((به نام خداوند بخشنده و مهربان))

نام و نام خانوادگی: ساناز گرامی

شماره دانشجویی: 9929873

درس: مبانی سیستم‌های هوشمند

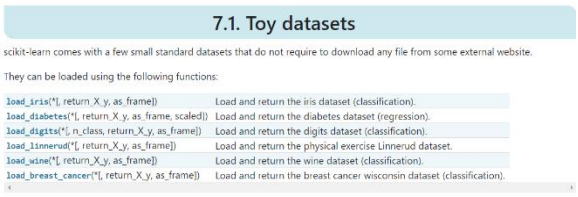
استاد: دکتر مهدی علیاری

مینی‌پروژه1

سوال اول

1. با استفاده از sklearn.datasets، یک دیتاست با 1000 نمونه، 2 کلاس و 2 ویژگی تولید کنید.

ابتدا قبل از هرچیز کتابخانه‌های pandas، matplotlib.pyplot و numpy را فراخوانی می‌کنیم. سپس با استفاده از sklearn.datasets و از میان دیتاست‌های موجود (شکل1)، دیتاست digits را ایمپورت و در X, y بارگذاری می‌کنیم. در مرحله بعد با استفاده از دستور make_classification، یک طبقه‌بندی تشکیل می‌دهیم و طبق اطلاعات داده شده در صورت سوال، n_samples = 1000، n_features = 2، n_classes = 2، random_state = 73 (دو رقم آخر شماره دانشجویی) قرار می‌دهیم. پارامترهای n_dundant = 0، n_cluster_per_class = 1، class_sep = 2.0 را دلخواه انتخاب می‌کنیم. در آخر نمونه‌های تولید شده را با استفاده از دستور plot.scatter نمایش می‌دهیم (شکل2).



شکل1: فهرست دیتاست‌های scikit-learn

شکل2: تولید دیتاست و نمایش داده‌ها

2. با استفاده از حداقل دو طبقه‌بندی آماده‌ی پایتون و در نظر گرفتن فراپارامترهای مناسب، دو کلاس موجود در دیتاست قسمت قبلی را از هم تفکیک کنید. ضمن توضیح روند انتخاب فراپارمترها (مانند تعداد دوره آموزش و نرخ یادگیری)، نتیجه دقت و آموزش و ارزیابی را نمایش دهید. برای بهبود نتیجه از چه تکنیک‌هایی استفاده کردید؟

برای تفکیک نمونه‌ها به دو بخش train و test، تابع train_test_split از کتابخانه sklearn.model_selection را فراخوانی می‌کنیم. سپس ویژگی‌ها (X) و تارگت‌ها (y) را به دو بخش train و test با test_size = 0.25 تقسیم می‌کنیم.

در این سوال، از سه طبقه‌بندی آماده LogisticRegression و SGDClassifier و Perceptron استفاده خواهیم کرد. بدین منظور باید از کتابخانه sklearn.linear_model، این طبقه‌بندی‌ها را ایمپورت کرد (شکل3).

```
#1.2

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, Perceptron, SGDClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

شکل3: ایمپورت طبقه‌بندی‌های آماده

```
# LogisticRegression
model1 = LogisticRegression(random_state = 73)
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + '\n')

the accuracy of LogisticRegression is: 0.992
```

شکل4: طبقه‌بندی LogisticRegression

:LogisticRegression

به منظور طبقه‌بندی به روش LogisticRegression مدل شماره1 با random_state = 73 را تعریف می‌کنیم. تعداد دوره آموزش با پارامتر max_iter مشخص می‌شود که مقدار پیش‌فرض آن 100 است. ابتدا با استفاده از همین مقادیر پیش‌فرض طبقه‌بندی را انجام می‌دهیم. با استفاده از دستور model1.fit، مدل LogisticRegression را با داده‌ها فیت می‌کنیم. سپس با استفاده از model1.predict، لیبل کلاس نمونه‌های X_test را پیش‌بینی کرده و در نهایت با استفاده از دستور model1.score، دقت پیش‌بینی را نمایش می‌دهیم (شکل4).

برای یافتن تاثیر پارامتر max_iter بر روی دقت طبقه‌بندی، یکبار آن را به 50 و بار دیگر به 200 تغییر می‌دهیم. همانطور که در شکل5 نشان داده شده است، تفاوتی در میزان دقت مشاهده نمی‌شود و می‌توان در این مثال از همان مقدار پیش‌فرض 100 برای پارامتر max_iter استفاده کرد. در مرحله بعد، مقدار پارامتر test_size را از

0.25 به 0.35 و سپس به 0.15 تغییر می‌دهیم تا تاثیر آن را بر میزان دقت ببینیم (شکل6). هرچه test_size کوچکتر باشد، دقت بالاتر است.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.35)
model1 = LogisticRegression(random_state = 73)
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + " (test_size = 0.35)" + '\n')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15)
model1 = LogisticRegression(random_state = 73)
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + " (test_size = 0.15)" + '\n')
```

```
the accuracy of LogisticRegression is: 0.9907692307692307 (test_size = 0.35)

the accuracy of LogisticRegression is: 0.9929411764705882 (test_size = 0.15)
```

