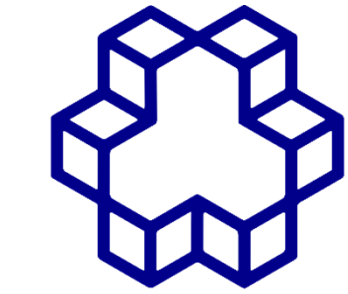
بسمه تعالی



دانشگاه صنعتی خواجه نصیرالدین طوسی

نام و شماره دانشجویی :

علی عبدی

9728463

عنوان :

مینی پروژه اول

تشریح مساله و کدها

سوال اول

بخش اول

❖ در این بخش برای اجرای کدها در ابتدا کتابخانه های مورد نیاز خود را طبق کدهای زیر بارگذاری میکنیم:

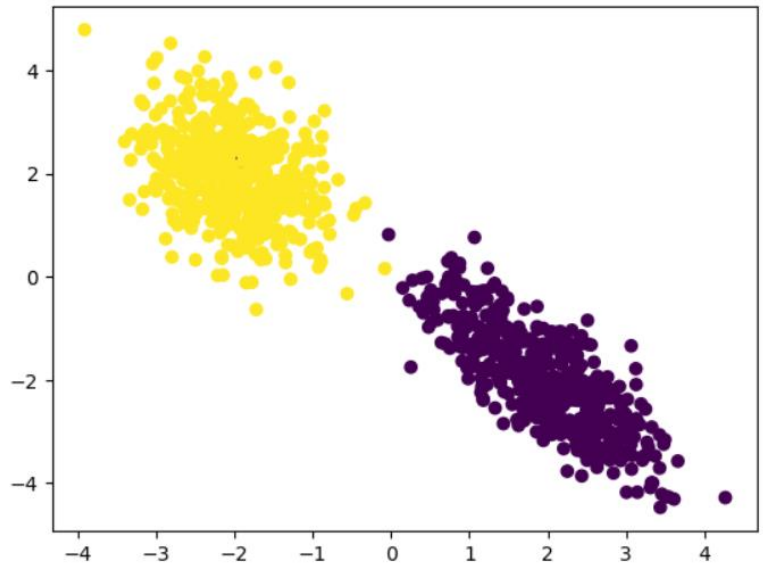
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression, SGDClassifier
from mlxtend.plotting import plot_decision_regions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

❖ حال با توجه به صورت سوال نمونه های مصنوعی خود را با دستور make_classification ایجاد میکنیم و آرگومان های آن را طوری تنظیم میکنیم که داده های ما دارای 1000 نمونه، دو ویژگی و دو کلاس باشند سپس این داده ها را رسم میکنیم که دستورات زیر این کارها را برای ما انجام میدهند:

```
X, y = make_classification(n_samples=1000, n_features=2, n_redundant=0,
n_clusters_per_class=1, class_sep=2, n_classes=2, random_state=27)
print(X.shape, y.shape)

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

که خروجی را به صورت زیر به ما میدهد:

```
(1000, 2) (1000,)
<matplotlib.collections.PathCollection at 0x7f5969360760>
```



بخش دوم

- حال باید داده های خود را با استفاده از روش LogisticRegression و روش SGDClassifier آموزش داده و دقت مدل خود را ارزیابی کنیم، ابتدا از روش اول شروع میکنیم، در ابتدا با استفاده از دستور train_test_split داده های خود را به نسبت 80 به 20 به داده های آموزش و آزمون تقسیم میکنیم، میتوانیم شکل ماتریس ها را با کد زیر ببینیم و همچنین در کد زیر خروجی پیشبینی شده از روش LogisticRegression و خروجی خود داده های تست را مشاهده میکنیم:

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
model = LogisticRegression(solver='sag', max_iter=300, random_state=14,)
model.fit(x_train, y_train)
print(x_train.shape), print(y_train.shape), print(x_test.shape),
print(y_test.shape)
y_pred=model.predict(x_test)
y_pred, y_test
```

که خروجی کد بالا بصورت زیر است:

```
(800, 2)
(800,)
(200, 2)
(200,)
(array([1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 1]),
 array([1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1,
       0, 1]))
```

❖ حال مدل خود را با معیار ارزیابی accuracy روی داده های آموزش و آزمون محاسبه میکنیم که نتایج بصورت زیر است:

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

```
0.9975
```

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

```
0.985
```

❖ حال داده های خود را با روش دوم یعنی روش SGDClassifier آموزش داده و خروجی پیشبینی شده و خروجی واقعی داده آزمون را با استفاده از کدهای زیر خواهیم دید:

```
model1 = SGDClassifier(loss='log_loss',max_iter=300, random_state=27)
model1.fit(x_train, y_train)
y_pred1=model1.predict(x_test)
y_pred1, y_test
```

که خروجی کدهای بالا بصورت زیر است:

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

- حال مدل خود را با معیار ارزیابی accuracy روی داده های آموزش و آزمون محاسبه میکنیم که نتایج بصورت زیر است:

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

```
0.99625
```

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

```
1.0
```

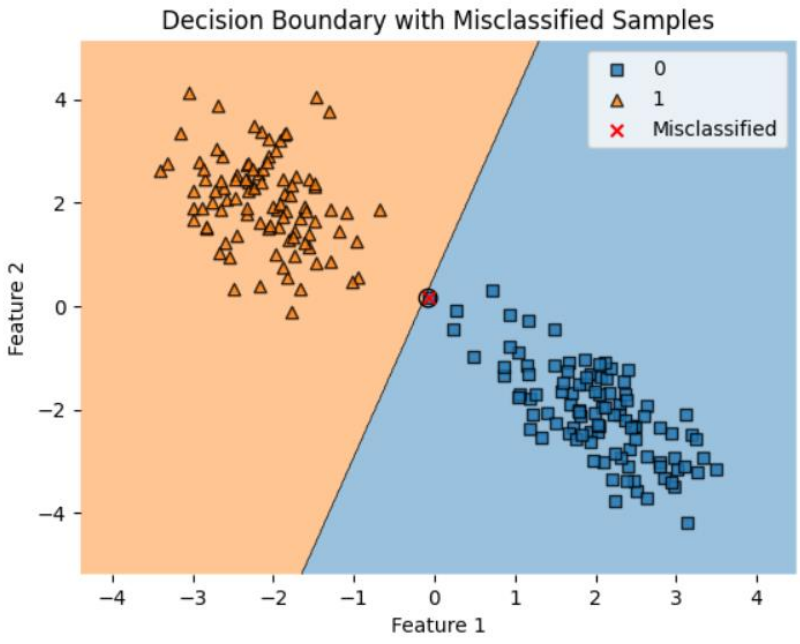
- همانطور که معیارهای ارزیابی ما نشان میدهند مدل ما در هر دو روش بسیارخوب آموزش دیده و اگر این معیار عدد پایینی بود میتوانستیم با بالا بردن تعداد ایپاک ها و یا تغییر بهینه ساز در روش اول و تغییر تابع اتلاف و بالا بردن تعداد ایپاک ها در روش دوم مدل ما بهتر آموزش ببیند و مقادیر معیار ارزیابی افزایش یابد.

بخش سوم

❖ ما در این بخش مرز و نواحی تصمیم‌گیری را برای انواع روش های گفته شده رسم میکنیم، همچنین داده هایی که به اشتباه طبقه‌بندی شده اند را با رنگ قرمز به نمایش درمی آوریم، در روش `LogisticRegression` مرز و نواحی تصمیم گیری با کد و خروجی آورده شده است:

```
plot_decision_regions(x_test, y_pred, clf=model, legend=2,
X_highlight=x_test[y_test != y_pred])
misclassified_indices = np.where(y_test != y_pred)[0]
plt.scatter(x_test[misclassified_indices, 0],
x_test[misclassified_indices, 1], marker='x', color='red',
label='Misclassified')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Boundary with Misclassified Samples')
plt.legend()
plt.show()
```

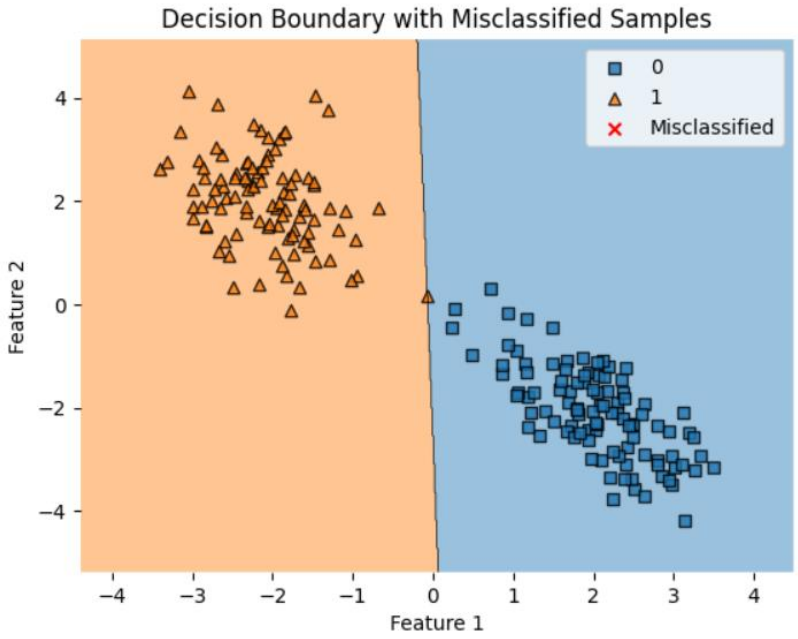
که خروجی کدها بصورت شکل زیر است:



