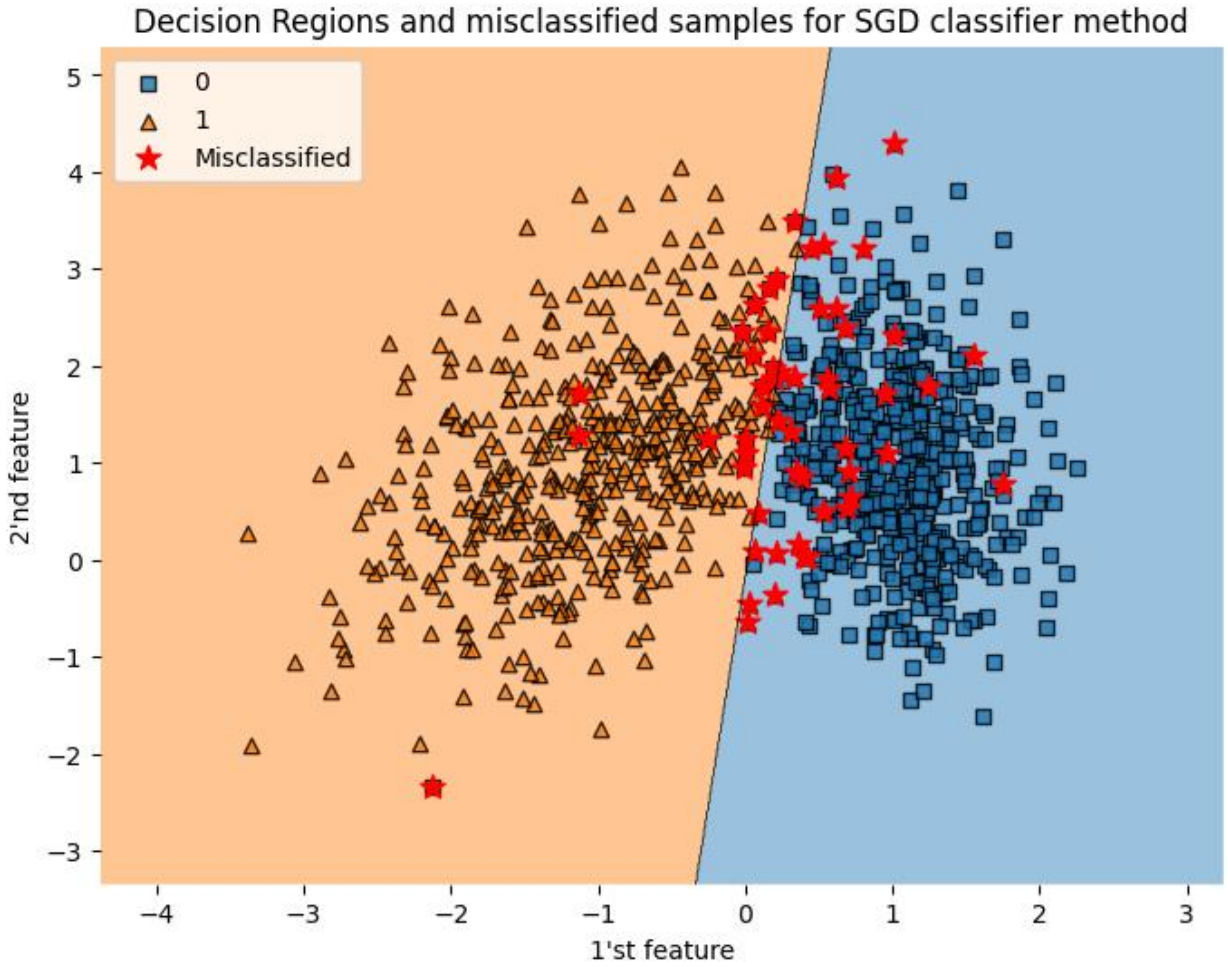
طبقه بندی به روش SGD Classifier:

```
plt.figure(figsize=(8,6))
plot_decision_regions(X, y, clf=SGD_classifier, legend=2)

# misclassified samples
misclassified = X[y != SGD_classifier.predict(X)]
plt.scatter(misclassified[:, 0], misclassified[:, 1], c='red', marker='*',
s=100, label='Misclassified')

# label and title
plt.xlabel("1'st feature")
plt.ylabel("2'nd feature")
plt.title('Decision Regions and misclassified samples for SGD classifier
method')
plt.legend(loc="upper left")
```

نتایج:



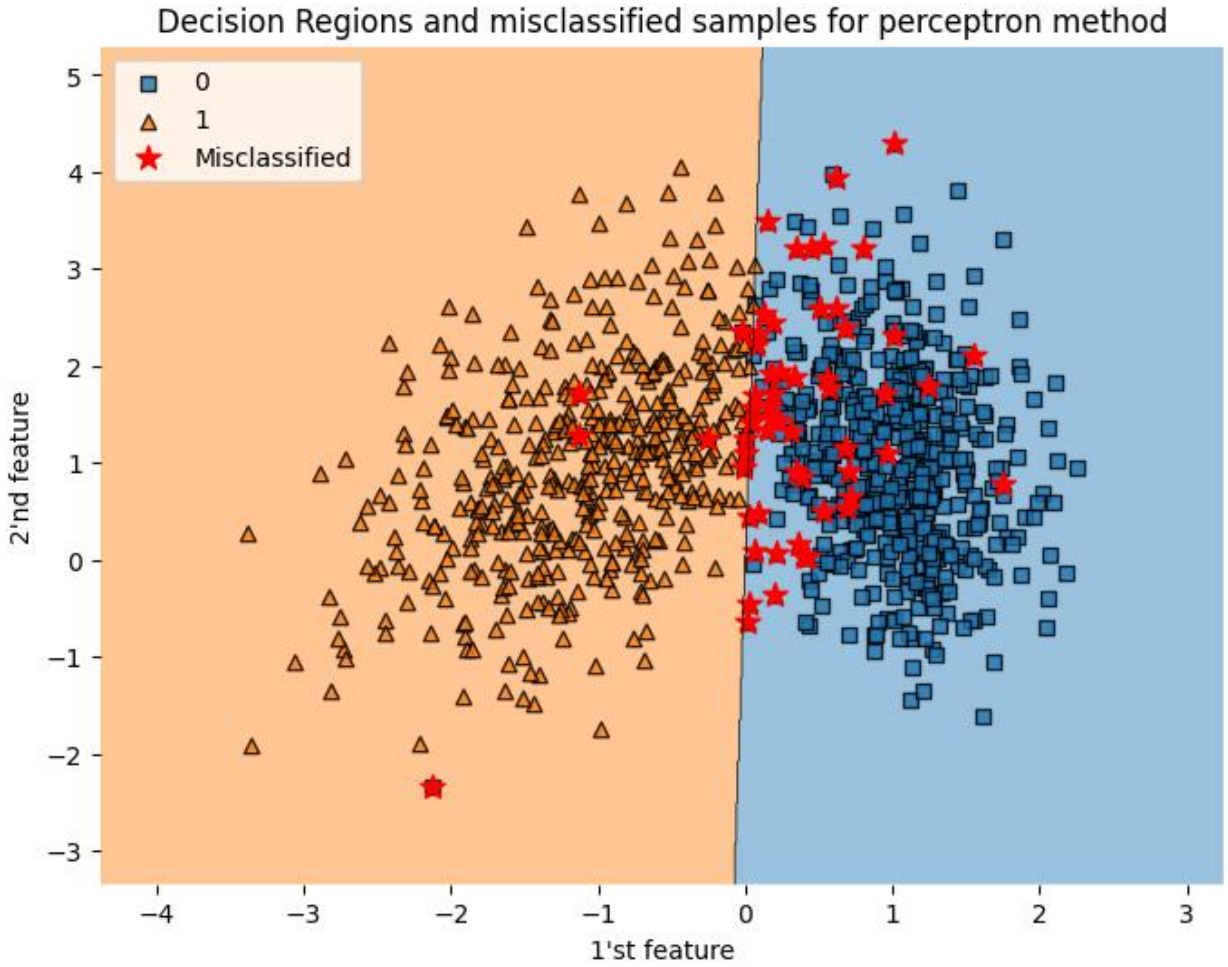
طبقه بندی یه روش Perceptron:

```
plt.figure(figsize=(8,6))
plot_decision_regions(X, y, clf=perceptron_classifier, legend=2)

# misclassified samples
misclassified = X[y != perceptron_classifier.predict(X)]
plt.scatter(misclassified[:, 0], misclassified[:, 1], c='red', marker='*',
s=100, label='Misclassified')

# label and title
plt.xlabel("1'st feature")
plt.ylabel("2'nd feature")
plt.title('Decision Regions and misclassified samples for perceptron
method')
plt.legend(loc="upper left")
```

نتایج:


Decision Regions and misclassified samples for perceptron method
0
1
Misclassified
2'nd feature
1'st feature
5
4
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3

بخش ۴

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

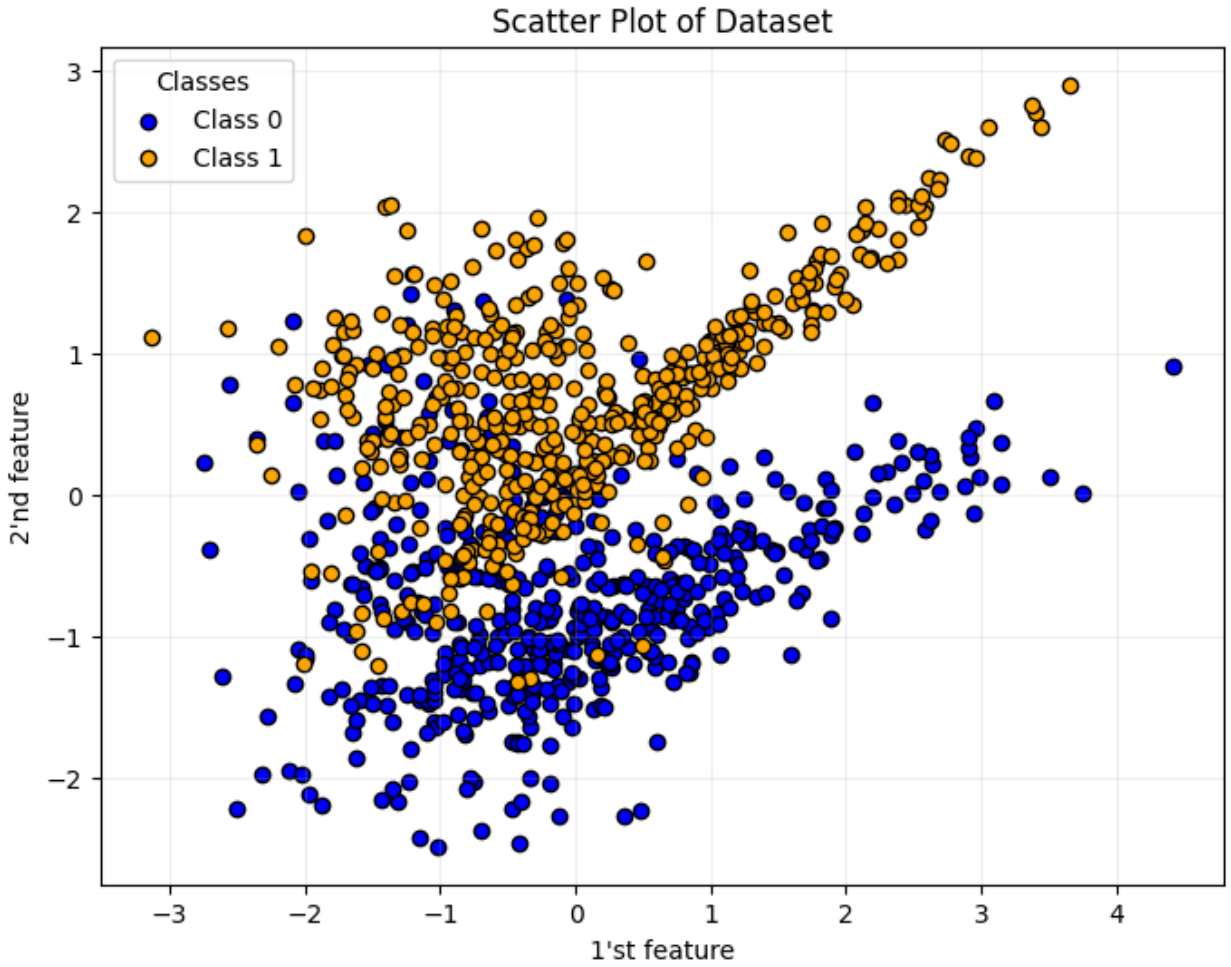
از آنجا که همه توضیحات در بخش های قبل راجع به سوال ۲ و۳ به طور کامل داده شد، پس تنها کد و نتایج در جدید در اینجا آورده خواهد شد.

همانطور که در بخش ۱ گفتیم می توانیم اطلاعات دیگر همانند تعداد شکل های داده های در هر کلاس را به کمک n_clusters_per_class تغییر بدهیم، همچنین برای تعیین میزان تفکیک پذیری کلاس ها از هم از پارامتر class_sep استفاده می کنیم که هر چه کمتر باشد کلاس ها بیشتر در هم فرو رفته اند، در این بخش برای اینکه دیتاست تولید شده را چالش برانگیز تر و سخت تر کنیم می توانیم مقدار n_clusters_per_class را از مقدار default آن یعنی ۱ به ۲ تغییر بدهیم. از طرفی برای اینکه در هم روی داده ها بیشتر شود پارامتر class_sep را از مقدار default آن یعنی ۱ به ۰.۷ تغییر بدهیم.

بنابراین برای ایجاد دیتاست جدید از کد زیر استفاده می کنیم:

```
# Create the dataset
X, y = make_classification(
    n_samples=1000,
    n_features=2,
    n_informative=2,
    n_redundant=0,
    n_repeated=0,
    n_clusters_per_class=2,
    class_sep=0.7,
    n_classes=2,
    random_state=93
    )
# Create a scatter plot
plt.figure(figsize=(8, 6))  # Set the figure size
scatter_class_0 = plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue',
label='Class 0', edgecolors='k', marker='o')
scatter_class_1 = plt.scatter(X[y == 1, 0], X[y == 1, 1], color='orange',
label='Class 1', edgecolors='k', marker='o')
plt.xlabel("1'st feature")
plt.ylabel("2'nd feature")
plt.title('Scatter Plot of Dataset')  # Title for the plot
plt.legend(handles=[scatter_class_0,                    scatter_class_1],
title='Classes',loc="upper left")
plt.grid(alpha=0.2)  # Display grid lines
```

نتایج:



همانطور که انتظار داشتیم داده های هر کلاس انگار شامل ۲ نمونه داده می باشند و همچنین نسبت به حالت قبل میزان در هم روی داده ها تغییر کرده است.

بخش ۲:

سپس دیتاست ایجاد شده را به کمک همان ۳ روشی که در بخش ۲ طبقه بندی کردیم، طبقه بندی می کنیم، باز هم همانند قبل داده های test و train را با همان نسبت ۰.۲ جداسازی کرده.

نتایج:

روش Logistic Regression :

```
y_pred, y_test
```

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

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

روش SGD Classifier:

```
y_pred1, y_test
```

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

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

روش Perceptron:

```
y_pred2, y_test
```

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

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

نتایج Accuracies:

```
Logistic Regression Train Accuracy: 0.86
Logistic Regression Test Accuracy: 0.89
SGD Classifier Train Accuracy: 0.86
SGD Classifier Test Accuracy: 0.89
Perceptron Train Accuracy: 0.85
Perceptron Test Accuracy: 0.84
```

همانطور که مشخص است که از آنجا که مجموعه داده ما پیچیده تر نسبت به قبل شد دقت ما نیز پایین تر آمد.