شکل6: بررسی تاثیر پارامتر test_size

```
# LogisticRegression
model1 = LogisticRegression(random_state = 73)
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + " (max_iter = 100)" + '\n')

model1 = LogisticRegression(random_state = 73, max_iter = 50)
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + " (max_iter = 50)" + '\n')

model1 = LogisticRegression(random_state = 73, max_iter = 200)
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + " (max_iter = 200)" + '\n')
```

```
the accuracy of LogisticRegression is: 0.9933333333333333 (max_iter = 100)

the accuracy of LogisticRegression is: 0.9933333333333333 (max_iter = 50)

the accuracy of LogisticRegression is: 0.9933333333333333 (max_iter = 200)
```

شکل5: بررسی تاثیر پارامتر max_iter

پارامتر دیگری که می‌تواند در دقت طبقه‌بندی تاثیر داشته باشد، پارامتر penalty است. این پارامتر نوع تنظیم اعمال شده به مدل را تعیین می‌کند. منظم‌سازی تکنیکی است که به کاهش پیچیدگی مدل و بهبود توانایی تعمیم آن کمک می‌کند. در LogisticRegression، مقدار پیش‌فرض این پارامتر ’l2‘ است. از آنجایی که Solver lbfg تنها از عبارات ’l2‘ و ’none‘ برای این پارامتر پشتیبانی می‌کند، یکبار مقدار پنالتی را ’noun‘ می‌گذاریم و میزان دقت را با حالت ’l2‘ مقایسه می‌کنیم (شکل7). همانطور که مشاهده می‌شود در حالت penalty = ’none‘ کمی دقت کاهش پیدا کرده است.

:Perceptron

با استفاده از تجربیات طبقه‌بندی قبل، طبقه‌بندی Perceptron را انجام می‌دهیم (شکل8).

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15)
model1 = LogisticRegression(random_state = 73, penalty = 'none')
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + " (penalty = 'none')" + '\n')
```

```
the accuracy of LogisticRegression is: 0.992

the accuracy of LogisticRegression is: 0.991764705882353 (penalty = 'none')
```

شکل7: بررسی تاثیر penalty

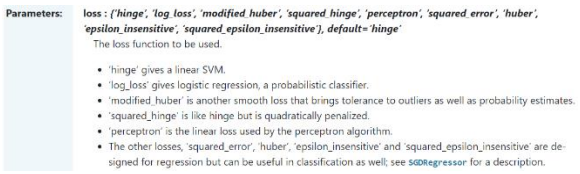
```
# Perceptron
model2 = Perceptron(random_state = 73, penalty = 'l2')
model2.fit(X_train, y_train)
model2.predict(X_test), y_test
a2 = model2.score(X_train, y_train)
print("the accuracy of Perceptron is: " + str(a2) + '\n')
```