❖ حال مرز و نواحی تصمیم‌گیری را در روش `SGDClassifier` با کد زیر بدست می‌آوریم، داده هایی که به اشتباه طبقه‌بندی شده اند با رنگ قرمز مشخص است:

```
plot_decision_regions(x_test, y_pred1, clf=model1, legend=2,
X_highlight=x_test[y_test != y_pred1])
misclassified_indices = np.where(y_test != y_pred1)[0]
plt.scatter(x_test[misclassified_indices, 0],
x_test[misclassified_indices, 1], marker='x', color='red',
label='Misclassified')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Boundary with Misclassified Samples')
plt.legend()
plt.show()
```

که خروجی کدهای بالا بصورت زیر است:



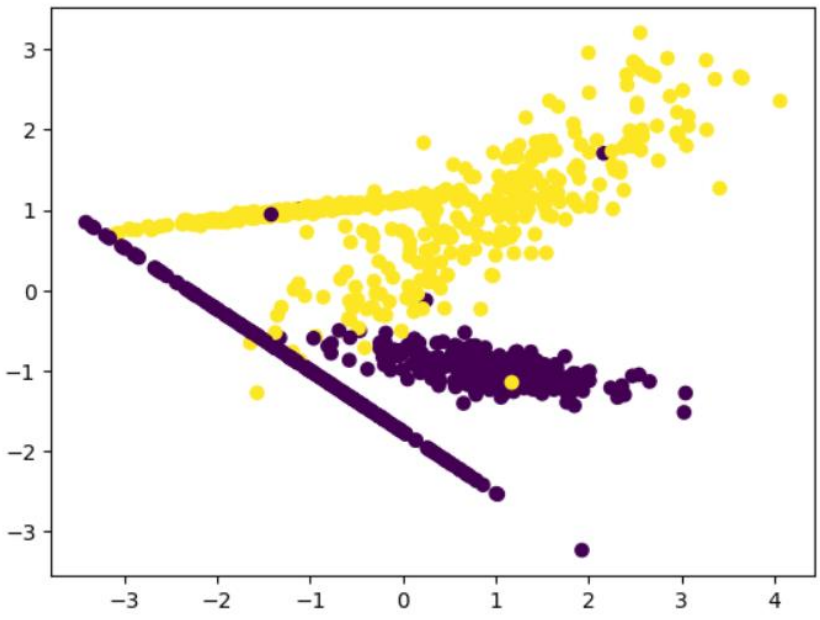
بخش چهارم

- در این بخش برای اینکه دیتاست ما سخت تر و چالش برانگیزتر شود باید در دستور `make_classification` برخی آرگومان‌ها را تغییر دهیم؛ از جمله با کاهش مقدار آرگومان `class_sep` داده های ما تودرتو میشوند و کلاس‌بندی آنها سختتر میشود؛ همچنین اگر مقدار عدد طبیعی آرگومان `n_clusters_per_class` را بالا ببریم، تعداد حالات توزیع داده های ما در هر کلاس بیشتر میشود و کلاس‌بندی آنها سختتر میشود؛ و در نهایت اگر تعداد نمونه های خود را هم افزایش دهیم، کلاس‌بندی سختتر میشود؛ کدهای مربوط به این بخش دقیقا همان کدهای قبلی است و تنها آرگومان های گفته شده را تغییر میدهیم تا دیتاست پیچیده‌تر شود و بخشی از کدهای قبلی بصورت زیر تغییر میکنند:

```
X, y = make_classification(n_samples=1000, n_features=2, n_redundant=0,
n_clusters_per_class=2, class_sep=1, n_classes=2, random_state=27)
print(X.shape, y.shape)

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

که خروجی کدها بصورت زیر است:

```
(1000, 2) (1000,)
<matplotlib.collections.PathCollection at 0x7b692b1d06a0>
```



❖ حال همان کدهای قبلی را وارد خواهیم نمود و در روش LogisticRegression با معیار ارزیابی accuracy دقت مدل ما با توجه به دیتاست جدید بصورت زیر بدست می‌آید:

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

```
0.95375
```

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

```
0.94
```

❖ همانطور که انتظار داشتیم دقت مدل ما با توجه به پیچیده شدن دیتاست پایین آمده است؛ حال همان کدهای قبلی را وارد خواهیم نمود و در روش SGDClassifier با معیار ارزیابی accuracy دقت مدل ما با توجه به دیتاست جدید بصورت زیر بدست می‌آید:

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

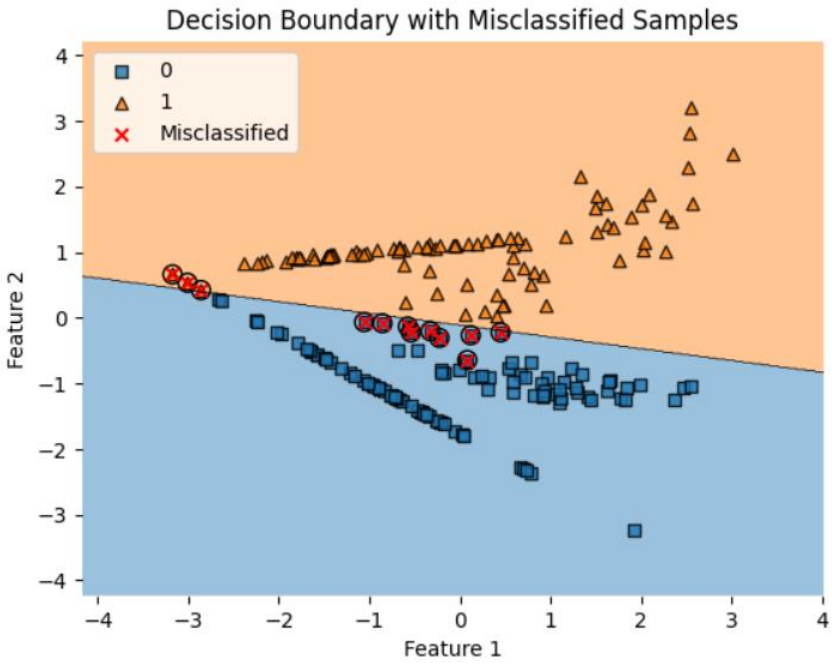
```
0.96125
```

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

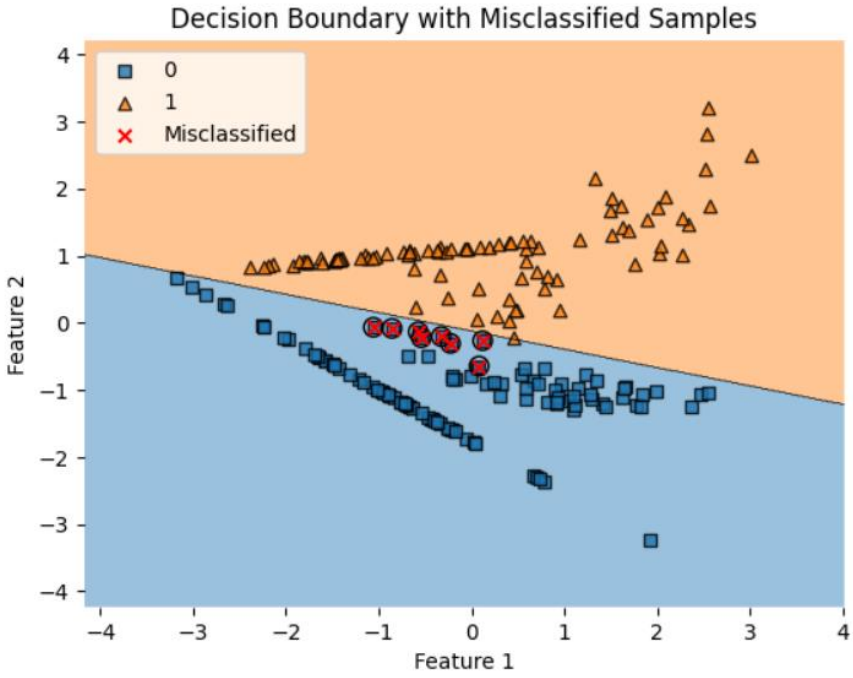
```
0.96
```

❖ در این روش نیز با توجه به انتظاری که داشتیم به علت پیچیده بودن دیتاست دقت مدل پایین آمده است.

❖ مرزها و نواحی تصمیم‌گیری با توجه به دیتاست جدید در روش LogisticRegression به شکل زیر است و همانطور که انتظار داشتیم تعداد داده هایی که به اشتباه کلاس‌بندی شده‌اند افزایش یافته است:



❖ مرزها و نواحی تصمیم‌گیری با توجه به دیتاست جدید در روش `SGDClassifier` به شکل زیر است همانطور که انتظار داشتیم تعداد داده هایی که به اشتباه کلاس‌بندی شده‌اند افزایش یافته است :



بخش پنجم

❖ اگر یک کلاس به دیتاست ما اضافه شود آنگاه خروجی سه کلاسه میشود و طبیعی است که کار کلاس-بند بجای یک خط با چند خط انجام میشود و تعداد پارامترهای ما در محاسبه تابع اتلاف و بهینه‌ساز افزایش می‌یابد و نمونه‌ای از کد روش `LogisticRegression` که from scratch نوشت شده در پایین آمده‌است:

```
import numpy as np

class MyLogisticRegression:
    def __init__(self, learning_rate=0.01, num_epochs=1000):
        self.learning_rate = learning_rate
        self.num_epochs = num_epochs
        self.theta = None

    def _softmax(self, z):
        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return exp_z / np.sum(exp_z, axis=1, keepdims=True)

    def _one_hot_encode(self, y, num_classes):
        m = len(y)
        Y_encoded = np.zeros((m, num_classes))
        Y_encoded[np.arange(m), y] = 1
        return Y_encoded

    def _log_loss(self, h, Y_encoded):
        epsilon = 1e-15
        h = np.clip(h, epsilon, 1 - epsilon)
        return -np.sum(Y_encoded * np.log(h)) / len(Y_encoded)

    def fit(self, X, y):
        m, n = X.shape
        num_classes = len(np.unique(y))
        self.theta = np.zeros((n, num_classes))

        Y_encoded = self._one_hot_encode(y, num_classes)

        for epoch in range(self.num_epochs):
            z = X @ self.theta
            h = self._softmax(z)
            gradient = X.T @ (h - Y_encoded) / m
            self.theta = self.theta - self.learning_rate * gradient
```

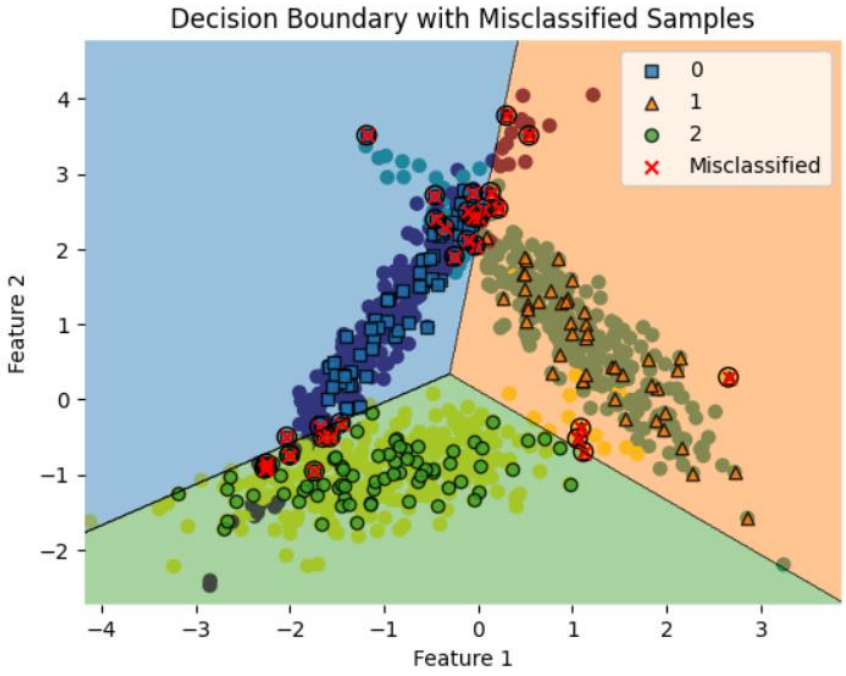
```
            # محاسبه تابع هزینه برای نظارت بر پیشرفت
            cost = self._log_loss(h, Y_encoded)
            print(f"Epoch {epoch + 1}/{self.num_epochs}, Cost: {cost}")

    def predict(self, X):
        z = X @ self.theta
        h = self._softmax(z)
        return np.argmax(h, axis=1)

# مثال استفاده از کلاس MyLogisticRegression با تابع هزینه log-loss
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# ایجاد داده‌های تصادفی با 3 کلاس
X, y = make_classification(n_samples=1000, n_features=2, n_informative=1,
n_classes=3, random_state=42, n_redundant=0, n_repeated=0)

# تقسیم داده به دو بخش آموزش و تست
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# ایجاد و آموزش مدل
model = MyLogisticRegression(learning_rate=0.01, num_epochs=1000)
model.fit(X_train, y_train)

# پیش‌بینی بر روی داده‌های تست
y_pred = model.predict(X_test)

# ارزیابی دقت مدل
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

❖ حال میخواهیم با استفاده از کتابخانه‌های آماده پایتون یک دیتاست سه کلاسه را دسته‌بندی کنیم، برای اینکار کافیست در دستور `make_classification` آرگومان `n_features` را برابر سه قرار بدهیم تا دیتاست مصنوعی سه کلاسه ایجاد شود و سپس در دستور `LogisticRegression` آرگومان `multi_class` را برابر `'ovr'` قرار دهیم، سایر کدها همانند قبل خواهد بود که به شکل زیر است:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import numpy as np
from mlxtend.plotting import plot_decision_regions

# ایجاد داده‌های تصادفی با 3 کلاس
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2,
n_classes=3, n_clusters_per_class=1, random_state=42, n_redundant=0,
n_repeated=0)
plt.scatter(X[:, 0], X[:, 1], c=y)

# تقسیم داده به دو بخش آموزش و تست
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# ایجاد LogisticRegression برای چند کلاس
clf = LogisticRegression(multi_class='ovr', random_state=42)

# آموزش مدل
clf.fit(X_train, y_train)

# پیش‌بینی بر روی داده‌های تست
y_pred = clf.predict(X_test)

# ارزیابی دقت مدل
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# نمایش مرز تصمیم گیری و نمونه‌های اشتباه طبقه‌بندی شده با رنگ قرمز
plot_decision_regions(X_test, y_pred, clf=clf, legend=2,
X_highlight=X_test[y_test != y_pred])
misclassified_indices = np.where(y_test != y_pred)[0]
plt.scatter(X_test[misclassified_indices, 0],
X_test[misclassified_indices, 1], marker='x', color='red',
label='Misclassified')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Boundary with Misclassified Samples')
plt.legend()
plt.show()
```

❖ خروجی کدهای بالا که مرز و نواحی تصمیم‌گیری و همچنین دقت مدل را مشخص میکند بصورت زیر خواهد بود:

```
Accuracy: 0.845
```



سوال دوم

بخش اول

- در این سوال یک دیتاست داریم که خروجی آن براساس یکسری ویژگی تشخیص میدهد اسکناس تقلبی هست یا نه، و دیتاست دارای چهار ویژگی و دو کلاس میباشد، فایل دیتاست به فرمت txt اما ستون ویژگی ها و خروجی نامگذاری نشده است بنابراین ما در همان فایل txt نامگذاری ویژگی ها را بصورت x1,x2,x3,x4 و خروجی را y قرار میدهیم، این ویژگی ها براساس شکل ظاهری و رنگ و جنس اسکناس استخراج شده‌اند و براساس این ها مدل ما باید تشخیص دهد که اسکناس تقلبی است یا خیر؛ حال دیتاست خود را با دستورات زیر بارگذاری کرده و به فرمت csv درمی‌آوریم:

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

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.7.3)
Collecting gdown
  Downloading gdown-5.0.0-py3-none-any.whl (16 kB)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.13.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.11.17)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.7.3
    Uninstalling gdown-4.7.3:
      Successfully uninstalled gdown-4.7.3
Successfully installed gdown-5.0.0
Downloading...
From: https://drive.google.com/uc?id=1Won6xkyYCcJLJ7eMpVt5VA_4P0tE1nb7
To: /content/data_banknote_authentication.txt
100% 46.4k/46.4k [00:00<00:00, 90.7MB/s]
```

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

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

1372 rows × 5 columns

بخش دوم

- در مفهوم یادگیری ماشین و پردازش داده، Data Shuffling به معنای تصادفی کردن یا مخلوط کردن داده‌ها است. این عمل به تصادفی کردن ترتیب داده‌ها در دیتاست اشاره دارد. اهمیت Data Shuffling در پردازش داده و یادگیری ماشین به موارد زیر برمی‌گردد:

- 1. **ممانعت از بیش‌برازش (Overfitting)**: در صورتی که داده‌ها در یک ترتیب خاص و ارتباطات آنها مشخص باشند، مدل ممکن است این ارتباطات را به خاطر بسپارد و برازش زیادی به داده‌های آموزشی داشته باشد. اگر داده‌ها قبل از هر دوره آموزش تصادفی شوند، ممانعت از بیش‌برازش افزایش می‌یابد.

- 2. **تضمین همگرایی بهینه:** Data Shuffling می‌تواند در تضمین همگرایی بهینه مدل کمک کند. اگر داده‌ها به صورت تصادفی به مدل داده شوند، ممکن است فرآیند یادگیری به سمت بهینه‌ترین مقدارها هدایت شود.

- 3. **مطمئن شدن از تعمیم‌پذیری:** اگر مدل تنها بر اساس ترتیب داده‌ها آموزش ببیند، ممکن است در مقابل داده‌های جدید یا داده‌های آزمایشی نتواسته خوب عمل کند. Data Shuffling مطمئن می‌شود که مدل با تنوع کافی از داده‌ها آموزش می‌بیند.