برای اینکه نتایج بهبود پیدا کند پارامتر ها را به صورت دقیق تعیین می کنیم و با تغییر آنها به بهترین نتیجه ممکن می رسیم(tune کردن فراپارمترها)، از طرفی می توان زمانی که آموزش به همگرایی نرسیده است حداکثر دوره های همگرایی را افزایش داد و همچنین روش های بهینه سازی را تغییر داد.

یکی دیگر از پارامترهای مهم نرخ آموزش است که تعیین آن اهمیت دارد، همچنین برای بهبود نتایج می توان پارامترهای بیشتری را دخیل کرد و یا چون مجموعه داده پیچیده تر شده است از سایر classifier ها و حتی classifier های غیر خطی استفاده کرد.

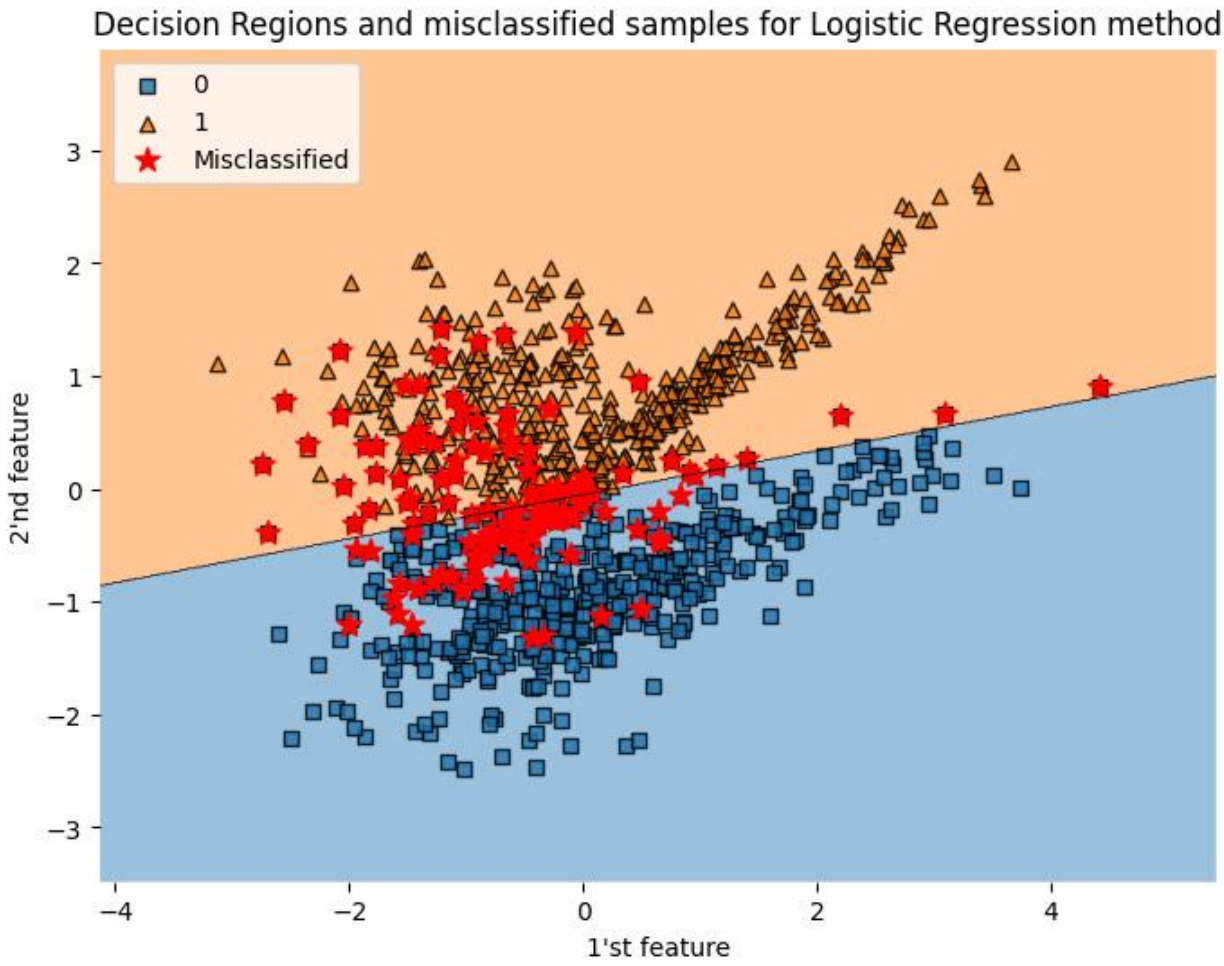
همچنین استفاده از تعداد داده یکسان از هر کلاس در داده های تست.

همچنین نرمال کردن داده ها نیز می تواند سودمند باشد که در بخش های بعد تاثیر آن را می بینیم.

بخش ۳

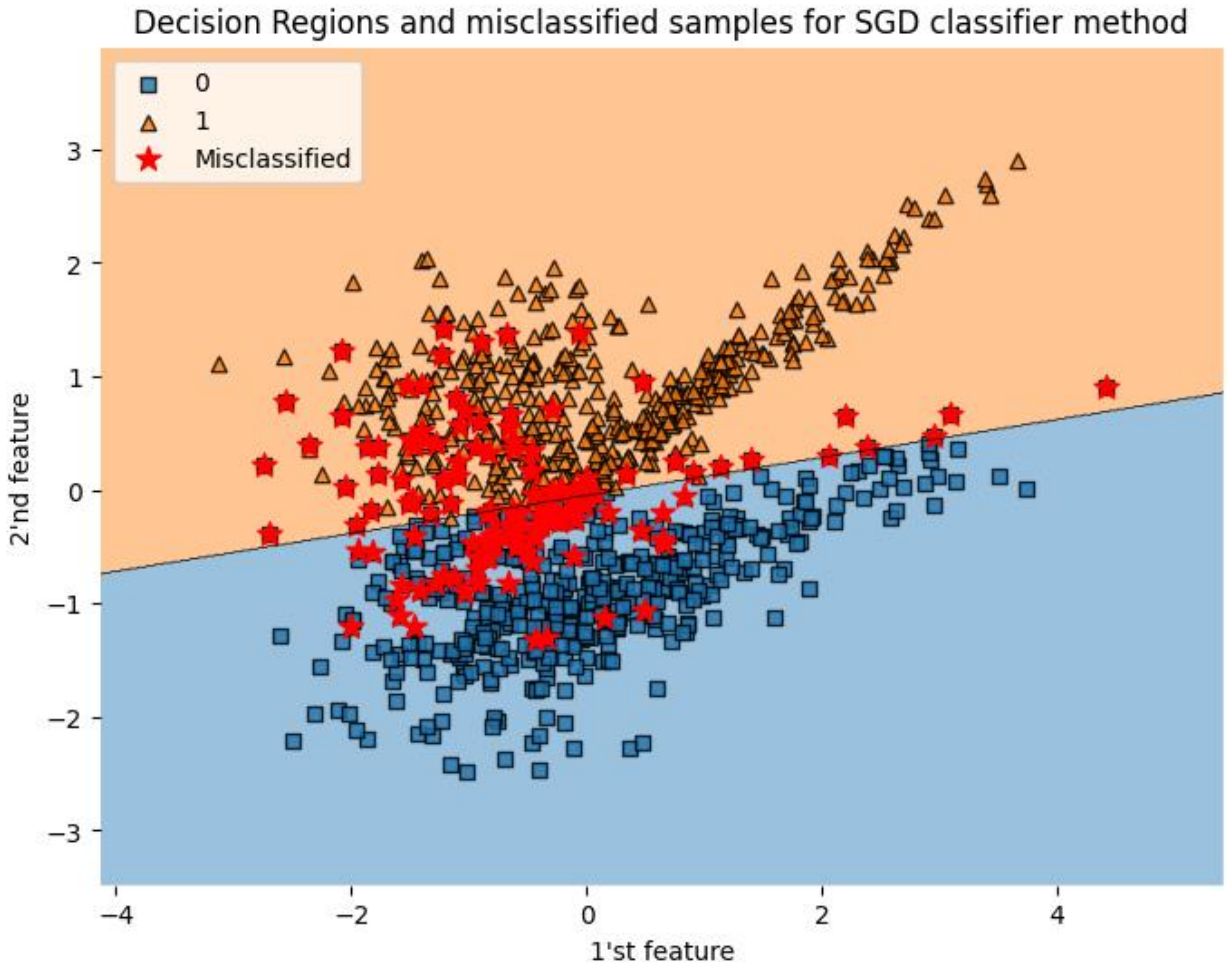
طبقه بندی به روش Logistic Regression :

نتایج:



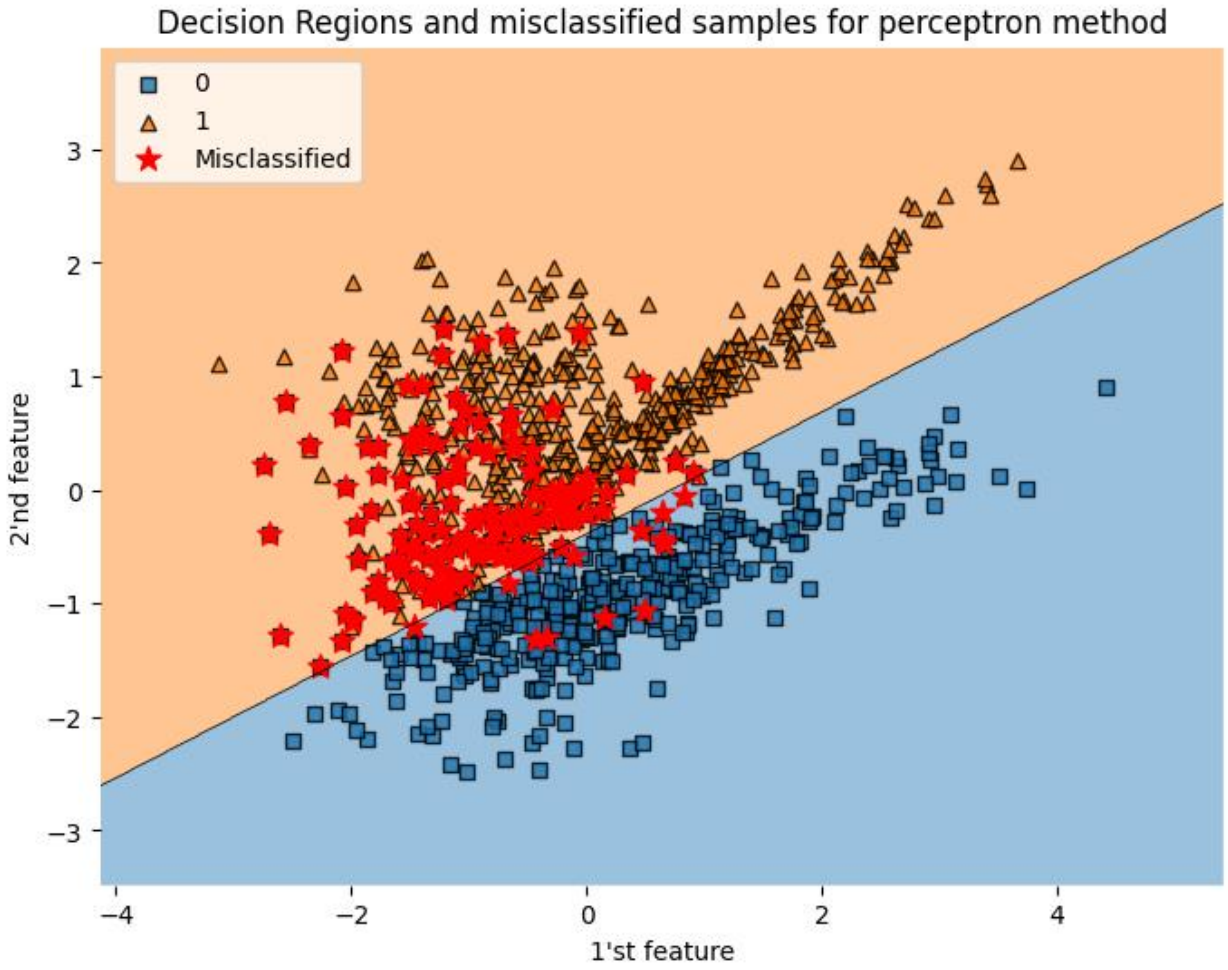
طبقه بندی به روش SGD Classifier:

نتایج:



طبقه بندی یه روش Perceptron:

نتایج:



بخش ۵

اگر یک کلاس به داده های تولید شده در قسمت ۱ اضافه شود، در کدام قسمت ها از بلوک دیاگرام آموزش و ارزیابی تغییراتی ایجاد می شود؟ در مورد این تغییرات توضیح دهید. آیا می توانید در این حالت پیاده سازی را به راحتی و با استفاده از کتابخانه ها و کد های آماده پایتونی انجام دهید؟ پیاده سازی کنید.

مشخص است که با افزودن یک کلاس، کار پیچیده تر خواهد شد. اگر بخواهیم به کمک کد های آماده پایتونی این کار را انجام بدهیم، در فرآیند آموزش برای محاسبه تابع اتلاف باید از تابع اتلاف مناسب استفاده کنیم مثلا کراس آنتروپی باینری برای تنها مجموعه داده دو کلاسه قابل استفاده است و برای مجموعه داده های سه کلاسه قابل استفاده نیست..

همچنین استفاده از تابع سیگموید در توضیحات درس برای مجموعه داده های دو کلاسه به این صورت بود که اگر خروجی سیگموید بیشتر از ۰.۵ باشد به احتمال زیاد مربوط به کلاس ۱ است و اگر کمتر از ۰.۵ باشد به احتمال زیاد مربوط به کلاس صفر است و در فرآیند آموزش پارامتر های w به گونه ای در هر مرحله اصلاح می شوند که تابع اتلاف کمتریم مقدار خود را داشته باشد و بتوانیم خروجی داده های train را به هدف مورد نظر نزدیک و نزدیک تر کنیم.

اما در سه کلاسه باید تمهیدات دیگری را در نظر بگیریم چون در اینجا دیگر با آن موضوع دو کلاسه روبه رو نیستیم، بنابراین می توانیم توابع دیگری به غیر سیگموید را در در نظر بگیریم که بتوانیم سه کلاس را برای آنها در نظر بگیریم و یا از سیگموید به گونه ای استفاده کنیم که بتوانیم ۳ کلاس را برای آن در نظر بگیریم.

بنابراین در این حالت برای محاسبه yhat و همچنین تابع اتلاف باید تمهیداتی را درنظر بگیریم.

در ارزیابی یا همان test نیاز به آپدیت پارامترهای مدل نیست و تنها باید خروجی داده ها مشخص شود، بنابراین تنها نیاز است که تابع مد نظر ما که مثلا سیگموید باشد بتواند داده ها را در ۳ دسته طبقه بندی کند.

در رابطه با ارزیابی نتایج نیز باید تمهیدات جدیدی را بیندیشیم، در رابطه با Binary Classification داشتیم:

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

اما در اینجا چون سه کلاس داریم باید راه دیگری برای محاسبه دقت بیابیم، زیرا با این تکنیک تنها دو کلاس را می توانستیم پوشش دهیم زیرا خروجی سیگموید بین صفر و یک بود.

در هر صورت دیدیم که فرآیند آموزش و ارزیابی هر دو با تغییر مواجه می شد، در کتابخانه های آماده پایتون این موارد همه لحاظ شده است و همچنین می توان در برخی از روش ها در فراپارامترها روش های مختلف را برای داده های چند کلاسه استفاده کرد(مثلا در روش logistic Regression).