```
the accuracy of LogisticRegression is: 0.9906666666666667

the accuracy of Perceptron is: 0.992
```

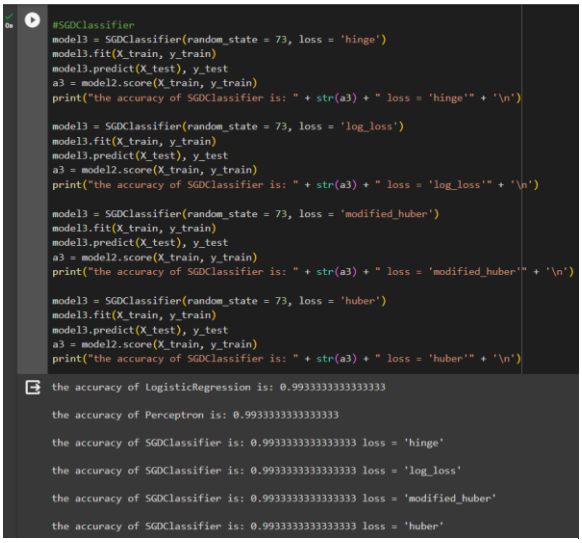
شکل8: طبقه‌بندی Perceptron

:SGDClassifier

پارامتر متفاوتی که در این قسمت وجود دارد loss است که تعیین‌کننده تابع ضرر است و می‌تواند موارد آورده شده در شکل9 را بپذیرد؛ اما همانطور که در شکل10 دیده می‌شود، هیچ‌یک بر دیگری برتری ندارد و دقت همه آنها یکسان است.



شکل9: پارامتر loss



شکل10: طبقه‌بندی SGDClassifier

---

3. مرز و نواحی تصمیم‌گیری برآمده از مدل آموزش دیده خود را به همراه نمونه‌ها در یک نمودار نشان دهید. اگر می‌توانید نمونه‌هایی که اشتباه طبقه‌بندی شده‌اند را با شکل متفاوت نشان دهید.

به منظور رسم نواحی تصمیم‌گیری، ابتدا ماکسیمم و مینیمم داده در هر ویژگی را به ترتیب با استفاده از دستورات x_i_min و x_i_max در راستای یک بعد (بعد صفرم) پیدا می‌کنیم. سپس با استفاده از دستور linspace تعداد n نقطه (n عدد دلخواه است) را میان مقادیر مینیمم و ماکسیمم بدست می‌آوریم. چون می‌خواهیم تعداد زوج نقطه داشته باشیم، استفاده از دستور meshgrid نیز لازم است.

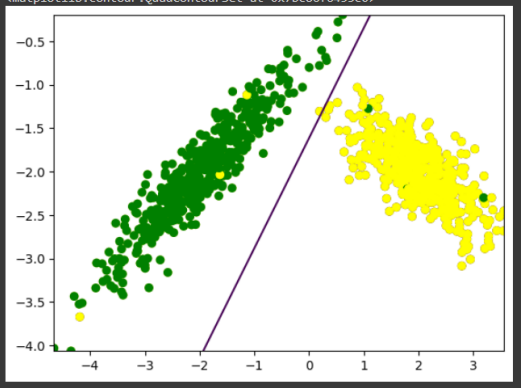
سپس با استفاده از دستور x_i.flatten()، داده‌ها را flat کرده و با استفاده از دستور stack، آنها را به هم می‌چسبانیم. در نهایت از دستور decision_function استفاده کرده (شکل11) و مرز و نواحی تصمیم‌گیری را به ترتیب با استفاده از دستورات scatter و contour برای هر طبقه‌بندی رسم می‌کنیم (شکل‌های12، 13 و 14).

```python
# 1.3

x1_min, x2_min = X.min(0)
x1_max, x2_max = X.max(0)
n = 500
x1r = np.linspace(x1_min, x1_max, n)
x2r = np.linspace(x2_min, x2_max, n)
x1m, x2m = np.meshgrid(x1r, x2r)
Xm = np.stack((x1m.flatten(), x2m.flatten()), axis = 1)
```

شکل11: کدنویسی رسم مرز و نواحی تصمیم‌گیری

```python
ym = model1.decision_function(Xm)
colors = np.array(['yellow', 'green'])
plt.scatter(X[:, 0], X[:, 1], c = colors[y])
plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels = [0])
```

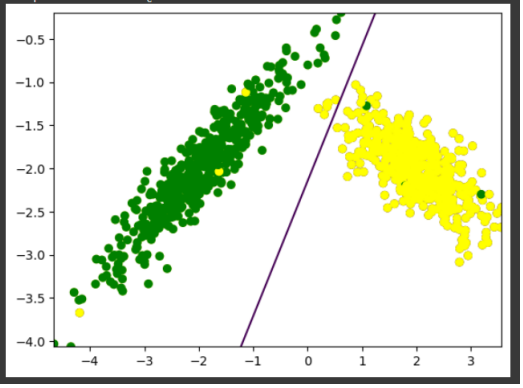
```
the accuracy of LogisticRegression is: 0.992

the accuracy of Perceptron is: 0.9906666666666667

the accuracy of SGDClassifier is: 0.9906666666666667

<matplotlib.contour.QuadContourSet at 0x7bc86f6453c0>
```



شکل12: مرز و نواحی تصمیم‌گیری طبقه‌بندی LogisticRegression