- همانطور که در شکل زیر میبینیم قبل از عمل مخلوط کردن یا بر زدن نمونه‌های ما طوری چیده شده‌اند که ابتدا تمامی نمونه‌های کلاس صفر آمده و سپس تمامی نمونه‌های کلاس یک:

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

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

1372 rows × 5 columns

❖ حال ما با استفاده از دستورات زیر نمونه‌های خود را مخلوط کرده و خروجی را میبینیم:

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

```
            x1        x2        x3        x4  y
1093  0.744280  -3.77230   1.61310  1.575400  1
1030 -1.843900  -8.64750   7.67960 -0.666820  1
1206 -2.434900  -9.24970   8.99220 -0.500010  1
821  -4.017300  -8.31230  12.45470 -1.437500  1
246   1.647200   0.48213   4.74490  1.225000  0
...        ...       ...       ...       ... ..
235   2.046600   2.03000   2.17610 -0.083634  0
770   0.343400   0.12415  -0.28733  0.146540  1
680   3.446500   2.95080   1.02710  0.546100  0
726   0.040498   8.52340   1.44610 -3.930600  0
138   5.241800  10.53880  -4.11740 -4.279700  0

[1372 rows x 5 columns]
```

❖ حال شماره هر نمونه که در ستون اول نوشته شده را در اکسل دوباره مرتب کرده و داده های جدید را بصورت زیر بارگذاری میکنیم:

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

	x1	x2	x3	x4	y
0	0.744280	-3.77230	1.61310	1.575400	1
1	-1.843900	-8.64750	7.67960	-0.666820	1
2	-2.434900	-9.24970	8.99220	-0.500010	1
3	-4.017300	-8.31230	12.45470	-1.437500	1
4	1.647200	0.48213	4.74490	1.225000	0
...	...	...	...	...	...
1367	2.046600	2.03000	2.17610	-0.083634	0
1368	0.343400	0.12415	-0.28733	0.146540	1
1369	3.446500	2.95080	1.02710	0.546100	0
1370	0.040498	8.52340	1.44610	-3.930600	0
1371	5.241800	10.53880	-4.11740	-4.279700	0

1372 rows × 5 columns

❖ حال داده‌های پیش‌پردازش شده موردنظر را به نسبت 80 به 20 به دو بخش آموزش و ارزیابی تقسیم میکنیم:

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

```
(array([[ 0.74428 , -3.7723  ,  1.6131  ,  1.5754  ],
        [-1.8439  , -8.6475  ,  7.6796  , -0.66682 ],
        [-2.4349  , -9.2497  ,  8.9922  , -0.50001 ],
        ...,
        [ 3.4465  ,  2.9508  ,  1.0271  ,  0.5461  ],
        [ 0.040498,  8.5234  ,  1.4461  , -3.9306  ],
        [ 5.2418  , 10.5388  , -4.1174  , -4.2797  ]]),
 array([[1],
        [1],
        [1],
        ...,
        [0],
        [0],
        [0]]))
```

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

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

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

بخش سوم

❖ حال بدون استفاده از کتابخانه‌های آماده پایتون روش آموزش Logistic Regression (from Scratch) را به ترتیب کدهای زیر از ابتدا کدنویسی میکنیم:

❖ در ابتدا تابع سیگموید که روی ورودی دیتاست اعمال میشود را تعریف میکنیم و سپس تابع رگرسیون لجستیک که ورودی‌های آن وزن‌ها و ورودی دیتاست است را تعریف میکنیم:

Logistic Regression (from Scratch)

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

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

❖ حال تابع اتلاف کراس آنتروپی باینری را طبق کد زیر تعریف میکنیم و سپس تابع محاسبه گرادیان:

Binary Cross Entropy (BCE)

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

Gradient

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

❖ حال تابعی برای بروزرسانی وزن‌ها یعنی گرادیان نزولی را تعریف میکنیم و سپس تابعی برای ارزیابی دقت مدل خود موسوم به accuracy تعریف میکنیم:

Gradient Descent

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

Accuracy

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

❖ در بخش قبل داده ها به دو دسته آموزش و ارزیابی تقسیم شده‌اند و حال خروجی پیشبینی شده اولیه را روی داده های ورودی‌ای که میخواهیم آموزش داهیم و وزن‌هایی که بصورت رندم انتخاب شده‌اند بدست می‌آوریم، حال به ورودی خود یک ستون که دارای درایه‌های یک است اضافه میکنیم، این برای این است که ابعاد ماتریس ما با ابعاد وزن‌ها یک شود و در واقع مقدار بایاس نیز حساب شود:

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

```
(1097, 1)
```

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

```
(1097, 5)
```

- حال m را برابر تعداد ویژگی‌ها قرار داده و سپس به وزن‌های خود مقدار اولیه میدهیم، همانطور که در قبل اشاره کردیم ابعاد ماتریس وزن‌ها یکی بیشتر از تعداد ویژگی‌هاست تا پارامتر بایاس نیز حساب شود، در ادامه پارامتر اتا و تعداد ایپاک‌های خود را وارد میکنیم:

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

```
(5, 1)
```

```
eta = 0.01
n_epochs = 60000 #N
```

- در ادامه خطاهای خود را در هر ایپاک محاسبه و ذخیره کرده و همچنین با روش گرادیان نزولی وزن-های مدل خود را بروز میکنیم و همچنین دستوری برای چاپ مقادیر خطا و مقادیر وزن‌ها در هر ایپاک مینویسیم:

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

❖ دستور زیر را نیز برای رسم خطا براساس تعداد ایپاک مینویسیم:

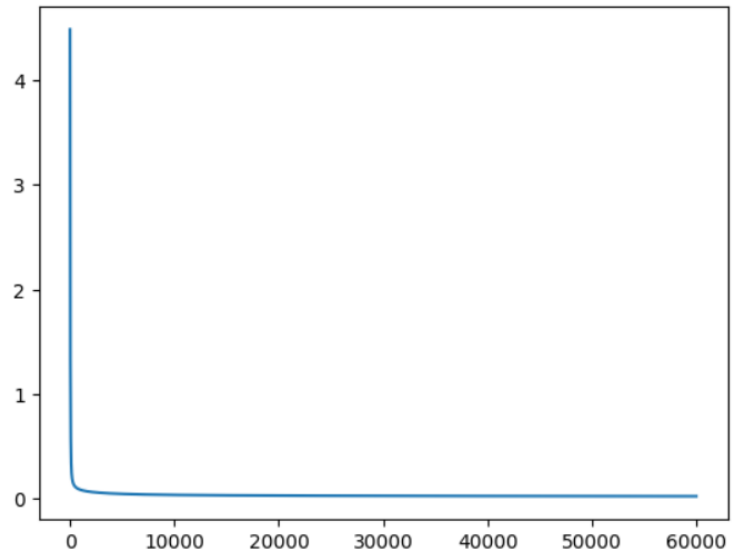
```
plt.plot(error_hist)
```

❖ حال مدل خود را از ابتدا کدنویسی کردیم و میتوانیم براساس این مدل داده‌های خود را آموزش دهیم.

❖ حال با توجه به سوال بخش سوم داده‌های خود را با مدلی که داریم آموزش داده و نمودار تابع اتلاف و نتیجه دقت مدل ما روی داده‌های تست را نمایش میدهیم:

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

```
[<matplotlib.lines.Line2D at 0x7b4c1bae0dc0>]
```

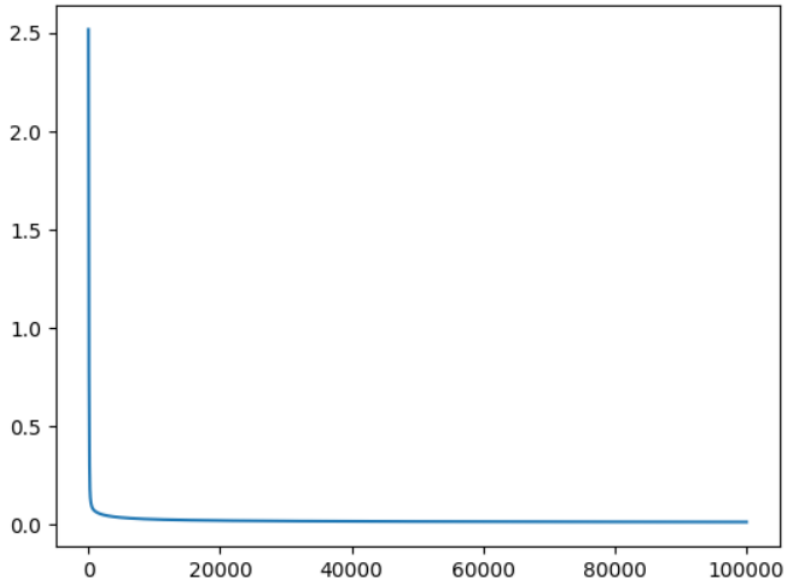



```
x_test = np.hstack((np.ones((len(x_test), 1)), x_test))
y_hat = logistic_regression(x_test, w)
accuracy(y_test, y_hat)
```

```
0.9890909090909091
```

❖ از روی نمودار تابع اتلاف نمیتوان در مورد عملکرد مدل توضیح داد زیرا ممکن است مدل ما به حالت بیش‌برازش برسد و برای اینکه بفهمیم تا چند ایپاک باید محاسبات را ادامه دهیم تا دچار بیش‌برازش نشویم باید داده ها را به دو بخش آموزش و اعتبارسنجی تقسیم کرده و داده‌های اعتبار سنجی را تا

تعداد زیادی ایپاک آموزش دهیم و سپس نمودار تابع اتلاف آنرا رسم کنیم و بعد ببینیم حدودا در چه ایپاکی نمودار تابع اتلاف ما از آن به بعد عدد ثابتی است یا مقدار آن افزایش می یابد، این نقطه عطف به ما نشان میدهد که مدل ما تا چند ایپاک باید روی داده های آموزش محاسبات را انجام دهد؛ بطور پیشفرض نام این تعداد خاص ایپاک را ایپاک نقطه عطف مینامیم؛ ما داده‌ها را به دو دسته آموزش و اعتبارسنجی تقسیم کردیم و داده‌های اعتبارسنجی را آموزش داده و نمودار تابع اتلاف آن را رسم کردیم:



- همانطور که در شکل بالا میبینیم اگر بخواهیم دقیق ایپاک نقطه عطف را بگوییم، این عدد تقریبا برابر 60000 ایپاک است به همین دلیل بود که در قسمت قبلی داده‌های آموزش را تا 60000 ایپاک آموزش دادیم.