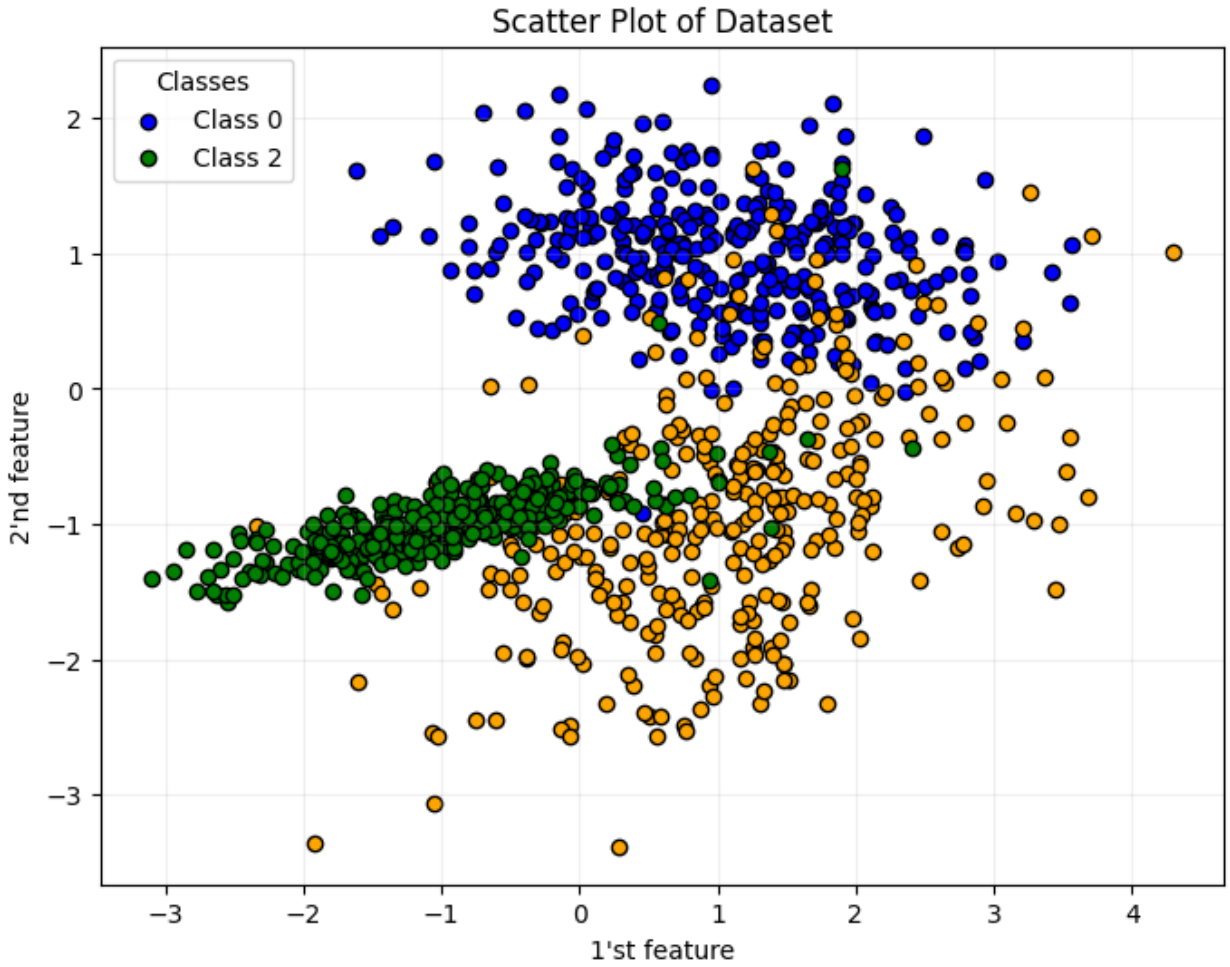
با اعمال در بخش ۱ به بعد تنها باید تعداد کلاس ها را ۳ تعریف کنیم و همچنین نمودارها را برای ۳ کلاس رسم کنیم:

ایجاد dataset:

```
# Create the dataset
X, y = make_classification(
    n_samples=1000,
    n_features=2,
    n_informative=2,
    n_redundant=0,
    n_repeated=0,
    n_clusters_per_class=1,
    class_sep=1,
    n_classes=3,
    random_state=93
    )

# Create a scatter plot
plt.figure(figsize=(8, 6))  # Set the figure size
scatter_class_0 = plt.scatter(X[y == 0, 0], X[y == 0, 1], color='blue',
label='Class 0', edgecolors='k', marker='o')
scatter_class_1 = plt.scatter(X[y == 1, 0], X[y == 1, 1], color='orange',
label='Class 1', edgecolors='k', marker='o')
scatter_class_1 = plt.scatter(X[y == 2, 0], X[y == 2, 1], color='green',
label='Class 2', edgecolors='k', marker='o')
plt.xlabel("1'st feature")
plt.ylabel("2'nd feature")
plt.title('Scatter Plot of Dataset')  # Title for the plot
plt.legend(handles=[scatter_class_0, scatter_class_1],
title='Classes',loc="upper left")
plt.grid(alpha=0.2)  # Display grid lines
```

نتایج:



طبقه بندی:

روش Logistic Regression:

```
y_pred, y_test
```

```
(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]))
```

روش SGD Classifier:

```
y_pred1, y_test
```

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

روش Perceptron:

```
y_pred2, y_test
```

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

نتایج Accuracies :

```
Logistic Regression Train Accuracy: 0.89
Logistic Regression Test Accuracy: 0.90
SGD Classifier Train Accuracy: 0.88
SGD Classifier Test Accuracy: 0.89
Perceptron Train Accuracy: 0.83
Perceptron Test Accuracy: 0.87
```

همانطور که مشخص است که از آنجا که با اضافه شدن یک کلاس مجموعه داده ما پیچیده تر نسبت به قبل شد دقت ما نیز پایین تر آمد.

برای اینکه نتایج بهبود پیدا کند پارامتر ها را به صورت دقیق تعیین می کنیم و با تغییر آنها به بهترین نتیجه ممکن می رسیم(tune کردن فراپارمترها)، از طرفی می توان زمانی که آموزش به همگرایی نرسیده است حداکثر دوره های همگرایی را افزایش داد و همچنین روش های بهینه سازی را تغییر داد.

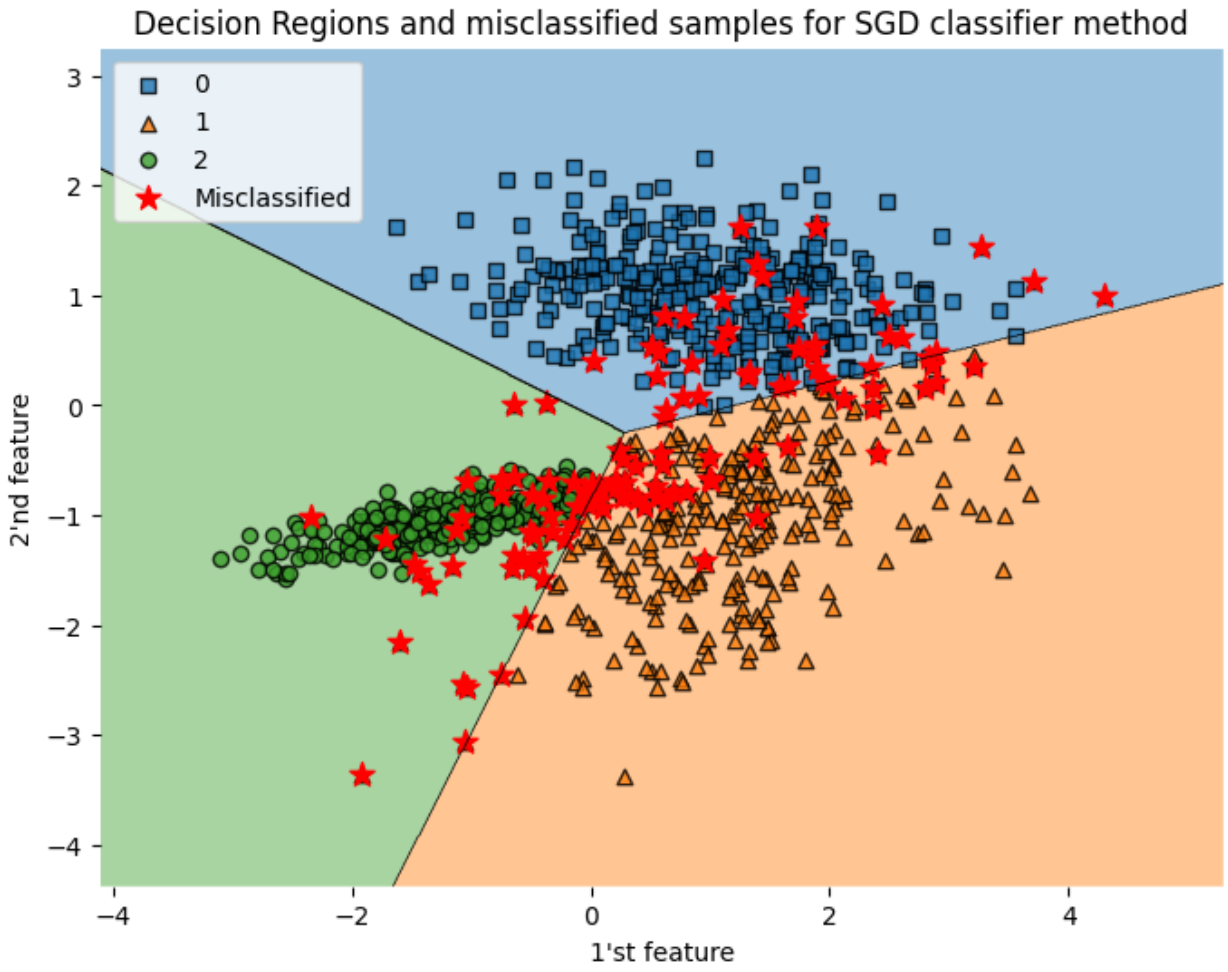
یکی دیگر از پارامترهای مهم نرخ آموزش است که تعیین آن اهمیت دارد، همچنین برای بهبود نتایج می توان پارامترهای بیشتری را دخیل کرد و یا چون مجموعه داده پیچیده تر شده است از سایر classifier ها و حتی classifier های غیر خطی استفاده کرد.

همچنین استفاده از تعداد داده یکسان از هر کلاس در داده های تست.

همچنین نرمال کردن داده ها نیز می تواند سودمند باشد که در بخش های بعد تاثیر آن را می بینیم.

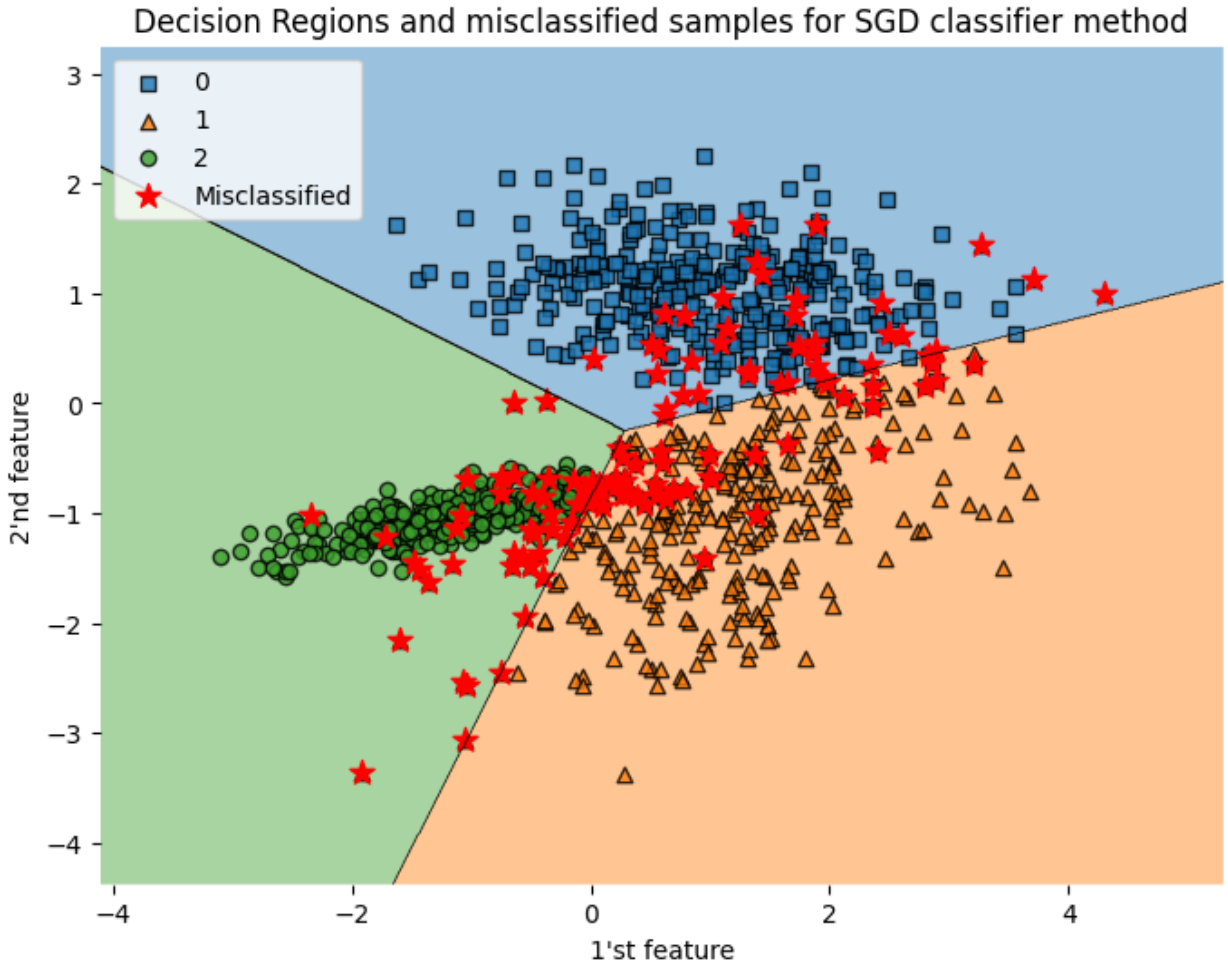
طبقه بندی به روش Logistic Regression :

نتایج:



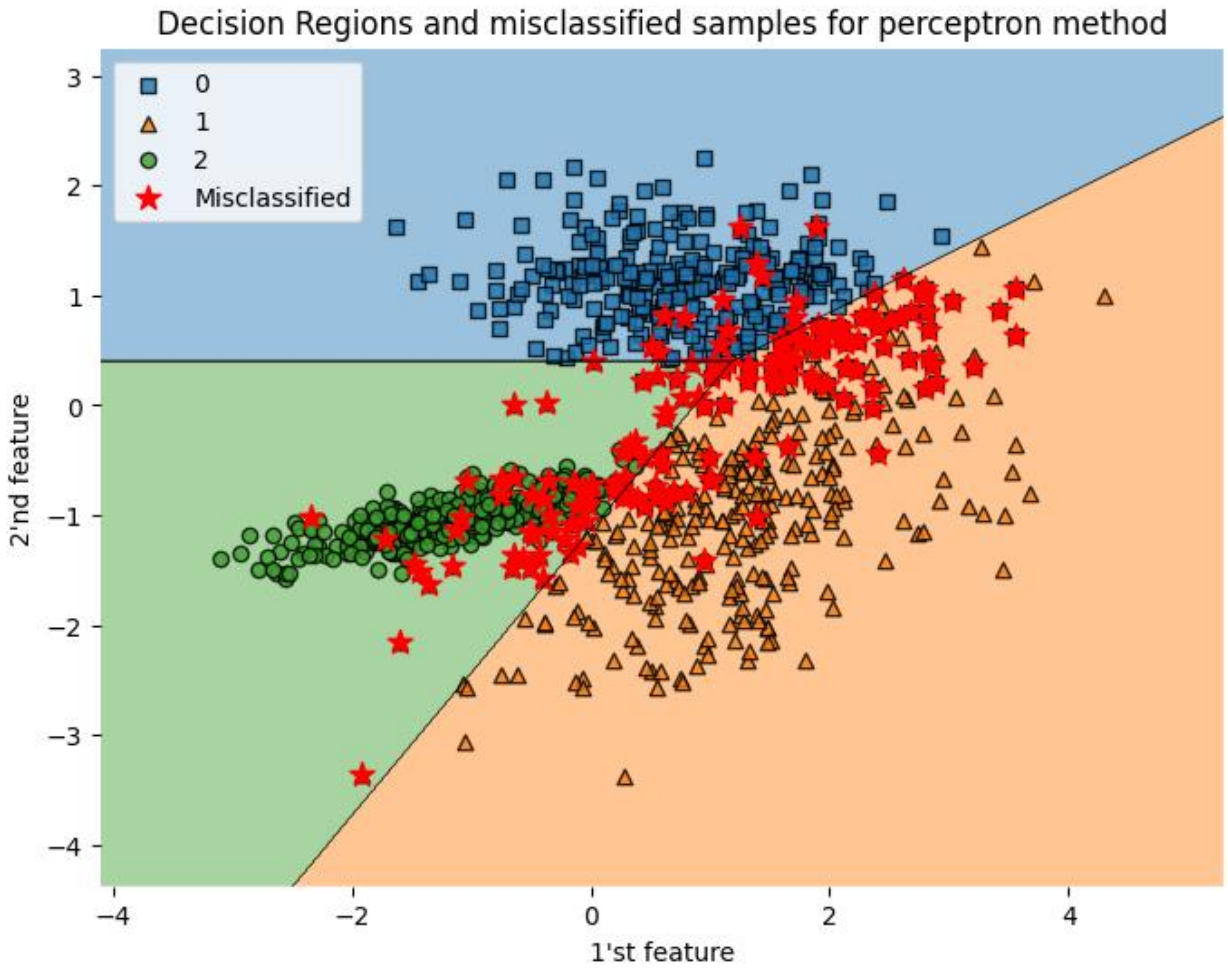
طبقه بندی به روش SGD Classifier:

نتایج:



طبقه بندی یه روش Perceptron:

نتایج:



سوال دوم

بخش ۱

۱- با مراجعه به این پیوند با یک دیتاست مربوط به حوزه بانکی آشنا شوید و ضمن توضیح کوتاه اهداف و ویژگی هایش، فایل آن را دانلود کرده و پس از بارگذاری در گوگل درایو خود، آن را با دستور gdown در محیط گوگل کولب قرار دهید. اگر تغییر فرمتی برای این فایل دیتاست نیاز می بینید، این کار را با دستورهای پایتونی انجام دهید.

دیتاست "Banknote Authentication" یک مجموعه داده است که مربوط به تشخیص اعتبار اسناد بانکی می‌شود. این دیتاست برای مسائل دسته‌بندی بین دو کلاس استفاده می‌شود: اسناد بانکی قابل اعتبار و اسناد بانکی تقلبی.

این دیتاست شامل ۴ ویژگی است که این ویژگی ها معمولاً مربوط به ویژگی‌های استخراج شده از تصاویر اسناد بانکی هستند. این ویژگی‌ها ممکن است شامل ویژگی‌های تصویری مانند ویژگی‌های مرتبط با رنگ، تراکم یا شکل‌های هندسی باشند که توسط الگوریتم‌های پردازش تصویر به دست آمده‌اند. توضیحات این دیتاست به شکل زیر است:

داده‌ها از تصاویری استخراج شده‌اند که از نمونه‌های اسناد بانکی واقعی و تقلبی گرفته شده‌اند. برای دیجیتال‌سازی، از یک دوربین صنعتی که معمولاً برای بازرسی چاپ استفاده می‌شود، استفاده شده است. تصاویر نهایی ابعاد ۴۰۰ × ۴۰۰ پیکسل دارند. به دلیل لنز شیء و فاصله تا شیء مورد بررسی، تصاویر خاکستری با وضوح حدود ۶۶۰ نقطه در اینچ به دست آمده‌اند. از ابزار تبدیل موجک برای استخراج ویژگی‌ها از تصاویر استفاده شده است.

این دیتاست را دانلود کرده و سپس آن را در گوگل درایو بارگزاری می کنیم و به کمک دستور gdown آن را در محیط کولب قرار می دهیم.این کار را به کمک کد زیر انجام می دهیم:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 11wi71E9PppwbkvUnk0LLPbQ9gprklqLe
```

همچنین برای ایجاد dataframe نیاز به کتابخانه panda داریم، بنابراین این کتابخانه و چند تا از کتابخانه های دیگر را فراخوانی می کنیم:

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

همچنین این دیتاست دارای پسوند txt است که مناسب نیست برای تبدیل آن به csv به شکل زیر عمل می کنیم:

```
data = pd.read_csv(r'/content/data_banknote_authentication.txt')
data.to_csv('/content/data_banknote_authentication.csv')
df = pd.read_csv('/content/data_banknote_authentication.csv')
df
```

نتیجه به شکل زیر خواهد بود:

	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
1368	1368	-3.75030	-13.45860	17.5932	-2.77710	1
1369	1369	-3.56370	-8.38270	12.3930	-1.28230	1
1370	1370	-2.54190	-0.65804	2.6842	1.19520	1

1371 rows × 6 columns

همانطور که مشخص است خود داده ها شماره گذاری شده هستند و نیاز به شماره گذاری ندارند، از طرفی داده اول شماره گذاری نشده است، این دیتاست دارای ۴ ویژگی و یک هدف است که این هدف تقلبی بودن یا نبودن اسناد را مشخص می کند. همچنین ویژگی ها به نام مشخص در یک ستون قرار نگرفته اند، پس برای رفع این مشکلات به شکل زیر عمل می کنیم:

```
data = pd.read_csv(r'/content/data_banknote_authentication.txt')
headerlist = ["1'st feature", "2'nd feature", "3'rd feature", "4'th
feature", "target"]
data.to_csv('/content/data_banknote_authentication.csv',
header=headerlist, index=False)
df = pd.read_csv('/content/data_banknote_authentication.csv')
df
```

نتایج:

	1'st feature	2'nd feature	3'rd feature	4'th feature	target
0	4.54590	8.16740	-2.4586	-1.46210	0
1	3.86600	-2.63830	1.9242	0.10645	0
2	3.45660	9.52280	-4.0112	-3.59440	0
3	0.32924	-4.45520	4.5718	-0.98880	0
4	4.36840	9.67180	-3.9606	-3.16250	0
...	...	...	...	...	...
1366	0.40614	1.34920	-1.4501	-0.55949	1
1367	-1.38870	-4.87730	6.4774	0.34179	1
1368	-3.75030	-13.45860	17.5932	-2.77710	1
1369	-3.56370	-8.38270	12.3930	-1.28230	1
1370	-2.54190	-0.65804	2.6842	1.19520	1

1371 rows × 5 columns

بخش ۲

۲- ضمن توضیح فرآیند بر زدن (محلوط کردن)، داده ها را مخلوط کرده و با نسبت تقسیم دلخواه و معقول به دو بخش آموزش و ارزیابی تقسیم کنید.

فرایند بر زدن یا شافل کردن داده‌ها به معنای تغییر ترتیب داده‌ها در دیتاست است. این کار معمولاً با استفاده از عملیات تصادفی انجام می‌شود. شافل کردن داده‌ها در متغیرهای آماری و ماشین لرنینگ اهمیت زیادی دارد. در زیر، به برخی از اهمیت‌های شافل کردن داده‌ها اشاره می‌شود:

- **حذف الگوهای مرتبط با ترتیب اولیه:** در صورتی که داده‌ها به ترتیب مشخص شده و بدون شافل کردن مستقیماً به مدل داده شوند، ممکن است الگوهایی که مربوط به ترتیب اولیه هستند، در مدل تاثیر بگذارند و مدل به شکل ناخواسته با الگوهای خاص به ترتیب داده‌ها آموزش ببیند. با شافل کردن، این الگوها حذف می‌شوند.
- **جلوگیری از برازش بیش از حد(Overfitting) :** اگر مدل داده‌ها را به ترتیب یاد بگیرد و از آنها الگوهای خاصی استخراج کند، ممکن است در ارتباط با داده‌های جدید که به شکل متفاوتی مرتب شده‌اند، عملکرد ضعیفی داشته باشد. شافل کردن به مدل کمک می‌کند که الگوهای جدیدی بیاموزد و به طور کلی‌تر به داده‌های جدید عملکرد بهتری داشته باشد.
- **تعمیم‌پذیری بهتر:** با شافل کردن داده‌ها، مدل بهتر می‌تواند الگوهای عمومی را یاد بگیرد و به داده‌های جدیدی که از پیش ندیده است، تعمیم بدهد. این امر موجب می‌شود مدل به طور کلی‌تر و کارآمدتری برای پیش‌بینی و تصمیم‌گیری ارائه دهد.
- **تسهیل ارزیابی درست:** در برخی موارد، ارزیابی دقت یا عملکرد مدل نیاز به اجتناب از هر گونه ترتیب در داده‌ها دارد تا اطمینان حاصل شود که مدل به صورت عمومی و بدون افت توانایی پیش‌بینی دارد. شافل کردن داده‌ها موجب ایجاد یک زمینه آزمون مناسب می‌شود.

برای مخلوط کردن داده ها از کتابخانه زیر استفاده می کنیم:

```
from sklearn.utils import shuffle
```

حال می توانیم به کمک کد زیر داده ها را مخلوط می کنیم:

```
df = shuffle(df, random_state=93)
df
```

همانطور که مشخص است برای حفظ تکرار پذیری از random_state استفاده می کنیم.

نتایج:

	1'st feature	2'nd feature	3'rd feature	4'th feature	target
729	2.8672	10.00080	-3.20490	-3.109500	0
103	4.2027	0.22761	0.96108	0.972820	0
82	1.8664	7.77630	-0.23849	-2.963400	0
772	-1.5252	-6.25340	5.35240	0.599120	1
897	-5.2406	6.62580	-0.19908	-6.860700	1
...	...	...	...	...	...
132	0.4339	5.53950	2.03300	-0.404320	0
858	-5.8730	9.17520	-0.27448	-6.042200	1
1240	-2.2811	-0.85669	2.71850	0.044382	1
83	3.2450	6.63000	-0.63435	0.869370	0
1032	1.5077	1.95960	-3.05840	-0.122430	1

1371 rows × 5 columns

همچنین برای تقسیم بندی داده ها به داده های آموزش و ارزیابی نیاز است که ویژگی ها را از target جدا کنیم، برای اینکار به صورت زیر عمل می کنیم:

```
X = df[["1'st feature", "2'nd feature", "3'rd feature", "4'th
feature"]].values

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

نتایج به صورت زیر خواهد بود:

```
(array([[ 2.8672 , 10.0008 , -3.2049 , -3.1095 ], [ 4.2027 , 0.22761 ,
0.96108 , 0.97282 ], [ 1.8664 , 7.7763 , -0.23849 , -2.9634 ], ..., [-
2.2811 , -0.85669 , 2.7185 , 0.044382], [ 3.245 , 6.63 , -0.63435 ,
0.86937 ], [ 1.5077 , 1.9596 , -3.0584 , -0.12243 ]]),

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

حال به کمک همان روش سوال ۱ داده ها را به نسبت ۸۰ به ۲۰ به داده های آموزش و ارزیابی تقسیم می کنیم:

```
from sklearn.model_selection import train_test_split
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
```

انتظار داریم X یک آرایه ۱۰۹۶ در ۴ باشد زیرا ۱۳۷۰ نمونه داشتیم و هر نمونه دارای ۴ ویژگی است و y باید یک آرایه ۲۷۵ در ۱ باشد.

نتایج:

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

نتایج همانی شد که انتظار داشتیم.

بخش ۳

۳- بدون استفاده از کتابخانه های آماده پایتون، مدل، تابع اتلاف و الگوریتم یادگیری و ارزیابی را کد نویسی کنید تا دو کلاس موجود در دیتاست به خوبی از یکدیگر تفکیک شوند. نمودار تابع اتلاف را رسم کنید و نتیجه دقت ارزیابی روی داده های تست را محاسبه کنید. نمودار تابع اتلاف را تحلیل کنید. آیا می توان از روی نمودار تابع اتلاف و قبل از مرحله ارزیابی با قطعیت در مورد عملکرد مدل نظر داد؟ چرا و اگر نمی توان راه حل چیست؟

برای اینکار همانند روشی که در درس توضیح داده شد از روش Logistic Regression استفاده می کنیم، برای این روش نیاز است تا ابتدا تابع سیگموید را تعریف کنیم و سپس و همچنین تابع خروجی yhat این روش که سیگموید $W^T X$ است را نوشته و سپس تابع مربوط به تابع اتلاف کراس آنتروپی را بنویسیم. حال نیاز است تا تابع گرادیان را نوشته و به کمک آن تابع و نرخ یادگیری تابع مربوط به روش گرادیان نزولی را بنویسیم.

همچنین از آنجا که نیاز به محاسبه دقت داریم باید تابع مربوط به دقت نیز نوشته شود.

```
def sigmoid(x):
  return 1/(1+np.exp(-x))
def logistic_regression(x, w):
  y_hat = sigmoid(x @ w)
  return y_hat
def bce(y, y_hat):
  loss = -(np.mean(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)))
  return loss
def gradient(x, y, y_hat):
  grads = (x.T @ (y_hat-y))/len(y)
  return grads
def gradient_descent(w, eta, grads):
  w -= eta*grads
  return w
def accuracy(y, y_hat):
  acc = np.sum(y == np.round(y_hat))/len(y)
  return acc
```

حال برای مرحله آموزش آماده هستیم، در این مرحله نیاز است تا یک درایه یک به انتهای ویژگی های هر داده اضافه شود:

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

انتظار داریم که x_train یک آرایه ۱۰۹۶ در ۵ باشد، نتایج نیز همین را نشان می دهد:

```
(1096, 5)
```

و همچنین با توجه به تعداد ویژگی ها که در اینجا ۴ است ضرایب w به صورت تصادفی و به صورت یک آرایه ۵ در یک تولید شوند همچنین نیاز است تا نرخ یادگیری و تعداد ایپاک ها را مشخص کنیم:

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

eta = 0.01 # learning rate
n_epochs = 4500 # number of epoc
```

نتایج:

```
(5, 1)
```

حال فرآیند یادگیری طبق الگوریتم زیر انجام می دهیم:

```
error_hist = []

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

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

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

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

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

برای اینکار می توانیم هر ۱۰۰ ایپاک خطا و w ها را نشان دهیم:

```
Epoch=99,    E=0.1362,   w=[ 0.92259859 -0.95595875 -0.32127601 -0.34916918
0.06105625]
Epoch=199,   E=0.1138,   w=[ 0.94193463 -1.00946655 -0.40141315 -0.45750698
0.04515726]
Epoch=299,   E=0.1027,   w=[ 0.96860915 -1.06048863 -0.45548359 -0.52215279
0.02190546]
Epoch=399,   E=0.09527,  w=[ 9.97060039e-01 -1.10645533e+00 -4.97520313e-01
-5.69458852e-01
 -4.88665910e-04]
Epoch=499,   E=0.08966,  w=[ 1.02559513 -1.14751017 -0.53243635 -0.6080176
-0.02024103]
```

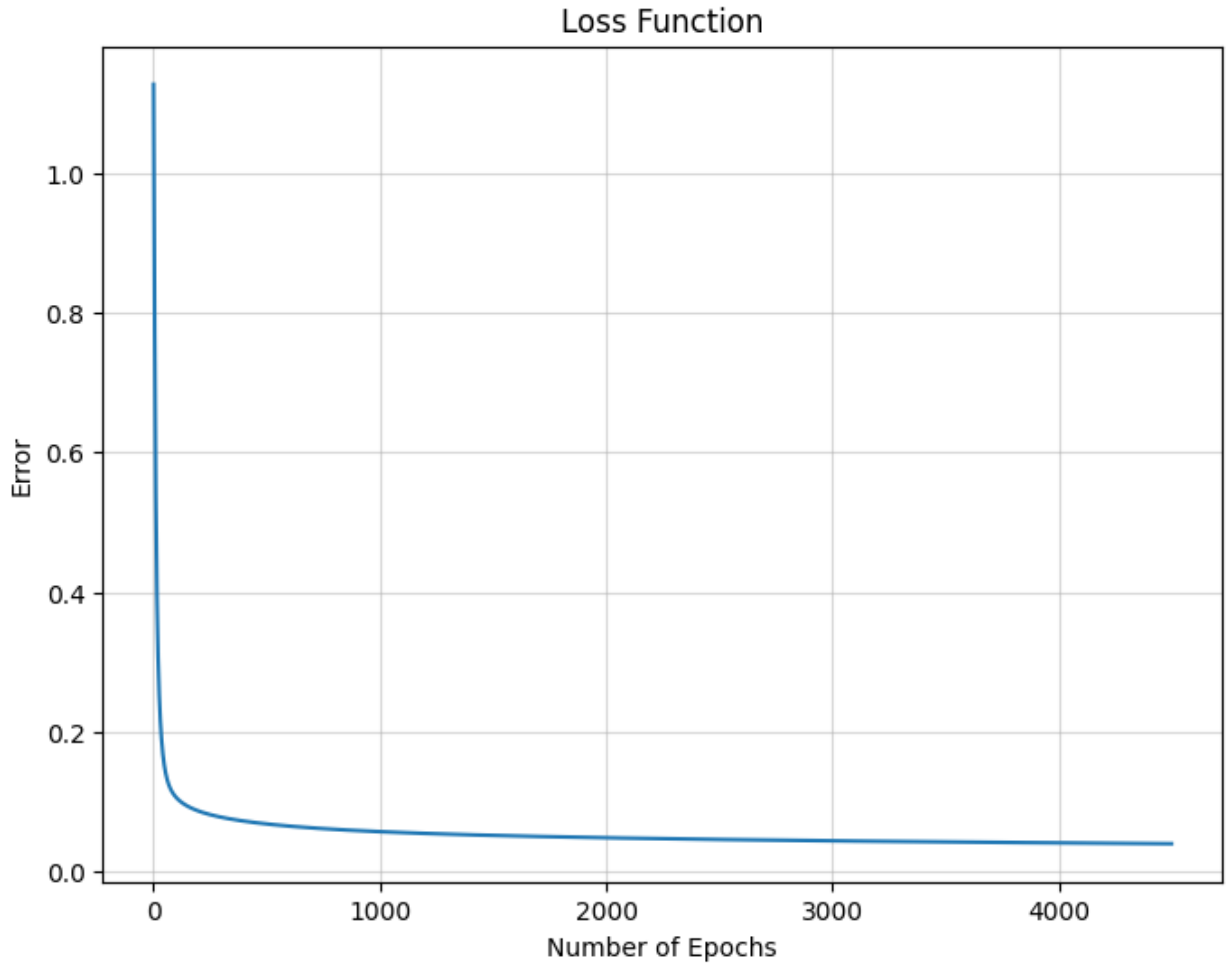
```
Epoch=599,   E=0.0852,   w=[ 1.05363658 -1.18438602 -0.56264068 -0.6412451
-0.03720283]
Epoch=699,   E=0.08152,  w=[ 1.08098648 -1.21781304 -0.58944949 -0.67080074
-0.05166586]
Epoch=799,   E=0.07842,  w=[ 1.10759029 -1.24839769 -0.61365756 -0.6976162
-0.06400139]
Epoch=899,   E=0.07574,  w=[ 1.1334505  -1.27662039 -0.63578661 -0.722273
-0.07455428]
Epoch=999,   E=0.07339,  w=[ 1.15859235 -1.30285819 -0.65620212 -0.745164
-0.08361777]
Epoch=1099,        E=0.07131,  w=[ 1.18304932 -1.3274082  -0.67517314 -
0.76657096 -0.09143366]
Epoch=1199,        E=0.06944,  w=[ 1.20685688 -1.3505068  -0.69290538 -
0.78670533 -0.09819933]
Epoch=1299,        E=0.06775,  w=[ 1.23004974 -1.37234421 -0.70956078 -
0.80573149 -0.10407582]
Epoch=1399,        E=0.06621,  w=[ 1.25266069 -1.39307524 -0.72526981 -
0.82378074 -0.10919491]
Epoch=1499,        E=0.06479,  w=[ 1.27472027 -1.41282737 -0.74013953 -
0.84096031 -0.11366508]
Epoch=1599,        E=0.06349,  w=[ 1.29625667 -1.43170671 -0.75425916 -
0.85735926 -0.11757615]
Epoch=1699,        E=0.06228,  w=[ 1.31729589 -1.44980245 -0.76770398 -
0.87305269 -0.12100295]
Epoch=1799,        E=0.06115,  w=[ 1.33786188 -1.46719026 -0.78053819 -
0.88810467 -0.12400826]
Epoch=1899,        E=0.0601,   w=[ 1.35797672 -1.4839349  -0.79281707 -
0.90257043 -0.12664504]
Epoch=1999,        E=0.05911,  w=[ 1.37766082 -1.50009224 -0.8045886  -
0.91649798 -0.12895827]
Epoch=2099,        E=0.05819,  w=[ 1.39693306 -1.51571082 -0.8158947  -
0.92992941 -0.13098632]
Epoch=2199,        E=0.05731,  w=[ 1.415811   -1.53083308 -0.82677225 -
0.94290186 -0.1327621 ]
Epoch=2299,        E=0.05648,  w=[ 1.43431095 -1.54549638 -0.8372539  -
0.95544828 -0.13431402]
Epoch=2399,        E=0.0557,   w=[ 1.45244811 -1.55973379 -0.84736866 -
0.96759806 -0.13566667]
Epoch=2499,        E=0.05496,  w=[ 1.4702367  -1.57357471 -0.85714245 -
0.97937758 -0.13684143]
Epoch=2599,        E=0.05425,  w=[ 1.48769003 -1.58704545 -0.86659854 -
0.99081059 -0.137857  ]
Epoch=2699,        E=0.05357,  w=[ 1.5048206  -1.60016961 -0.87575789 -
1.00191854 -0.13872977]
Epoch=2799,        E=0.05293,  w=[ 1.52164014 -1.61296849 -0.88463945 -
1.01272093 -0.13947414]
Epoch=2899,        E=0.05232,  w=[ 1.53815971 -1.62546138 -0.8932604  -
1.02323551 -0.14010286]
Epoch=2999,        E=0.05173,  w=[ 1.55438974 -1.63766577 -0.90163639 -
1.03347848 -0.1406272 ]
Epoch=3099,        E=0.05117,  w=[ 1.57034008 -1.64959764 -0.9097817  -
1.04346469 -0.14105718]
Epoch=3199,        E=0.05062,  w=[ 1.58602007 -1.66127157 -0.91770941 -
1.05320778 -0.14140172]
```

```
Epoch=3299,        E=0.05011,  w=[ 1.60143854 -1.67270097 -0.9254315  -
1.06272033 -0.1416688 ]
Epoch=3399,        E=0.04961,  w=[ 1.61660386 -1.68389814 -0.93295902 -
1.07201392 -0.14186555]
Epoch=3499,        E=0.04913,  w=[ 1.63152401 -1.69487442 -0.94030213 -
1.08109929 -0.14199839]
Epoch=3599,        E=0.04867,  w=[ 1.64620655 -1.70564033 -0.94747023 -
1.08998639 -0.14207307]
Epoch=3699,        E=0.04822,  w=[ 1.6606587  -1.71620556 -0.95447203 -
1.09868446 -0.14209481]
Epoch=3799,        E=0.04779,  w=[ 1.67488732 -1.72657916 -0.96131559 -
1.10720212 -0.14206829]
Epoch=3899,        E=0.04738,  w=[ 1.68889899 -1.7367695  -0.96800842 -
1.11554737 -0.14199777]
Epoch=3999,        E=0.04698,  w=[ 1.70269995 -1.74678442 -0.97455751 -
1.12372771 -0.1418871 ]
Epoch=4099,        E=0.04659,  w=[ 1.71629619 -1.75663123 -0.98096936 -
1.13175016 -0.14173978]
Epoch=4199,        E=0.04621,  w=[ 1.72969344 -1.76631675 -0.98725006 -
1.13962127 -0.14155899]
Epoch=4299,        E=0.04585,  w=[ 1.74289718 -1.77584738 -0.9934053  -
1.14734721 -0.14134765]
Epoch=4399,        E=0.0455,   w=[ 1.75591266 -1.78522914 -0.9994404  -
1.15493376 -0.1411084 ]
 w=[ 1.76874492 -1.79446768 -1.00536038 -  E=0.04516,      Epoch=4499,
1.16238638 -0.14084366]
```

نمودار تابع اتلاف:

```
plt.figure(figsize=(8, 6))
plt.plot(error_hist)
plt.xlabel("Number of Epochs")
plt.ylabel("Error")
plt.title('Loss Function')
plt.grid(alpha=0.5)
```

نتایج:



محاسبه دقت برای داده های تست:

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

نتایج:

```
0.9818181818181818
```

همانطور که مشخص است به دقتی حدود ۹۸ درصد رسیدیم.

نمودار تابع اتلاف در طول آموزش می تواند بینش ارزشمندی در مورد همگرایی یادگیری مدل ارائه دهد، اما همیشه برای نتیجه گیری قطعی در مورد عملکرد آن کافی نیست. از جمله دلایل آن می توان به موارد زیر اشاره کرد:

- تطبیق بیش از حد و تعمیم: یک مدل ممکن است در مجموعه داده های آموزش عملکرد خوبی داشته باشد، اما تا حدودی ارزیابی واقعی آن با ارزیابی بر روی مجموعه داده های تست است. تطبیق بیش از حد زمانی اتفاق می‌افتد که یک مدل داده‌های آموزشی را به خوبی یاد می‌گیرد، از جمله نویز و نقاط پرت آن، اما نمی‌تواند به داده‌های جدید و دیده نشده تعمیم یابد. بررسی دقت در یک مجموعه تست جداگانه برای ارزیابی عملکرد بسیار مهم است.

- معیارهای ارزیابی: علاوه بر تابع اتلاف، سایر معیارهای ارزیابی مانند دقت نیز برای درک جامع عملکرد مدل ضروری هستند. تابع اتلاف ممکن است تمام جنبه‌های رفتار مدل را در بر نگیرد، به‌ویژه زمانی که با مجموعه داده‌های 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 تابع اتلاف 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 تابع اتلاف ممکن است در چند دوره تثبیت نشود. نظارت بر منحنی در تعداد مناسبی از دوره‌ها ضروری است. علاوه بر این، استفاده از تکنیک‌هایی مانند توقف زودهنگام می‌تواند از تطبیق بیش از حد جلوگیری کند.

راه حل برای بررسی عملکرد:

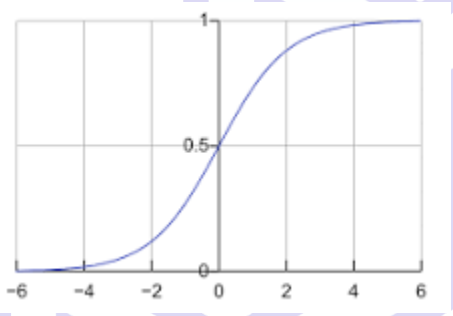
- ارزیابی بر روی یک مجموعه داده های تست
- استفاده از معیارهای مختلف ارزیابی

بخش ۴

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

نرمال‌سازی داده‌ها یک مرحله مهم در پردازش و تحلیل داده‌هاست که هدف آن ایجاد یک دسته‌بندی یکنواخت از داده‌ها برای اجتناب از مشکلات ناشی از مقیاس‌ها و واحدهای مختلف در داده‌ها است.

یکی از موارد مهمی که می توان به آن اشاره کرد این است که مثلا در روش Logistic Regression ما از تابع سیگموید استفاده کردیم که نمودار آن به شکل زیر است:



همانطور که مشخص است این نمودار یک ناحیه اشباع دارد (یک ناخیه اشباع در صفر و یک ناحیه اشباع در ۱)، موضوعی که در اینجا اهمیت دارد این است که ما علاقه مند به قرار گیری داده ها در ناحیه اشباع نیستیم و اگر داده ها را نرمالیزه نکنیم امکان این که داده ها در محدوده اشباع قرار بگیرند وجود دارد و ممکن است بسیار طول بکشد تا خطا از حدی کمتر شود. بنابراین نرمالیزه کردن داده ها می تواند بسیار سودمند باشد، البته این بدان معنا نیست که نرمالیزه کردن داده ها همیشه سودمند است و باعث افزایش دقت می شود. حتی دقت روش های نرمالیزه کردن نیز با هم متفاوت است، بنابراین انتخاب روش مناسب نرمالیزه کردن می تواند سبب افزایش دقت شود.

دو روش متداول برای نرمال‌سازی داده‌ها 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:

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

:Z-score Standardization

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

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

برای نرمال سازی داده ها باید داده ها را به طور کلی نرمال کنیم، چه داده های آموزش و چه داده های ارزیابی، یعنی قبل از اینکه داده ها را به دسته ارزیابی و تست تقسیم کنیم باید فرآیند نرمال سازی انجام شود، زیرا سیستم بر اساس رنج نرمال شده داده ها آموزش می بیند و نمی توان سیستم را با داده های نرمال شده آموزش داد و با داده هایی که نرمالیزه نشده اند تست کرد. زیرا رنج آنها کاملا متفاوت است و قطعا سیستم دچار اشتباه خواهد شد.

برای نرمال سازی داده ها تنها باید داده های مربوط به feature ها نرمالیزه شوند، زیرا خروجی به صورت باینری است.

از روش Z-score Standardization برای نرمال سازی داده ها استفاده می کنیم:

بنابراین ابتدا میانگین و انحراف معیار داده ها را محاسبه می کنیم و سپس از رابطه زیر برای نرمال سازی داده ها استفاده می کنیم:

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

```
mean = np.mean(X, axis=0)
std = np.std(X, axis=0)

normalized_X = (X - mean) / std
normalized_X
```

نتایج:

```
array([[-0.42208972, 0.26292336, -1.12362946, -1.06401182], [ 1.75522238,
1.40265256, -1.27298425, -1.19324112], [-1.25569709, -0.25490261,
0.28357818, 0.53689021],