بخش چهارم و پنجم

نرمال‌سازی داده‌ها (Normalization):

- نرمال‌سازی داده‌ها یک فرآیند مهم در پیش‌پردازش داده‌های عددی است که هدف آن تبدیل داده‌ها به یک مقیاس مشخص و معقول است. این فرآیند به تعادل و همگرایی بهتر مدل‌های یادگیری ماشین

کمک می‌کند و به ایجاد یک عملکرد بهتر در مدل‌ها کمک می‌کند. در زیر، دو روش معمول برای نرمال‌سازی داده‌ها به همراه اهمیت آنها توضیح داده شده است:

1. نرمال‌سازی مین-مکس (Min-Max Normalization):

روش:

- ❖ - مقادیر هر ویژگی را به یک بازه مشخص تبدیل می‌کند، معمولاً [0,1].
- ❖ - فرمول نرمال‌سازی مین-مکس برای هر ویژگی x به شکل زیر است:

$$x_{\text{normalized}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

اهمیت:

- ❖ - جلوگیری از افزایش و کاهش بیش از حد ویژگی‌ها.
- ❖ - تضمین مقیاس یکسان برای ویژگی‌ها که به مدل کمک می‌کند تا بر روی همه ویژگی‌ها به یک اندازه تاثیر بگذارد.

2. نرمال‌سازی معیاری (Standardization / Z-score Normalization):

روش:

- ❖ - تبدیل داده به یک توزیع نرمال با میانگین ۰ و انحراف معیار ۱.
- ❖ - فرمول نرمال‌سازی معیاری برای هر ویژگی x به شکل زیر است:

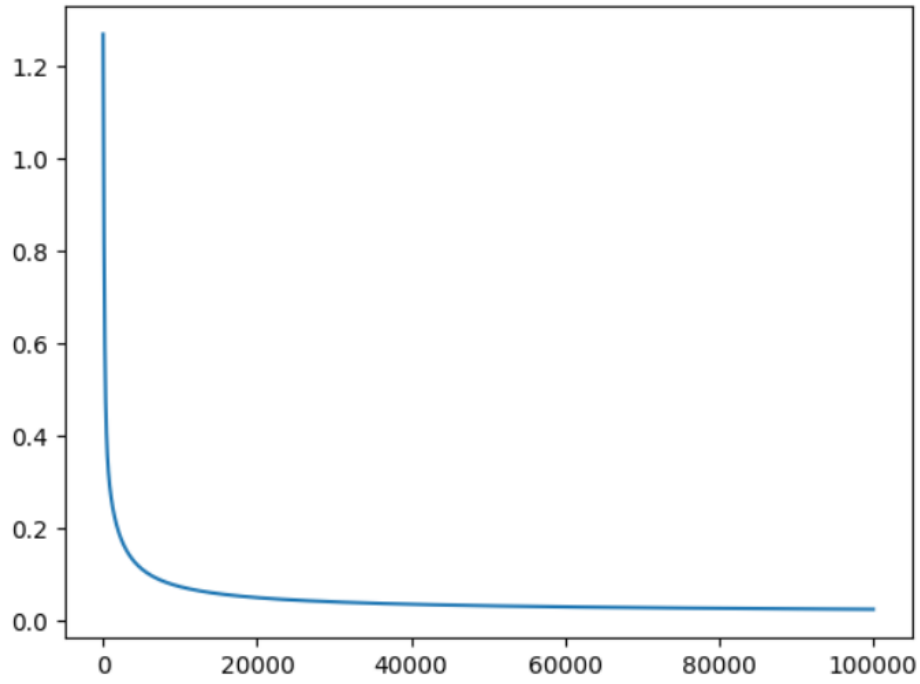
$$x_{\text{standardized}} = \frac{x - \text{mean}(x)}{\text{std}(x)}$$

اهمیت:

- ❖ - تقویت عملکرد مدل در مواجهه با ویژگی‌هایی که توزیع آنها دور از توزیع نرمال است.
- ❖ - تضمین که مقادیر تبدیل شده دارای میانگین صفر و انحراف معیار یک باشند که موجب بهبود همگرایی مدل می‌شود.

- ❖ در پایتون، از کتابخانه Scikit-Learn می‌توان از `MinMaxScaler` برای نرمال‌سازی مین-مکس و از `StandardScaler` برای نرمال‌سازی معیاری استفاده کرد.

❖ ما با روش StandardScaler داده‌های خود را نرمالسازی میکنیم، ما داده‌های بخش ارزیابی یا به اصطلاح test را نیز باید نرمالسازی کنیم زیرا مدل ما براساس مقیاس جدید داده‌ها آموزش دیده و برای ارزیابی مدل نیز باید داده نرمالایز شده به آن بدهیم؛ علاوه بر این ها ما داده‌ها را به دو دسته آموزش و اعتبارسنجی نیز تقسیم کرده و داده‌های اعتبارسنجی را نرمالایز کرده و سپس آموزش داده تا تعداد ایپاک نقطه عطف را بدست آوریم، نمودار تابع اتلاف داده‌های اعتبار سنجی ب شکل زیر درآمده:

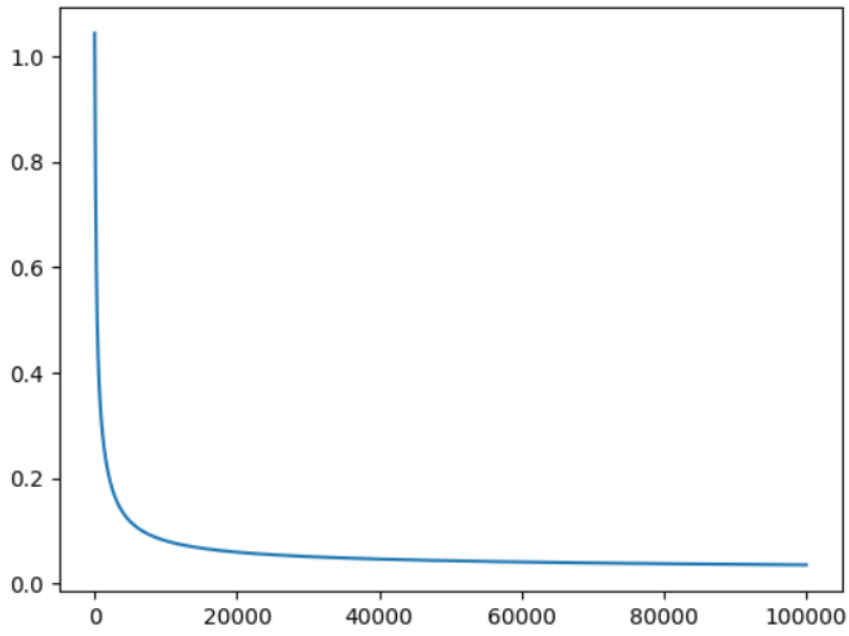


❖ همانطور که در نمودار بالا میبینیم تعداد ایپاک نقطه عطف اگر بخواهیم دقیق شویم حدود 10000 ایپاک است بنابراین ما داده‌های آموزش یا به اصطلاح داده‌های train نرمالایز شده خود را تا این عدد آموزش میدهیم.

❖ مدل ما برای آموزش دقیقا همان مدلی است که در بخش قبل داشتیم با این تفاوت که پس از تقسیم داده‌ها به دو دسته آموزش و ارزیابی کدهای زیر به مدل ما اضافه میشوند:

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

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

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

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

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

❖ حال داده‌های خود را با مدل اصلاح شده آموزش میدهیم، نمودار تابع اتلاف بصورت زیر است:



❖ همچنین دقت مدل ما بصورت روی داده‌های تست بصورت زیر است:

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

```
0.9781818181818182
```

❖ و نتایج پیش بینی مدل برای پنج نمونه داده بصورت زیر است:

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

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

بخش ششم

❖ پس از اینکه مانند بخشهای قبل داده‌های خود را بارگذاری کرده و مخلوط کرده و سپس مرتب میکنیم، حال با دستور زیر تعداد نمونه‌های موجود در هر کلاس را محاسبه میکنیم:

```
class_counts = df2['y'].value_counts()

# نمایش تعداد نمونه‌ها برای هر کلاس
print(class_counts)
```

```
0    762
1    610
Name: y, dtype: int64
```

- همانطور که میبینیم تعداد نمونه‌های کلاس صفر 762 و تعداد نمونه‌های کلاس یک 610 عدد است یعنی تعداد کلاس ها برابر نبوده و بصورت بالانس نشده هستند.
- عدم تعادل تعداد کلاس‌ها در دیتاست ممکن است منجر به مشکلات مختلف در آموزش و ارزیابی مدل‌های یادگیری ماشین شود. برخی از این مشکلات عبارتند از:

1. مدل متمرکز بر کلاس اکثریت:

- مدل ممکن است به سرعت یاد بگیرد که اکثریت داده‌ها را پیش‌بینی کند و در نتیجه، دقت نسبی به کلاس اقلیت کاهش یابد.