 ...,

[ 0.58930668, 0.96947739, -0.43593593, -0.31124522], [-0.46695402,
0.01883083, -0.87405493, -0.03922113], [ 1.55039792, 1.3800656 , -
1.32866446, -1.49239623]])
```

بخش ۵

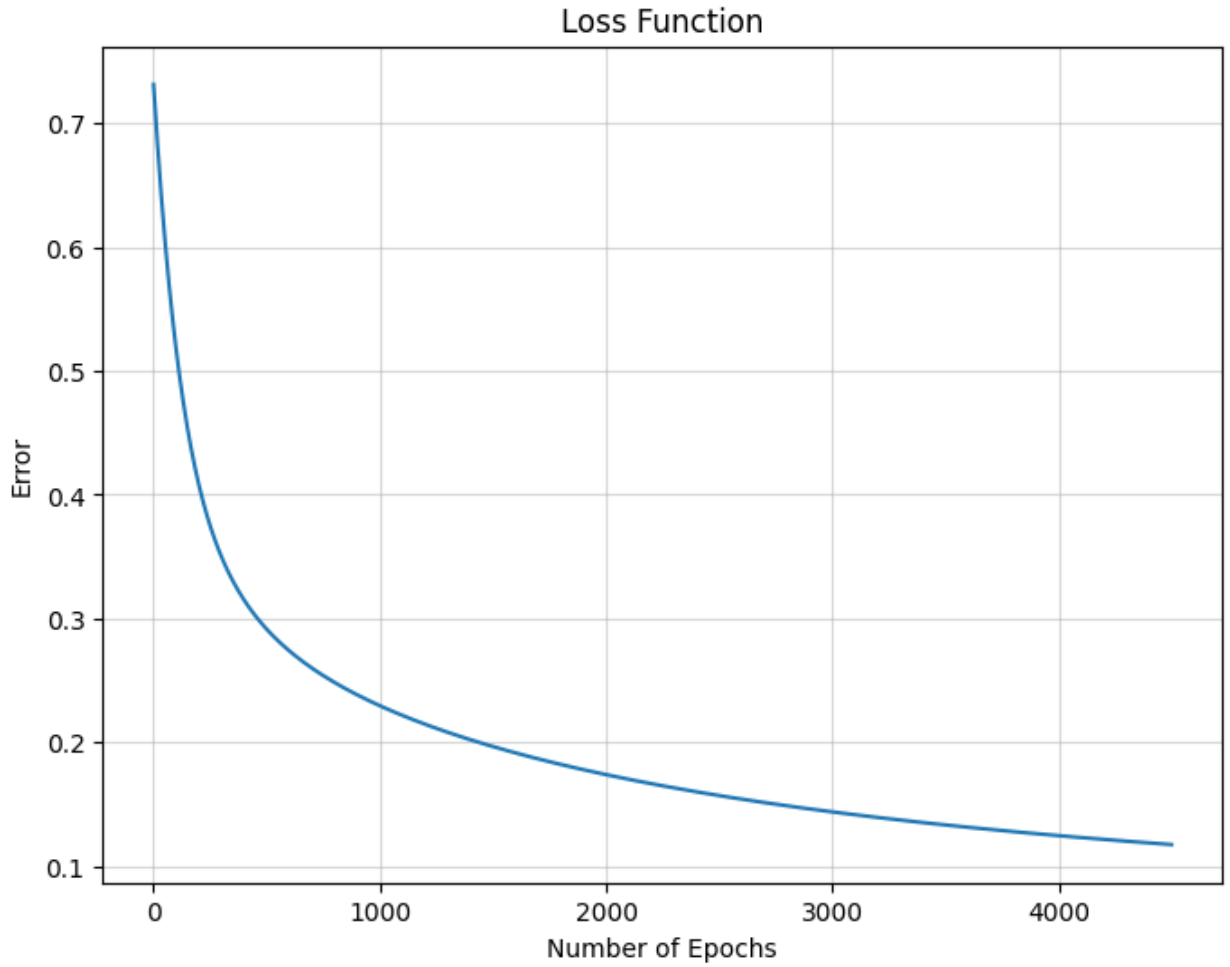
۵- تمام قسمت های ۱ تا ۳ را با استفاده از داده های نرمال شده تکرار کنید و نتایج پیش بینی مدل را برای پنج نمونه داده نشان دهید.

تنها تفاوتی که در این قسمت با قشمت های قبل دیده می شود این است که به جای اینکه داده های test و train از بین X و y انتخاب شوند، از بین Normalized_X و y انتخاب می شوند.

```
x_train , x_test , y_train , y_test = train_test_split(normalized_X , y , test_size = 0.2, random_state=93)
```

نتایج خروجی:

نمودار تابع اتلاف:



محاسبه دقت برای داده های تست:

```
0.9709090909090909
```

همانطور که مشخص است دقت نسبت به قبل تغییر خاصی نکرد و حتی کمتر هم شد، زیرا بدون نرمالیزه کردن داده ها دقت خودش بسیار خوب بود و چیزی حدود ۹۸ درصد بود، اما در داده هایی که در حالت عادی دقت خوبی نداشته باشند، نرمالیزه کردن و انتخاب روش مناسب برای نرمالیزه کردن، می تواند دقت را افزایش دهد.

انتخاب ۵ داده تصادفی از بین داده های تست و نمایش پیش بینی سیستم و همچنین target اصلی:

برای اینکار از دستور random.choice در کتابخانه numpy استفاده می کنیم.

بنابراین کد ما به شکل زیر خواهد بود:

```
# Number of desired samples
num_samples = 5
# Select 5 random indices from the test data
random_indices = np.random.choice(len(x_test), num_samples, replace=False)

# Data samples corresponding to the selected indices
random_samples = x_test[random_indices]
y_random_samples = y_test[random_indices]

y_hat_random_samples = logistic_regression(random_samples, w)

# Display the model predictions and real targets for the selected samples
print("Model Predictions for the Selected Samples:")
print(y_hat_random_samples)

print("Real Targets for the Selected Samples:")
print(y_random_samples)
```

نتایج:

```
Model Predictions for the Selected Samples:
[[0.95131988]
 [0.04527775]
 [0.9125601 ]
 [0.92424417]
 [0.05484296]]
Real Targets for the Selected Samples:
[[1]
 [0]
 [1]
 [1]
 [0]]
```

همانطور که مشخص است برای این ۵ داده پیش بینی به خوبی انجام شده است و بسیار به target اصلی نزدیک است.

بخش ۶

۶- با استفاده از کد نویسی 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 تعدل 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 وگزارش کنید.

به کمک کد زیر می توان وضعیت تعادل داده ها را نشان داد:

ابتدا داده های y را در یک DataFrame جداگانه قرار می دهیم و اسم این DataFrame را new_y قرار می دهیم.

حال به کمک value_counts تعداد داده های موجود در هر کلاس را می شماریم:

```
new_y = pd.DataFrame(y)
new_y.value_counts()
```

نتایج:

```
0    761
1    610
```

$$\frac{761}{761+610} \times 100\% \cong 55.51\%$$

بنابراین انتظار داریم ۵۵.۵۱ درصد داده ها متعلق به کلاس صفر هستند.

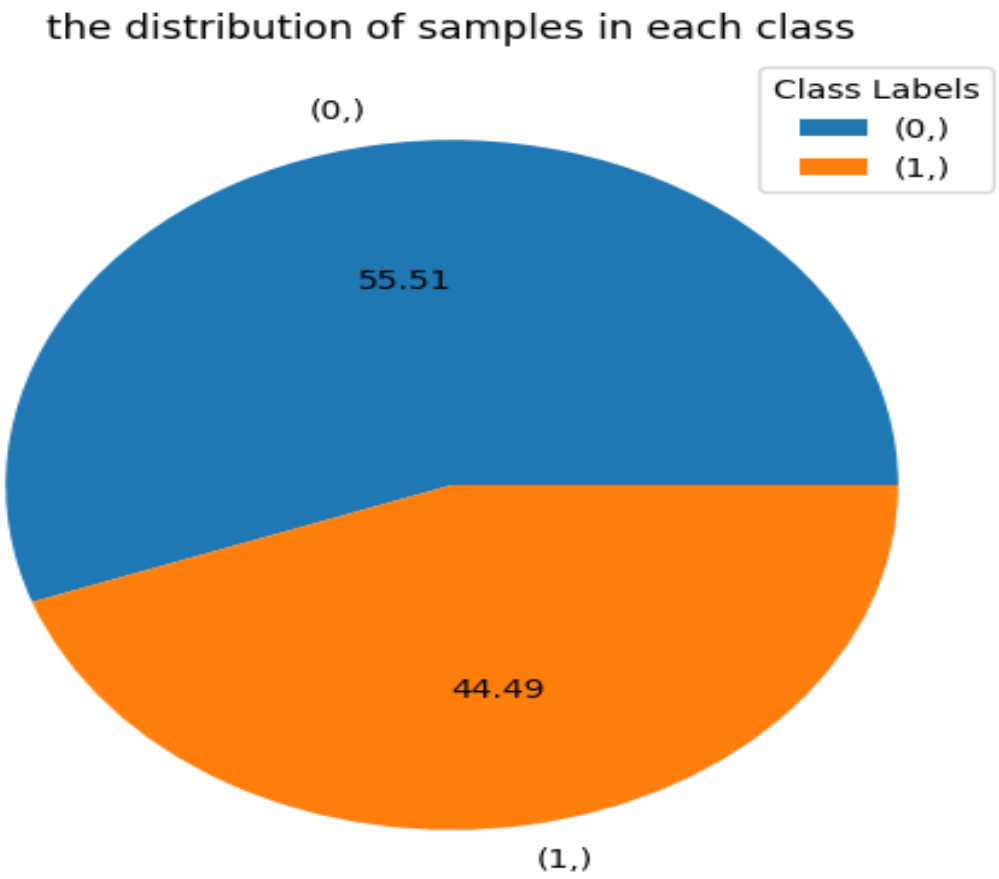
$$\frac{610}{761+610} \times 100\% \cong 44.49\%$$

بنابراین انتظار داریم 44.49 درصد داده ها متعلق به کلاس صفر هستند.

حال برای رسم نمودار از plot.pie استفاده می کنیم و نمودار را با عنوان درست نشان می دهیم:

```
plt.figure(figsize=(8, 6))
new_y.value_counts().plot.pie(autopct = "%.2f")
plt.title("the distribution of samples in each class")
plt.legend(labels=new_y.value_counts().index, title="Class Labels",
loc="upper right")
```

نتایج:



همانطور که طبق محاسبات بالا ونمودار بالا دیدیم مشخص است که اکثر داده ها متعلق به کلاس صفر هستند، پس تعداد نمونه های کلاس ها با هم برابر نیست، پس این مجموعه داده نامتعادل است.

عدم تعادل در دیتاست می‌تواند به مشکلات مختلفی منجر شود، از جمله:

❖ مشکل در آموزش مدل:

این موضوع می‌تواند منجر به یادگیری ناکافی برای کلاس‌های کم‌نمونه شود. همانطور که مشخص است در این صورت داده های کلاسی که نمونه های بیشتری دارند یادگیری خیلی بهتری نسبت به کلاس کم نمونه دارند. البته در اینجا تعداد نمونه های دو کلاس تقریبا نزدیک به هم است، در مجموعه داده هایی که این اختلاف بیشتر است، این موضوع خودش را به طور فاحش نشان می دهد، البته همانطورکه در سوال ۱ هم گفتیم برای تست نیز بهتر است تعداد داده های یکسانی از هر دو کلاس انتخاب شوند، که این موضوع را جلوتر پیاده خواهیم کرد.

❖ تاثیرات انحرافی و کاهش دقت:

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) در پیش‌بینی‌ها منجر شود و در نتیجه دقت تخمین‌گر برای کلاس‌های با تعداد نمونه بیشتر بالا می‌رود، اما برای کلاس‌های کم‌نمونه کاهش می‌یابد. در نتیجه دقت و کارایی مدل ممکن است در مواجهه با داده‌های جدید تحت تأثیر قرار گیرد، زیرا مدل ممکن است بر اساس نمونه‌های زیاد یک کلاس و بی‌توجه به کلاس‌های کم‌نمونه باشد.

❖ 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.

برای حل مشکل عدم تعادل داده ها باید از تکنیک هایی همچون افزایش نمونه ها (oversampling) و یا کاهش نمونه ها (undersampling) استفاده کنیم. همچنین برخی از الگوریتم های یادگیری ماشین به طور خودکار تعادل را در داده ها برقرار می کنند

ما برای برقراری تعادل در داده ها از روش undersampling استفاده می کنیم، در این روش از بین داده های کلاس صفر که تعدادشان از داده های کلاس ۱، ۱۵۱ عدد بیشتر است، ۱۵۱ داده را به صورت رندوم حذف می کنیم، برای اینکار به روش زیر عمل می کنیم:

```
! pip install -U imbalanced-learn
from imblearn.under_sampling import RandomUnderSampler

y1 = pd.DataFrame(y)
rus = RandomUnderSampler(sampling_strategy=1, random_state=93)
x_res_undersampling , y_res_undersampling = rus.fit_resample(X , y1)
x_res_undersampling.shape , y_res_undersampling.shape
```

در این روش ابتدا خروجی را در یک dataFrame جدید به نام y1 قرار می دهیم، سپس به کمک undersampling داده های مذکور را به صورت رندوم حذف می کنیم.

انتظار داریم ۱۵۱ داده از کلاس صفر حذف شده باشد. و X و y جدید هر کدام شامل ۱۲۲۰ نمونه باشد.

نتایج:

```
((1220, 4), (1220, 1))
```

که همان انتظار ما است.

اگر این بار داده های هر کلاس را بشماریم، انتظار داریم که داده های هر کلاس برابر ۶۱۰ باشد.

کد:

```
y_res_undersampling.value_counts()
```

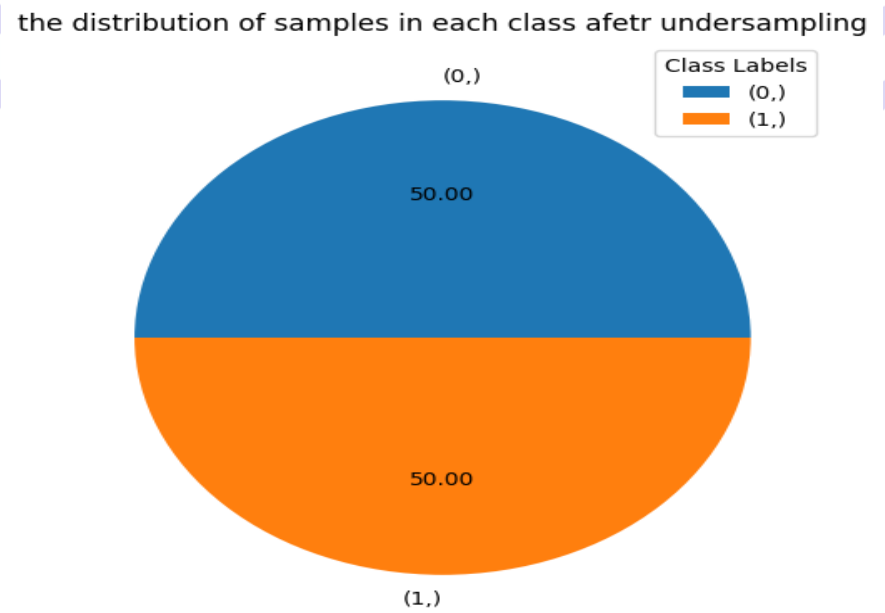
نتایج:

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

اگر این بار نمودار توزیع داده ها را ببینیم، مشخص است که تعادل در داده ها وجود دارد.

```
plt.figure(figsize=(8, 6))
y_res_undersampling.value_counts().plot.pie(autopct = '%.2f')
plt.title("the distribution of samples in each class afetr undersampling")
plt.legend(labels=new_y.value_counts().index, title="Class Labels",
loc="upper right")
```

نتایج:



بخش ۷

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

همانطور که دیدیم در بخش ۷ توانستیم چالش عدم تعادل داده ها را حل کنیم، پس تنها کافیست به کمک یک طبقه بند آماده پایتونی داده ها را طبقه بندی کنیم، در بخش ۶ توانستیم X و y متناظر با داده های متعادل را به دست بیاوریم، پس تنها کافیست به کمک یکی از classifier ها مثلا logistic Regression داده ها را طبقه بندی کنیم که همانند سوال ۱ عمل می کنیم:

همانطور که دیدیم برای این داده ها نیاز به نرمالیزه کردن نداریم، همچنین برای رسیدن به نتایج بهتر تعداد داده یکسانی از هر کلاس برای فرآیند ارزیابی انتخاب می کنیم و همچنین از random_state برای ایجاد تکرار پذیری استفاده می کنیم. بنابراین کد ما به شکل زیر خواهد بود:

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

X_train, X_test, y_train, y_test = train_test_split(X_res_undersampling,
y_res_undersampling, test_size=0.2, random_state=93,
stratify=y_res_undersampling)
```

```
LogReg_classifier = LogisticRegression(C=1.0, penalty='l2', solver='sag',
max_iter=20000, random_state=93)
LogReg_classifier.fit(X_train, y_train)
y_pred = LogReg_classifier.predict(X_test)
LogReg_train_accuracy = LogReg_classifier.score(X_train, y_train)
LogReg_test_accuracy = LogReg_classifier.score(X_test, y_test)
```

```
print(f"Logistic Regression Train Accuracy: {LogReg_train_accuracy:.2f}")
print(f"Logistic Regression Test Accuracy: {LogReg_test_accuracy:.2f}")
```

نتایج:

```
Logistic Regression Train Accuracy: 0.99
Logistic Regression Test Accuracy: 0.99
```

همانطور که مشخص است توانستیم به دقت ۹۹ درصد برای داده های ارزیابی برسیم، پس با متعادل کردن نمونه ها و همچنین رعایت یک سری موارد که در بالا راجع به آن صحبت کردیم، توانستیم به دقت مد نظر برسیم.

سوال ۳

بخش ۱

۱- به این پیوند مراجعه کرده و یک دیتاست مربوط به بیماری قلبی را دریافت کرده و توضیحات مختصری در مورد هدف و ویژگی های آن بنویسید. فایل دانلود شده دیتاست را روی گوگل درایو خود قرار داده و با استفاده از دستور gdown آن را در مخیط کولب بارگذاری کنید.

ابتدا فایل را از پیوند مربوطه دانلود کرده، در سایت مدنظر از بخش Data Card می توانیم اطلاعات این دیتاست را ببینیم.

این مجموعه داده شامل اطلاعات مختلف مربوط به شاخص های سلامت برای یک نمونه از افراد است، که توضیحات هر ستون آن به شکل زیر است:

- Heart Diseaseor Attack: نشان می دهد که آیا فرد دچار بیماری قلبی یا حمله قلبی شده است (۰ = خیر، ۱ = بله)
- HighBP : وضعیت فشار خون بالا : ۰ = خیر، ۱ = بله)
- HighChol : وضعیت کلسترول بالا (۰ = خیر، ۱ = بله)
- CholCheck : دفعات بررسی کلسترول (طبقه ای)
- BMI : شاخص توده بدن (پیوسته)
- Smoker : وضعیت سیگار کشیدن (۰ = خیر، ۱ = بله)
- Stroke : سابقه سکته مغزی (۰ = خیر، ۱ = بله)
- Diabetes : وضعیت دیابت (دودویی: ۰ = خیر، ۱ = بله)
- PhysActivity : سطح فعالیت بدنی (طبقه ای)
- Fruits : فراوانی مصرف میوه (طبقه ای)
- Veggies : فراوانی مصرف سبزیجات (طبقه ای)
- HvyAlcoholConsump : وضعیت مصرف الکل سنگین (۰ = خیر، ۱ = بله)
- AnyHealthcare : دسترسی به هر مراقبت بهداشتی (۰ = خیر، ۱ = بله)
- NoDocbcCost : بدون پزشک به دلیل هزینه (۰ = خیر، ۱ = بله)
- GenHlth : ارزیابی سلامت عمومی (طبقه ای)
- MentHlth : ارزیابی سلامت روان (طبقه ای)
- PhysHlth : ارزیابی سلامت جسمانی (طبقه ای)

- DiffWalk : وضعیت دشواری راه رفتن (۰ = خیر، ۱ = بله)
- Sex : جنسیت فرد (۰ = زن، ۱ = مرد)
- Age : سن فرد (پیوسته)
- Education : مقطع تحصیلی (طبقه ای)
- Income : سطح درآمد (طبقه ای)

این مجموعه داده شامل اطلاعات متنوع در زمینه سلامت، عوامل سبک زندگی و جمعیت‌شناسی برای یک گروه افراد می‌باشد، که آن را مناسب برای بررسی همبستگی‌ها و عوامل خطر پتانسیلی برای بیماری قلبی و شرایط سلامت دیگر می‌سازد.

این دیتاست را دانلود کرده و سپس آن را در گوگل درایو بارگزاری می کنیم و به کمک دستور gdown آن را در محیط کولب قرار می دهیم.این کار را به کمک کد زیر انجام می دهیم:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 14ko2e1olsQdldRdFtD9A26r6W3mVl7wP
```

همچنین برای ایجاد dataframe نیاز به کتابخانه panda داریم، بنابراین این کتابخانه و چند تا از کتابخانه های دیگر را فراخوانی می کنیم:

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

برای ایجاد dataframe به شکل زیر عمل می کنیم:

```
data = pd.read_csv(r'/content/data_banknote_authentication.txt')
data.to_csv('/content/data_banknote_authentication.csv')
df = pd.read_csv('/content/data_banknote_authentication.csv')
df
```

نتیجه به شکل زیر خواهد بود:

	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk
0	0	1	1	1	40	1	0	0	0	0	...	1	0	5	18	15	1
1	0	0	0	0	25	1	0	0	1	0	...	0	1	3	0	0	0
2	0	1	1	1	28	0	0	0	0	1	...	1	1	5	30	30	1
3	0	1	0	1	27	0	0	0	1	1	...	1	0	2	0	0	0
4	0	1	1	1	24	0	0	0	1	1	...	1	0	2	3	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
253656	0	0	0	1	25	0	0	0	1	1	...	1	0	1	0	0	0
253657	0	0	1	1	24	0	0	0	0	0	...	1	0	3	0	0	0
253658	0	0	0	0	27	0	0	0	1	0	...	1	1	2	0	0	0
253659	0	0	1	1	37	0	0	2	0	0	...	1	0	4	0	0	0
253660	0	0	1	1	34	1	0	0	0	1	...	1	0	3	0	2	1

253661 rows × 22 columns

بخش ۲

۲- ضمن توجه به محل قرارگیری هدف و ویژگی ها، دیتاست را به صورت یک دیتافریم درآورده و با استفاده از دستورات پایتونی، ۱۰۰ نمونه داده مربوط به کلاس ۱ و ۱۰۰ نمونه داده مربوط به کلاس صفر را در یک دیتا فریم جدید قرار دهید و در قسمت های بعدی با این دیتافریم جدید کار کنید.

برای انجام اینکار ابتدا باید داده های دو کلاس را از هم جدا کنیم، این داده ها مربوط به بیماری قلبی می باشند، بنابراین بر اساس اینکه شخص دچار حمله قلبی شده است یا نه از هم جدا می کنیم:

```
# Separate the data into two classes
class_0_data = df[df['HeartDiseaseorAttack'] == 0]
class_1_data = df[df['HeartDiseaseorAttack'] == 1]
```

حال نیاز است که از هر کدام از این کلاس ها ۱۰۰ داده به صورت تصادفی انتخاب کنیم، برای اینکار به صورت زیر عمل می کنیم:

```
# choosing 100 random datas
nclass_0_data = class_0_data.sample(n=100, random_state=93)
nclass_1_data = class_1_data.sample(n=100, random_state=93)
```

همچنین برای اینکه خاصیت تکرار پذیری ایجاد شود از random_state استفاده می کنیم.

حال نیاز است که این دو DataFrame را با هم ترکیب کنیم، برای اینکار به صورت زیر عمل می کنیم:

```
# Combine datas
new_df = pd.concat([nclass_0_data, nclass_1_data], ignore_index=True)
```

از طرفی برای اینکه شماره نمونه ها در DataFrame جدید نیز اصلاح شود ignore_index را برابر با True قرار می دهیم و طبق همان مواردی که در بخش ۲ گفتیم بریا از بین بردن نظم داده ها، داده ها را مخلوط می کنیم:

```
from sklearn.utils import shuffle
df = shuffle(new_df, random_state=93)
df
```

برای ایجاد خاصیت تکرار پذیری نیز از random_state استفاده می کنیم.

نتایج:

	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk
89	0	0	0	1	24	0	0	0	1	1	...	1	0	3	0	30	0
50	0	0	0	1	26	0	0	0	1	1	...	1	0	2	2	0	0
68	0	1	1	1	32	0	0	0	1	0	...	1	0	2	0	0	0
28	0	0	1	1	28	0	0	0	1	1	...	1	0	2	0	0	0
65	0	0	1	0	27	1	0	0	0	1	...	1	0	3	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
97	0	1	0	1	38	0	0	0	0	0	...	1	0	5	0	3	1
152	1	1	1	1	45	0	0	0	1	0	...	1	0	3	5	2	0
132	1	1	0	1	49	0	0	0	1	1	...	1	0	4	4	30	1
91	0	1	0	1	32	1	0	0	1	1	...	1	0	2	0	0	0
165	1	1	1	1	34	0	0	0	1	0	...	1	0	2	0	0	1

200 rows × 22 columns

در بخش های بعد از این دیتافریم جدید استفاده می کنیم.

بخش ۳

۳- با استفاده از حداقل دو طبقه بند آماده پایتون و در نظر گرفتن فراپارامترهای مناسب، دو کلاس موجود در دیتاست را از هم تفکیک کنید. نتیجه دقت آموزش و ارزیابی را نمایش دهید.

برای این طبقه بندی دقیقا همانند همان سوال ۱ عمل می کنیم و از همان Classifier ها که در آنجا استفاده کردیم، استفاده می کنیم. فقط نیاز است پارامترها را تغییر دهیم تا به بهترین دقت ممکن برسیم.

فقط نیاز است فرا پارمترها را به گونه ای تغییر دهیم که به بهترین دقت ممکن برسیم.

ابتدا نیاز است تا feature ها را از target جدا کنیم، برای اینکار به صورت زیر عمل می کنیم:

```
X = df.drop('HeartDiseaseorAttack', axis=1)
y = df['HeartDiseaseorAttack']
X.shape, y.shape
```

نتایج:

```
((200, 21), (200,))
```

به کمک دستور اول همه ی ستون ها به جز ستون target را در آرایه X قرار می دهیم که مربوط به feature ها می باشد. و همچنین target را در آرایه y قرار می دهیم.

ابتدا نیاز است تا داده های آموزش و ارزیابی را از هم جدا کنیم:

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

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=93, stratify=y)

# Display the dimensions of the training and testing sets
print(f'Dimensions of the training features: {X_train.shape}')
print(f'Dimensions of the training target: {y_train.shape}')
print(f'Dimensions of the testing features: {X_test.shape}')
print(f'Dimensions of the testing target: {y_test.shape}')
```

نتایج:

```
Dimensions of the training features: (160, 21)
Dimensions of the training target: (160,)
Dimensions of the testing features: (40, 21)
Dimensions of the testing target: (40,)
```

همانطور که انتظار داشتیم ۲۰ درصد داده ها برای داده های تست انتخاب شدند.

روش Logistic Regression :

فراپارامترها را به صورت زیر انتخاب می کنیم:

```
LogReg_classifier = LogisticRegression(C=1.0, penalty='l2',
solver='newton-cholesky', max_iter=150, random_state=93)
LogReg_classifier.fit(X_train, y_train)
y_pred = LogReg_classifier.predict(X_test)
LogReg_train_accuracy = LogReg_classifier.score(X_train, y_train)
LogReg_test_accuracy = LogReg_classifier.score(X_test, y_test)
```

```
y_pred, y_test
```

نتایج:

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

روش SGD Classifier :

فراپارامترها را به صورت زیر انتخاب می کنیم:

```
SGD_classifier = SGDClassifier(alpha=0.0001, loss='log_loss',
max_iter=150, penalty='l2', random_state=93)
SGD_classifier.fit(X_train, y_train)
y_pred1 = SGD_classifier.predict(X_test)
SGD_train_accuracy = SGD_classifier.score(X_train, y_train)
SGD_test_accuracy = SGD_classifier.score(X_test, y_test)
```

```
y_pred, y_test
```

نتایج:

```
(array([0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0,
        0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1]),
114    1
190    1
29     0
153    1
39     0
110    1
133    1
78     0
52     0
87     0
106    1
44     0
92     0
81     0
169    1
88     0
33     0
187    1
101    1
173    1
131    1
94     0
71     0
15     0
1      0
125    1
184    1
157    1
174    1
141    1
97     0
64     0
105    1
62     0
65     0
37     0
189    1
27     0
183    1
193    1
Name: HeartDiseaseorAttack, dtype: int64)
```

روش Perceptron :

فراپارامترها را به صورت زیر انتخاب می کنیم:

```
perceptron_classifier = Perceptron(alpha=0.0001, penalty='l2',
max_iter=200, random_state=93)
perceptron_classifier.fit(X_train, y_train)
y_pred2 = perceptron_classifier.predict(X_test)
perceptron_train_accuracy = perceptron_classifier.score(X_train, y_train)
perceptron_test_accuracy = perceptron_classifier.score(X_test, y_test)
```

```
y_pred, y_test
```

نتایج:

```
(array([1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
        0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1]),
 114    1
 190    1
 29     0
 153    1
 39     0
 110    1
 133    1
 78     0
 52     0
 87     0
 106    1
 44     0
 92     0
 81     0
 169    1
 88     0
 33     0
 187    1
 101    1
 173    1
 131    1
 94     0
 71     0
 15     0
 1      0
 125    1
 184    1
 157    1
 174    1
 141    1
 97     0
 64     0
 105    1
 62     0
 65     0
 37     0
 189    1
 27     0
 183    1
 193    1
```

بررسی دقت در دادهای test و train:

به منظور نمایش دقت در این داده ها به صورت زیر عمل می کنیم:

```
print(f"Logistic Regression Train Accuracy: {LogReg_train_accuracy:.2f}")
print(f"Logistic Regression Test Accuracy: {LogReg_test_accuracy:.2f}")

print(f"SGD Classifier Train Accuracy: {SGD_train_accuracy:.2f}")
print(f"SGD Classifier Test Accuracy: {SGD_test_accuracy:.2f}")

print(f"Perceptron Train Accuracy: {perceptron_train_accuracy:.2f}")
print(f"Perceptron Test Accuracy: {perceptron_test_accuracy:.2f}")
```

نتایج:

```
Logistic Regression Train Accuracy: 0.84
Logistic Regression Test Accuracy: 0.75
SGD Classifier Train Accuracy: 0.78
SGD Classifier Test Accuracy: 0.75
Perceptron Train Accuracy: 0.71
Perceptron Test Accuracy: 0.78
```

بخش ۴

۴- در حالت استفاده از دستورات آماده سایکیت لرن، آیا راهی برای نمایش نمودار اتلاف وجود دارد؟ پیاده سازی کنید.

برای پیاده سازی این بخش تعداد تکرار های هر روش را برابر ۱ قرار می دهیم، سپس مدل فیت شده بر آن سیستم را با یکبار تکرار به دست می آوریم و اختلاف هدف با مقدار تخمین زده شده را به دست می آوریم. این کار را برای روش SGD Classifier به کمک partial_fit انجام می دهیم.

خطا را به روش log_loss محاسبه می کنیم، برای محاسبه log_loss نیز به مقدار احتمالی داده شده نیاز داریم، که از proba استفاده می کنیم.