2. عملکرد غیرمتناسب:

- در دیتاست‌های غیر متعادل، شاخص‌های ارزیابی مانند دقت (accuracy) ممکن است تا حد زیادی به دلیل تعداد نمونه‌های اکثریت بیشتر، نمایانگر عملکرد واقعی مدل نباشند.

3. اهمیت کلاس‌ها:

- اگر یک کلاس نسبت به دیگری اهمیت بیشتری داشته باشد (مثلاً کلاس اقلیت که ممکن است برای مسائل حیاتی تر باشد)، عدم تعادل می‌تواند به ناپایداری در تصمیم‌گیری مدل منجر شود.

4. داده‌های کمتر برای آموزش:

- کمترین تعداد نمونه‌ها برای کلاس اقلیت ممکن است باعث کاهش توانایی مدل در یادگیری الگوهای مربوط به این کلاس شود.

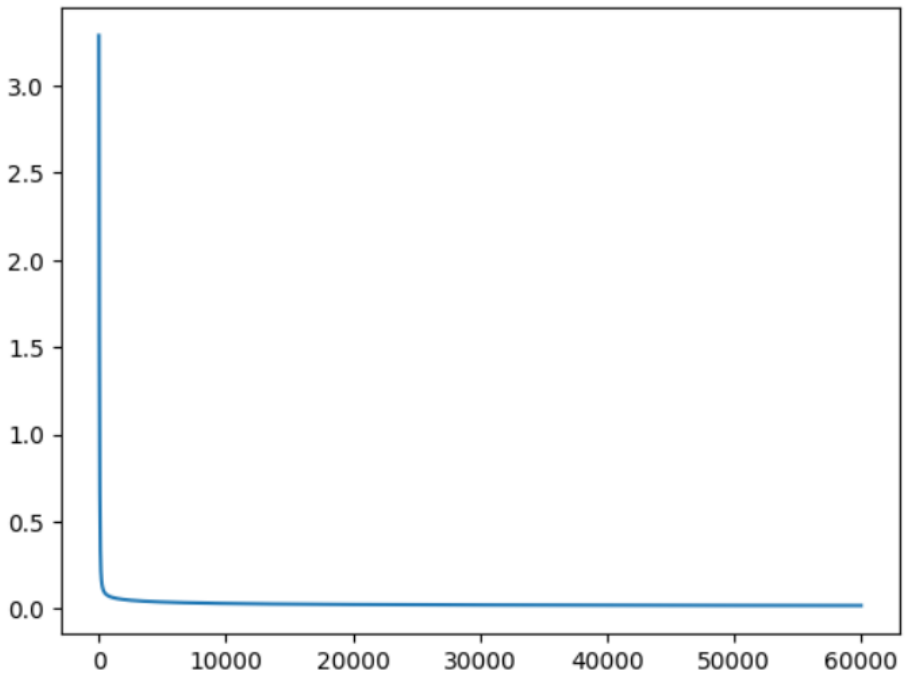
5. مسائل در ارزیابی:

- در ارزیابی مدل، اهمیت تشخیص درست نمونه‌های کلاس اقلیت بالاست. اگر مدل توانایی نداشته باشد که نمونه‌های اقلیت را به درستی تشخیص دهد، ممکن است در مواجهه با مسائل مهم ناتوان باشد.

- حال برای اینکه یک دیتاست جدید تولید کنیم که کلاس‌ها بطور متوازن آموزش ببینند تعداد نمونه-های هر کلاس را برابر عدد 610 قرار میدهیم یعنی برابر تعداد نمونه‌های کلاس کمتر؛ در ادامه با دستورات زیر دیتاست جدید خود را میسازیم:

```
class_0_samples = df2[df2['y'] == 0].sample(610)
class_1_samples = df2[df2['y'] == 1].sample(610)

# ایجاد دیتافریم جدید با 1220 نمونه انتخابی
df1 = pd.concat([class_0_samples, class_1_samples], ignore_index=True)
```

❖ در ادامه داده‌های ما همانند قبل با مدل نوشته شده آموزش خواهند دید؛ نمودار تابع اتلاف در این حالت بصورت زیر بدست می‌آید:



❖ همچنین دقت ما براساس معیار accuracy بصورت زیر بر روی داده‌های تست بدست می‌آید:

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

```
0.9877049180327869
```

❖ دقت بدست آمده چندان با دقت بدست آمده روی داده‌های تست غیرنرمالیزه شده تفاوت ندارد و تقریبا برابر است، این موضوع به دلیل این است که در این مساله تعداد نمونه‌های مربوط به کلاسها تقریبا نزدیک هم است و البته معیار ارزیابی دقت ما accuracy است که این معیار دقت را برای هر کلاس

بطور جداگانه نشان نمیدهد و ما باید از معیارهای دیگری از قبیل Recall و Precision استفاده کنیم که قطعا این معیارها نشان دهنده این خواهند بود که دقت و حساسیت روی داده‌های مربوط به هر کلاس بصورت متوازن درآمده است.

بخش هفتم

- در این بخش میخواهیم با استفاده از کتابخانه‌های آماده داده‌های خود را آموزش دهیم و مشکل نامتوازن بودن کلاسها را حل کنیم؛ دیتاست ما در اینجا همان دیتاست اصلاح شده در بخش های قبل است و این بار با دستور آماده `LogisticRegression` میخواهیم فرآیند آموزش را انجام دهیم و برای حل مشکل نامتوازن بودن کلاس ها کافیست آرگومان `class_weight` را در حالت `'balanced'` قرار دهیم، همچنین تعداد ایپاک ها را عدد 60000 قرار میدهیم؛ مطابق کدهای زیر:

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
model = LogisticRegression(solver='sag',class_weight='balanced', max_iter=60000, random_state=42)
history=model.fit(x_train, y_train)
print(x_train.shape), print(y_train.shape), print(x_test.shape), print(y_test.shape)
y_pred=model.predict(x_test)
y_pred, y_test
```

- دقت مدل ما براساس معیار accuracy بصورت زیر بر روی داده‌های تست بدست می‌آید:

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

```
0.995
```

سوال سوم

بخش اول

- در این سوال دیتاستی که میخواهیم به آن بپردازیم مربوط به تعیین حمله قلبی براساس تعدادی ویژگی از جمله داشتن دیابت، سیگاری بودن، داشتن فعالیت فیزیکی، سن و.... میباشد و ما میخواهیم با مدل‌های مختلفی داده‌های خود را آموزش دهیم.
- در ادامه کتابخانه‌های موردنیاز خود را بارگذاری میکنیم:

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

- حال دیتاست خود را با دستور gdown بارگذاری میکنیم:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1FS-JXMl-PFGzA2ogy1xdBKVI6VbVDQMF
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.7.3)
Collecting gdown
  Downloading gdown-5.0.1-py3-none-any.whl (16 kB)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.13.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.11.17)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.7.3
    Uninstalling gdown-4.7.3:
      Successfully uninstalled gdown-4.7.3
Successfully installed gdown-5.0.1
Downloading...
From: https://drive.google.com/uc?id=1FS-JXMl-PFGzA2ogy1xdBKVI6VbVDQMF
To: /content/heart_disease_health_indicators.csv
100% 11.8M/11.8M [00:00<00:00, 65.9MB/s]
```

❖ در ادامه دیتاست خود را به فرمت یک دیتافریم در می‌آوریم به صورت زیر:

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

	HeartDiseaseorAttack	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	Inc
0	0	1	1	1	40	1	0	0	0	0	...	1	0	5	18	15	1	0	9	4	
1	0	0	0	0	25	1	0	0	1	0	...	0	1	3	0	0	0	0	7	6	
2	0	1	1	1	28	0	0	0	0	1	...	1	1	5	30	30	1	0	9	4	
3	0	1	0	1	27	0	0	0	1	1	...	1	0	2	0	0	0	0	11	3	
4	0	1	1	1	24	0	0	0	1	1	...	1	0	2	3	0	0	0	11	5	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
253656	0	0	0	1	25	0	0	0	1	1	...	1	0	1	0	0	0	0	4	6	
253657	0	0	1	1	24	0	0	0	0	0	...	1	0	3	0	0	0	0	7	5	
253658	0	0	0	0	27	0	0	0	1	0	...	1	1	2	0	0	0	0	3	6	
253659	0	0	1	1	37	0	0	2	0	0	...	1	0	4	0	0	0	0	6	4	
253660	0	0	1	1	34	1	0	0	0	1	...	1	0	3	0	2	1	0	7	4	

253661 rows × 22 columns

❖ همانطور که در شکل بالا قابل مشاهده است ستون هدف یا خروجی ما که قاعدتا باید ستون آخر باشد، در ستون اول است، برای حل این مشکل باید ستون اول را با ستون آخر جابجا کنیم؛ ابتدا با دستور زیر نام ستون‌ها را عوض کرده و جابجا میکنیم:

```
df = df.rename(columns={'Income': 'HeartDiseaseorAttack', 'HeartDiseaseorAttack': 'Income'})
df
```

	Income	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	HeartDiseaseorAtt
0	0	1	1	1	40	1	0	0	0	0	...	1	0	5	18	15	1	0	9	4	
1	0	0	0	0	25	1	0	0	1	0	...	0	1	3	0	0	0	0	7	6	
2	0	1	1	1	28	0	0	0	0	1	...	1	1	5	30	30	1	0	9	4	
3	0	1	0	1	27	0	0	0	1	1	...	1	0	2	0	0	0	0	11	3	
4	0	1	1	1	24	0	0	0	1	1	...	1	0	2	3	0	0	0	11	5	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
253656	0	0	0	1	25	0	0	0	1	1	...	1	0	1	0	0	0	0	4	6	
253657	0	0	1	1	24	0	0	0	0	0	...	1	0	3	0	0	0	0	7	5	
253658	0	0	0	0	27	0	0	0	1	0	...	1	1	2	0	0	0	0	3	6	
253659	0	0	1	1	37	0	0	2	0	0	...	1	0	4	0	0	0	0	6	4	
253660	0	0	1	1	34	1	0	0	0	1	...	1	0	3	0	2	1	0	7	4	

253661 rows × 22 columns

❖ حال مقادیر داده‌های هر ستون را با دستور زیر جابجا میکنیم:

```
df['Income'], df['HeartDiseaseorAttack'] = df['HeartDiseaseorAttack'].copy(), df['Income'].copy()
df
```

	Income	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	HeartDiseaseorAtt
0	3	1	1	1	40	1	0	0	0	0	...	1	0	5	18	15	1	0	9	4	
1	1	0	0	0	25	1	0	0	1	0	...	0	1	3	0	0	0	0	7	6	
2	8	1	1	1	28	0	0	0	0	1	...	1	1	5	30	30	1	0	9	4	
3	6	1	0	1	27	0	0	0	1	1	...	1	0	2	0	0	0	0	11	3	
4	4	1	1	1	24	0	0	0	1	1	...	1	0	2	3	0	0	0	11	5	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
253656	8	0	0	1	25	0	0	0	1	1	...	1	0	1	0	0	0	0	4	6	
253657	3	0	1	1	24	0	0	0	0	0	...	1	0	3	0	0	0	0	7	5	
253658	5	0	0	0	27	0	0	0	1	0	...	1	1	2	0	0	0	0	3	6	
253659	1	0	1	1	37	0	0	2	0	0	...	1	0	4	0	0	0	0	6	4	
253660	3	0	1	1	34	1	0	0	0	1	...	1	0	3	0	2	1	0	7	4	

253661 rows × 22 columns

بخش دوم

❖ در این بخش صد نمونه از کلاس صفر و صد نمونه از کلاس یک جدا کرده و یک دیتاست جدیدی به فرمت دیتافریم بدست می‌آوریم:

```
# جدا کردن ویژگی‌ها و خروجی
features = df.iloc[:, :-1]  # ستونهای ۲ تا آخر به عنوان ویژگی‌ها
output = df.iloc[:, -1]    # ستون اول به عنوان خروجی

# ایجاد دیتافریم جدید برای هر کلاس
class_0_samples = df[output == 0].head(100)
class_1_samples = df[output == 1].head(100)

# ادغام داده‌های دو کلاس در یک دیتافریم جدید
new_df = pd.concat([class_0_samples, class_1_samples], ignore_index=True)
new_df
```

	Income	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	HeartDiseaseorAttack
0	3	1	1	1	40	1	0	0	0	0	...	1	0	5	18	15	1	0	9	4	0
1	1	0	0	0	25	1	0	0	1	0	...	0	1	3	0	0	0	0	7	6	0
2	8	1	1	1	28	0	0	0	0	1	...	1	1	5	30	30	1	0	9	4	0
3	6	1	0	1	27	0	0	0	1	1	...	1	0	2	0	0	0	0	11	3	0
4	4	1	1	1	24	0	0	0	1	1	...	1	0	2	3	0	0	0	11	5	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
195	5	1	1	1	25	1	0	2	1	1	...	1	0	2	1	0	0	0	11	6	1
196	6	0	1	1	29	0	0	0	0	1	...	1	0	5	0	30	1	1	13	5	1
197	5	1	0	1	31	0	0	2	0	0	...	1	0	5	0	30	1	1	13	4	1

❖ همانطور که در شکل بالا قابل مشاهده است در ستون خروجی ابتدا نمونه‌های کلاس صفر چیده شده و سپس نمونه‌های کلاس یک، بنابراین نیاز داریم تا به دلایل گفته شده در سوال قبل با دستور shuffle عملیات مخلوط کردن یا بر زدن را روی دیتاست جدید اعمال کنیم؛ با دستورات زیر اینکار انجام خواهد شد:

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

❖ حال دیتاست مخلوط شده را به فرمت دیتافریم درآورده و از این به بعد با این دیتاست کار خواهیم نمود:

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

	Income	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	Diabetes	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Education	HeartDiseaseorAttack
0	3	0	0	1	26	1	0	0	1	1	...	0	0	1	0	1	0	1	4	5	0
1	7	0	0	1	26	1	0	0	0	0	...	1	0	3	0	15	0	0	7	5	0
2	7	1	1	1	32	0	0	2	1	0	...	1	0	5	30	30	0	0	7	5	1
3	1	1	1	1	25	0	1	2	1	1	...	1	1	4	5	15	1	0	11	3	1
4	7	1	1	1	29	1	0	2	1	1	...	1	0	3	0	0	0	1	11	5	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
195	3	1	1	1	23	1	1	2	0	1	...	1	1	1	2	0	0	0	7	5	1
196	8	0	0	1	25	1	0	2	1	1	...	1	0	3	0	0	0	1	13	6	0
197	8	0	0	1	22	1	0	0	1	1	...	1	0	3	0	0	0	1	7	4	1
198	7	1	1	1	26	1	0	2	0	1	...	1	0	4	0	0	0	0	10	6	1

بخش سوم

❖ در این بخش میخواهیم داده‌های خود را با دو روش `SGDClassifier` و `LogisticRegression` آموزش دهیم؛ در ابتدا با روش `SGDClassifier` داده‌های خود را آموزش میدهیم؛ ابتدا ماتریس ویژگی‌ها و خروجی را میسازیم و سپس آنها را به دو دسته داده آموزش و ارزیابی تقسیم کرده و به روش `SGDClassifier` آموزش میدهیم؛ با دستورات زیر:

```
X = new_df.iloc[:, :-1]
y = new_df.iloc[:, -1].values.reshape(-1,1)
X.shape,y.shape
```

```
((200, 21), (200, 1))
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
model = SGDClassifier(loss='log_loss',max_iter=20000, random_state=27)
y_train = y_train.ravel()
y_test = y_test.ravel()
model.fit(x_train, y_train)
```

❖ همانطور که در کدهای بالا قابل مشاهده است نوع تابع اتلاف را 'log_loss' تعیین کرده و تعداد ایپاک‌ها نیز عدد 20000 تعیین شده است و در نهایت دقت مدل ما بر روی داده‌های آموزش و ارزیابی بصورت زیر میباشد:

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

```
0.70625
```

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

```
0.75
```

❖ علت پایین بودن دقت مدل میتواند این باشد که بعضی از داده‌های مربوط به کلاس‌ها تودرتو هستند و کلاس‌بندی این داده‌ها به روش خطی دقت پایینی دارد.

❖ در ادامه میخواهیم داده‌های خود را با روش LogisticRegression آموزش دهیم که کد این بخش بصورت زیر است:

```
X = df1.iloc[:, :-1]
y = df1.iloc[:, -1].values.reshape(-1,1)
X.shape,y.shape
```

```
((200, 21), (200, 1))
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
y_train = y_train.ravel()
y_test = y_test.ravel()
model = LogisticRegression(solver='liblinear', max_iter=1000, random_state=42)
model.fit(x_train, y_train)
```

❖ در ادامه دقت مدل خود را روی داده‌های آموزش و ارزیابی بدست خواهیم آورد:

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

```
0.75625
```

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

```
0.75
```

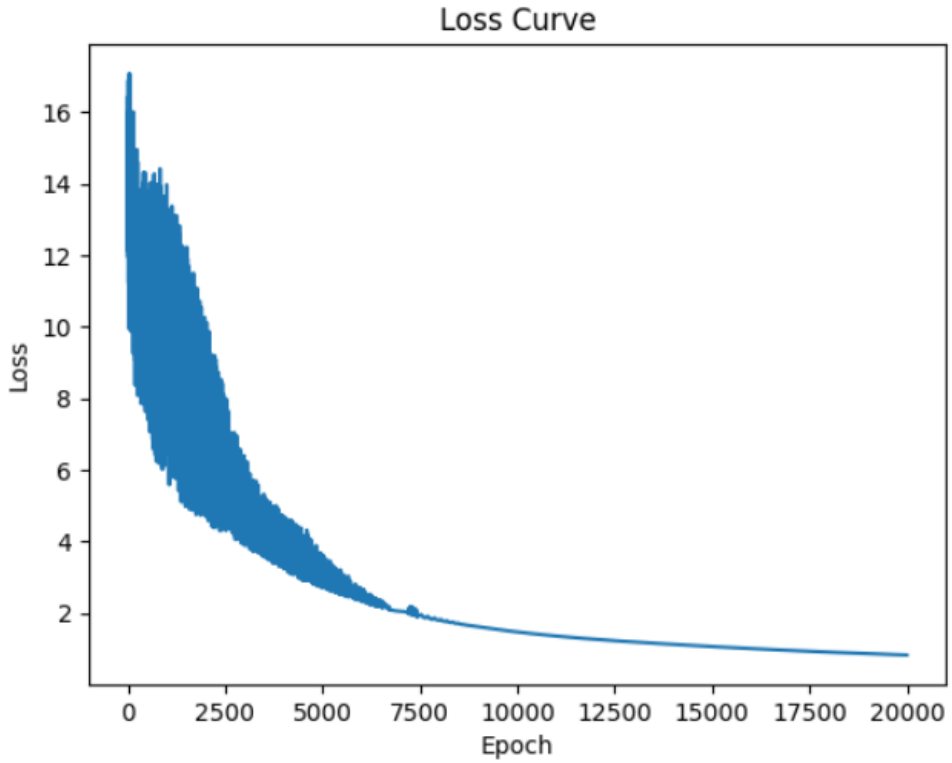
- همانطور که انتظار داشتیم به دلایل گفته شده دقت ما در این روش نیز مقدار پایینی است.