بنابراین به صورت زیر عمل می کنیم:

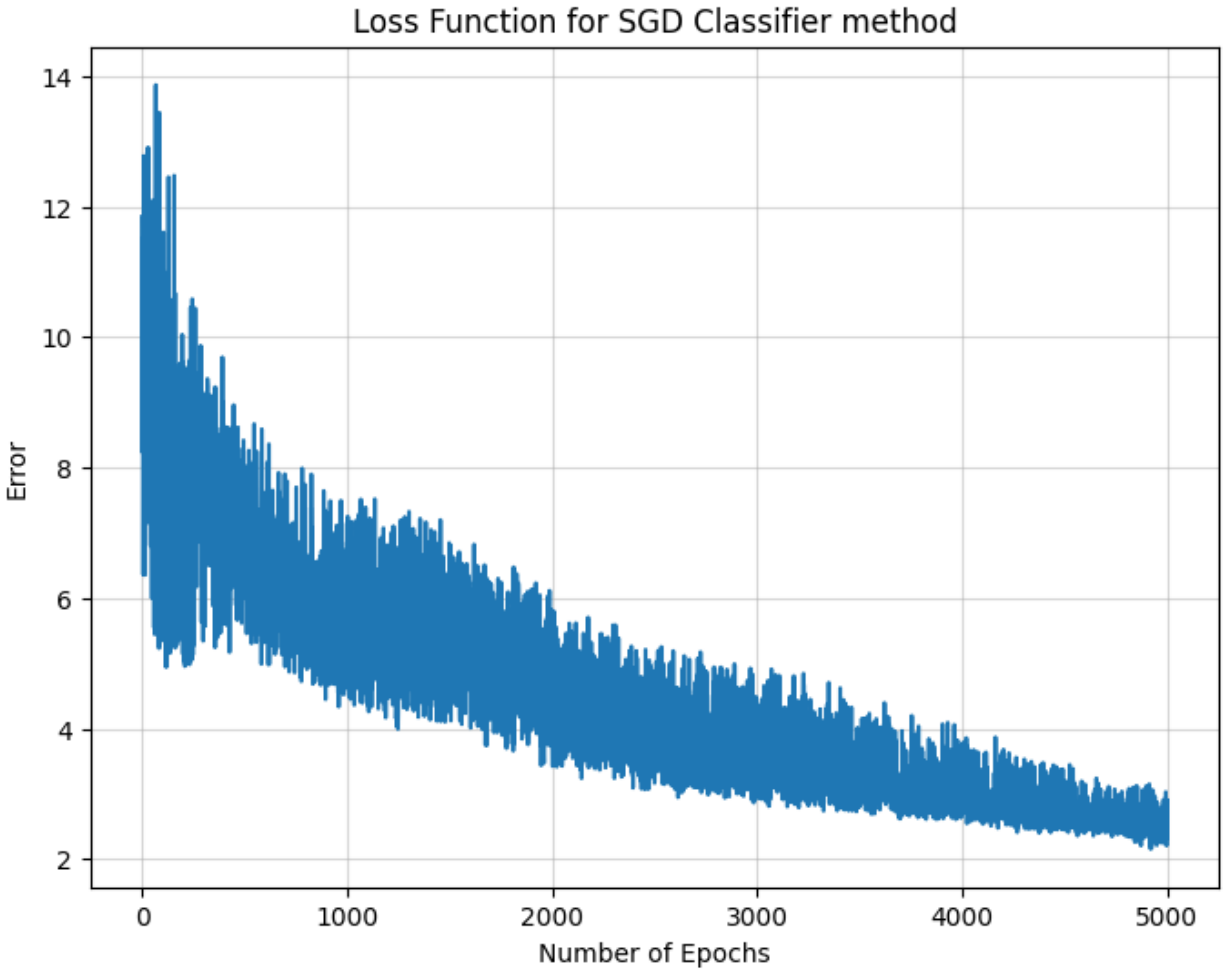
```
from sklearn.metrics import log_loss
error_hist = []
epochs = 5000

for _ in range(epochs):

    SGD_classifier.partial_fit(X_train, y_train, [0, 1])
    loss = log_loss(y_train , SGD_classifier.predict_proba(X_train))
    error_hist.append(loss)

plt.figure(figsize=(8, 6))
plt.plot(error_hist)
plt.xlabel("Number of Epochs")
plt.ylabel("Error")
plt.title('Loss Function for SGD Classifier method')
plt.grid(alpha=0.5)
```

نتایج:



همانطور که مشخص است با طی کردن epoch ها خطا رفته رفته کم می شود.

بخش ۵

۵- یک شاخصه ارزیابی (غیر از Accuracy) تعریف کنید و بررسی کنید که از چه طریقی می توان این شاخص جدید را در ارزیابی داده های تست نمایش داد. پیاده سازی کنید.

از شاخصه confusion matrix استفاده می کنیم که در کلاس درس نیز به آن اشاره شد.

Confusion Matrix یک جدول استفاده شده در مسائل دسته‌بندی برای ارزیابی عملکرد یک مدل یادگیری ماشین است. این جدول خلاصه‌ای از پیش‌بینی‌های یک دسته‌بند بر روی یک مجموعه داده است که مقادیر واقعی آن مجموعه به ما معلوم است. Confusion Matrix به خصوص برای مسائل دسته‌بندی دودویی و چنددسته‌ای مفید است.

Confusion Matrix در دسته بندی باینری:

	Predicted Negative	**Predicted Positive**
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

اصطلاحات کلیدی:

- True Positive (TP) : نمونه‌هایی که در واقعیت مثبت هستند و به درستی به عنوان مثبت پیش‌بینی شده‌اند.
- True Negative (TN) : نمونه‌هایی که در واقعیت منفی هستند و به درستی به عنوان منفی پیش‌بینی شده‌اند.
- False Positive (FP) : نمونه‌هایی که در واقعیت منفی هستند اما به اشتباه به عنوان مثبت پیش‌بینی شده‌اند (خطای نوع I)
- False Negative (FN) : نمونه‌هایی که در واقعیت مثبت هستند اما به اشتباه به عنوان منفی پیش‌بینی شده‌اند (خطای نوع II)

معیارهای برآمده از Confusion Matrix :

❖ دقت(Accuracy) : صحت کلی دسته‌بندی

$$\text{دقت} = \frac{TP + TN}{TP + TN + FP + FN}$$

❖ دقت پیش‌بینی مثبت(Precision) : نسبت مثبت‌های پیش‌بینی شده که واقعاً مثبت هستند.

$$\text{دقت پیش‌بینی مثبت} = \frac{TP}{TP + FP}$$

❖ دقت پیش‌بینی منفی(Specificity یا True Negative Rate) : نسبت منفی‌های واقعی که به درستی پیش‌بینی شده‌اند.

$$\text{دقت پیش‌بینی منفی} = \frac{TN}{TN + FP}$$

❖ و ....

برای پیاده سازی این بخش از کتابخانه sklearn.metrices استفاده می کنیم، همچنین برای نمایش این ماتریس و معیار های ارزیابی از confusion_matrix و ConfusionMatrixDisplay و precision_matrix استفاده می کنیم و معیار های دقت پیش‌بینی مثبت و دقت پیش‌بینی منفی را نیز نمایش می دهیم.

پس به صورت زیر عمل می کنیم:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
precision_score

conf_matrix = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix)
disp.plot()
plt.title('Confusion matrix for Logistic Regression method')
plt.show()
```

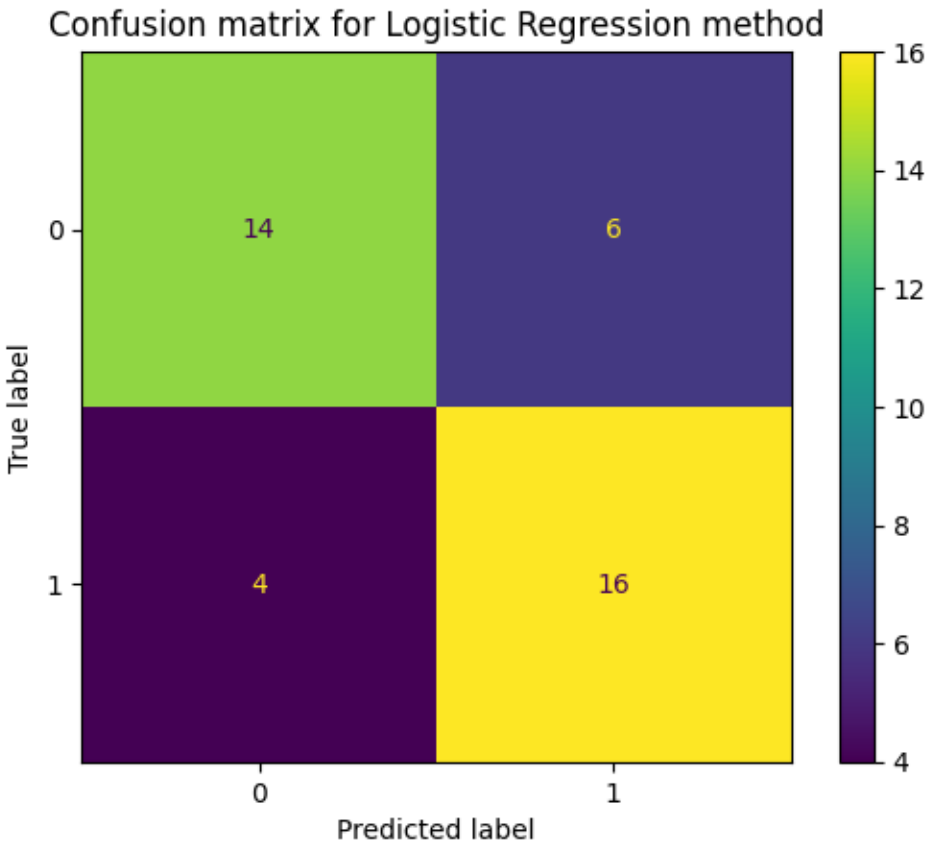
```
precision = precision_score(y_test, y_pred)
specificity = conf_matrix[0, 0] / (conf_matrix[0, 0] + conf_matrix[0, 1])

print("Precision of Logistic Regression method:", precision)
print("Specificity of Logistic Regression method:", specificity)
```

به طور مشابه برای دو روش دیگر نیز به همین صورت عمل می کنیم:

روش Logistic Regression :

نمایش Confusion Matrix :

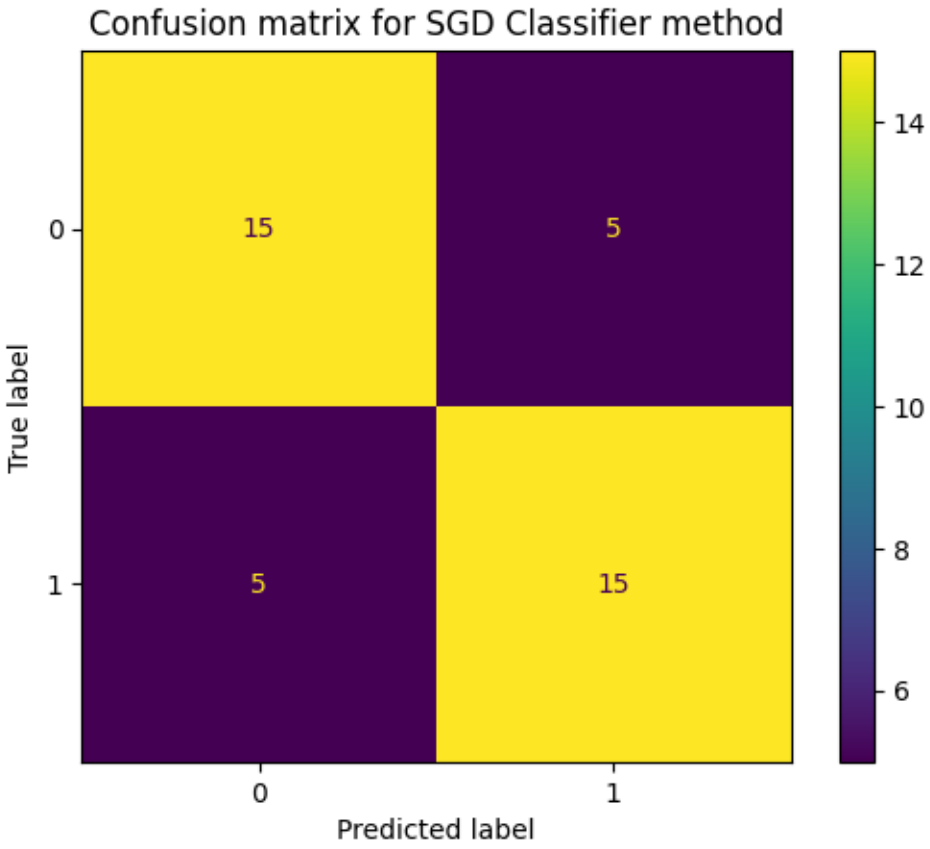


معیار های دقت پیش‌بینی مثبت و دقت پیش‌بینی منفی:

```
Precision of Logistic Regression method: 0.7272727272727273
Specificity of Logistic Regression method: 0.7
```

روش SGD Classifier :

نمایش Confusion Matrix :

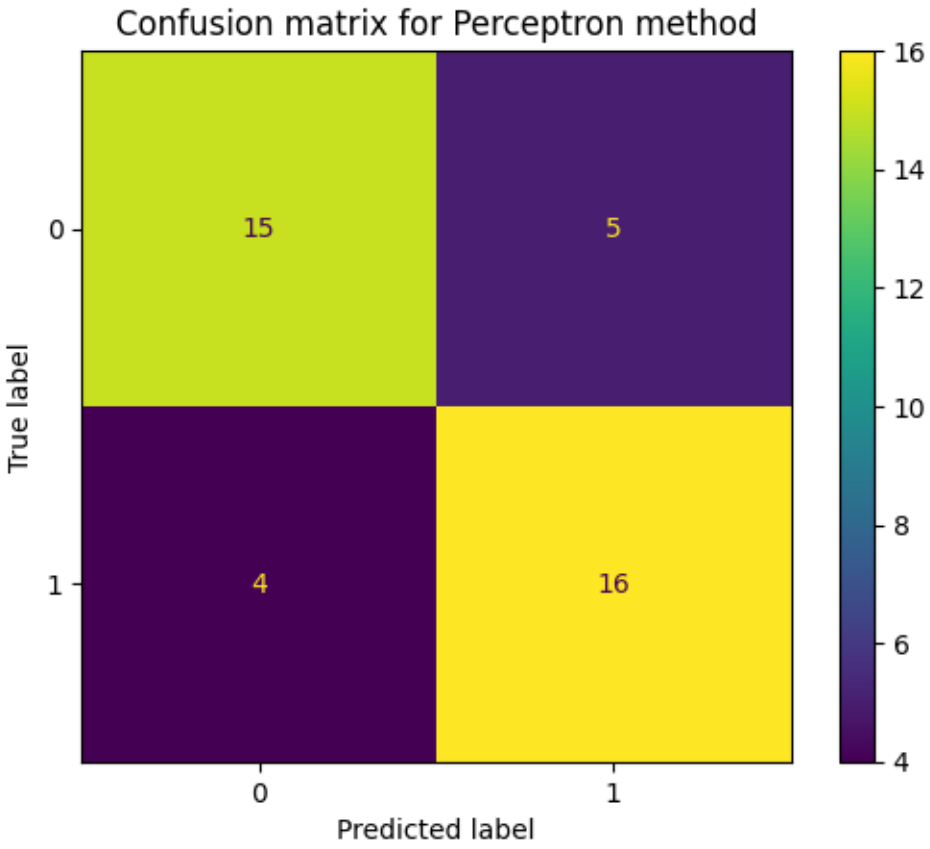


معیار های دقت پیش‌بینی مثبت و دقت پیش‌بینی منفی:

```
Precision of SGD Classifier method: 0.75
Specificity of SGD Classifier method: 0.75
```

روش Perceptron :

نمایش Confusion Matrix :



معیار های دقت پیش‌بینی مثبت و دقت پیش‌بینی منفی:

```
Precision of Perceptron method: 0.7619047619047619
Specificity of Perceptron method: 0.75
```