بخش چهارم

- در برخی از روش‌های کلاس بندی از قبیل MLPClassifier مقادیر خطا در هر ایپاک با استفاده از اتریبیوت loss_curve_ بدست آمده و قابل رسم است اما متاسفانه در دو روش SGDClassifier و LogisticRegression چنین اتریبیوتی در نسخه‌های جدید وجود ندارد که مقادیر خطا در هر ایپاک را در خود ذخیره کند و ما باید کد آنرا بصورت دستی وارد کنیم که ما در روش SGDClassifier مقادیر خطا در هر ایپاک را بدست آورده و ذخیره کرده و نمودار آنرا رسم کردیم:

```
# آموزش مدل و دریافت مقدار تابع اتلاف در هر تکرار
loss_history = [];

for epoch in range(20000):
    # آموزش مدل
    model.partial_fit(x_train, y_train, classes=np.unique(y))

    # پیش‌بینی احتمالات
    y_prob = model.predict_proba(x_train);

    # محاسبه تابع اتلاف
    loss = log_loss(y_train, y_prob);

    loss_history.append(loss);


# نمایش نمودار تغییرات تابع اتلاف در هر تکرار
plt.plot(loss_history)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss Curve')
plt.show()
```

❖ که خروجی کدهای بالا بصورت نمودار زیر است:



بخش پنجم

❖ معیارهای دیگری بجز accuracy نیز وجود دارند که میتوانیم مدل خود را بآنها بسنجیم، یکی از این معیارها ماتریس درهم‌ریختگی است که با کمک آن میتوانیم معیارهای recall و precision را که به ترتیب میزان حساسیت و دقت را روی هر کلاس نشان میدهند بدست آوریم، حال این معیارها را در روش SGDClassifier با استفاده از کدهای زیر بدست می‌آوریم:

```
y_pred=model.predict(x_test)
import seaborn as sns
from sklearn.metrics import confusion_matrix, precision_score, recall_score
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
# محاسبه recall 0 برای کلاس
recall_class_0 = recall_score(y_test, y_pred, pos_label=0)
print(f"Recall for Class 0: {recall_class_0}")
```

```
# محاسبه precision 0 برای کلاس
precision_class_0 = precision_score(y_test, y_pred, pos_label=0)
print(f"Precision for Class 0: {precision_class_0}")

# محاسبه recall 1 برای کلاس
recall_class_1 = recall_score(y_test, y_pred, pos_label=1)
print(f"Recall for Class 1: {recall_class_1}")

# محاسبه precision 1 برای کلاس
precision_class_1 = precision_score(y_test, y_pred, pos_label=1)
print(f"Precision for Class 1: {precision_class_1}")
```

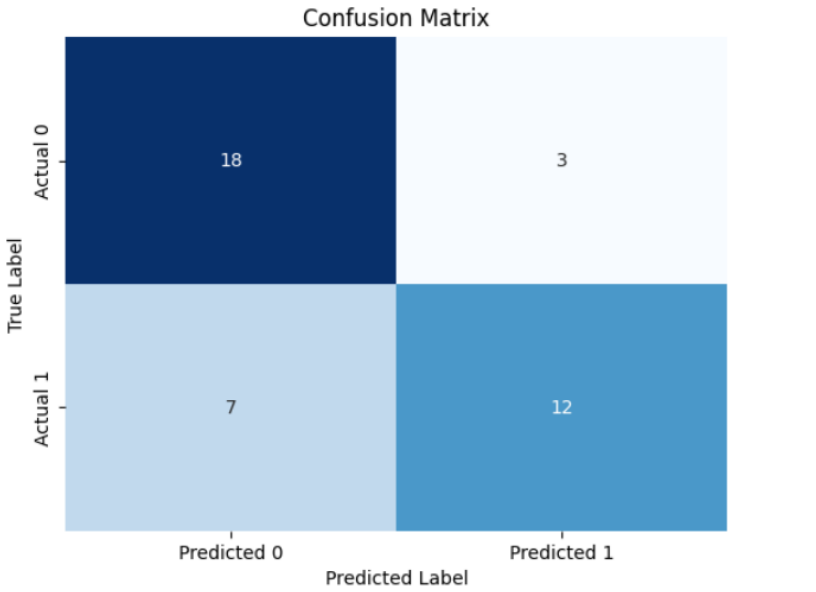
❖ که خروجی کدهای بالا بصورت زیر است:

```
Confusion Matrix:
[[18  3]
 [ 7 12]]
Recall for Class 0: 0.8571428571428571
Precision for Class 0: 0.72
Recall for Class 1: 0.631578947368421
Precision for Class 1: 0.8
```

❖ همانطور که میبینیم در خروجی ماتریس درهم‌ریختگی را داریم که براساس مقادیر درایه‌های آن میزان حساسیت و دقت کلاس‌ها بدست آمده، برای کلاس صفر میزان حساسیت و دقت به ترتیب برابر 0.8571 و 0.72 میباشد و همچنین برای کلاس یک میزان حساسیت و دقت به ترتیب برابر 0.6315 و 0.8 است.

❖ در ادامه حتی میتوانیم ماتریس درهم‌ریختگی را بصورت زیر رسم کنیم:

```
conf_matrix = confusion_matrix(y_test, y_pred)

# رسم ماتریس درهم‌ریختگی با استفاده از سی‌بورن
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])

plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

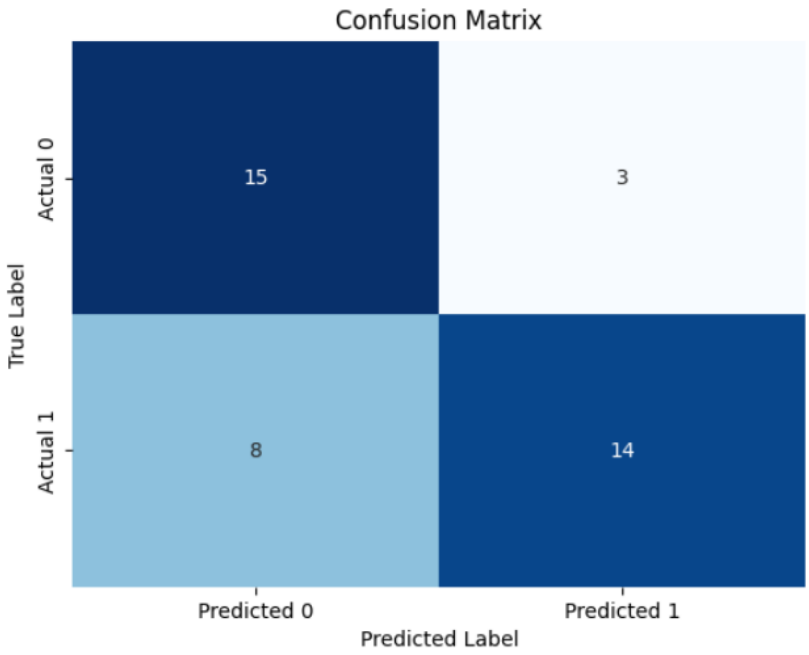


❖ حال کدهای گفته شده را نیز دقیقا برای بدست آوردن معیارهای اشاره شده در روش LogisticRegression پیاده‌سازی کرده که این بار برای این روش خروجی بصورت زیر است:

```
Confusion Matrix:
[[13  7]
 [ 3 17]]
Recall for Class 0: 0.65
Precision for Class 0: 0.8125
Recall for Class 1: 0.85
Precision for Class 1: 0.7083333333333334
```

❖ خب میبینیم که معیارهای حساسیت و دقت در این روش با توجه به ماتریس درهم‌ریختگی بدست آمده‌اند، برای کلاس صفر میزان حساسیت و دقت به ترتیب برابر 0.65 و 0.8125 میباشد و همچنین برای کلاس یک میزان حساسیت و دقت به ترتیب برابر 0.85 و 0.7083 است.

❖ حال ماتریس درهم‌ریختگی را رسم میکنیم:


Confusion Matrix
Actual 0
Actual 1
True Label
15
3
8
14
Predicted 0
Predicted 1
Predicted Label

پایان