```python
ym = model2.decision_function(Xm)
colors = np.array(['yellow', 'green'])
plt.scatter(X[:, 0], X[:, 1], c = colors[y])
plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels = [0])
```

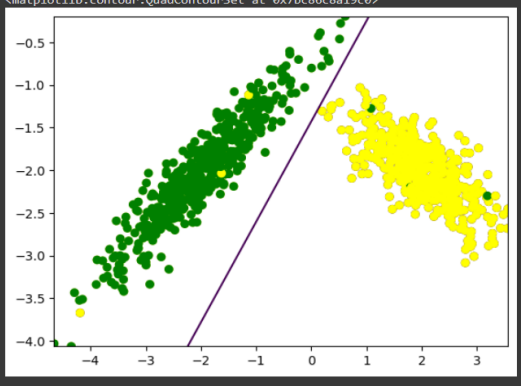
```
the accuracy of LogisticRegression is: 0.9946666666666667

the accuracy of Perceptron is: 0.988

the accuracy of SGDClassifier is: 0.988

<matplotlib.contour.QuadContourSet at 0x7bc86f125540>
```



شکل13: مرز و نواحی تصمیم‌گیری طبقه‌بندی Perceptron

```python
ym = model3.decision_function(Xm)
colors = np.array(['yellow', 'green'])
plt.scatter(X[:, 0], X[:, 1], c = colors[y])
plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels = [0])
```

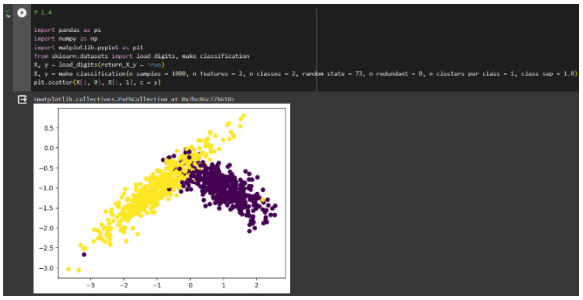
```
the accuracy of LogisticRegression is: 0.9933333333333333

the accuracy of Perceptron is: 0.9946666666666667

the accuracy of SGDClassifier is: 0.9946666666666667

<matplotlib.contour.QuadContourSet at 0x7bc86c8a19c0>
```



شکل14: مرز و نواحی تصمیم‌گیری طبقه‌بندی SGDClassifier

4. از چه طریقی می‌توان دیتاست تولید شده در قسمت1 را چالش‌برانگیزتر و سخت‌تر کرد؟ این کار را انجام داده و قسمت‌های2 و 3 را برای این داده‌های جدید تکرار و نتایج را مقایسه کنید.

اگر در تابع make_classification مقدار پارامتر class_sep را کوچکتر انتخاب کنیم، آنگاه جدا ساختن کلاس‌ها از یکدیگر سخت‌تر خواهد شد. در این مرحله، مقدار class_sep را از 2.0 به 1.0 تغییر می‌دهیم (شکل15).



شکل15: تعریف دیتاست با class_sep = 1.0

```
# LogisticRegression
model1 = LogisticRegression(random_state = 73)
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + '\n')
```

```
the accuracy of LogisticRegression is: 0.9426666666666667
```

شکل16: طبقه‌بندی LogisticRegression

```
# Perceptron
model2 = Perceptron(random_state = 73, penalty = 'l2')
model2.fit(X_train, y_train)
model2.predict(X_test), y_test
a2 = model2.score(X_train, y_train)
print("the accuracy of Perceptron is: " + str(a2) + '\n')
```

```
the accuracy of LogisticRegression is: 0.9333333333333333

the accuracy of Perceptron is: 0.9226666666666666
```

شکل17: طبقه‌بندی Perceptron

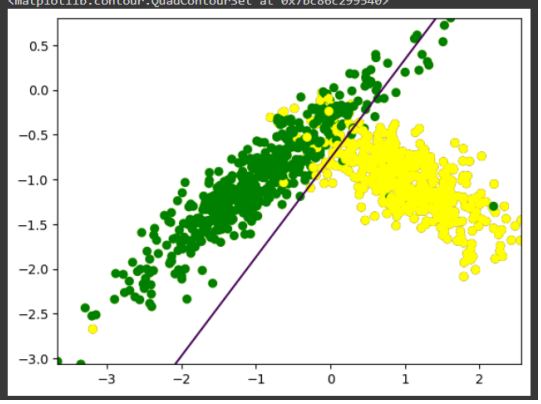
```
#SGDClassifier
model3 = SGDClassifier(random_state = 73, loss = 'hinge')
model3.fit(X_train, y_train)
model3.predict(X_test), y_test
a3 = model2.score(X_train, y_train)
print("the accuracy of SGDClassifier is: " + str(a3) + '\n')
```

```
the accuracy of LogisticRegression is: 0.928

the accuracy of Perceptron is: 0.9226666666666666

the accuracy of SGDClassifier is: 0.9226666666666666
```

شکل18: طبقه‌بندی SGDClassifier

```
ym = model1.decision_function(Xm)
colors = np.array(['yellow', 'green'])
plt.scatter(X[:, 0], X[:, 1], c = colors[y])
plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels = [0])
```

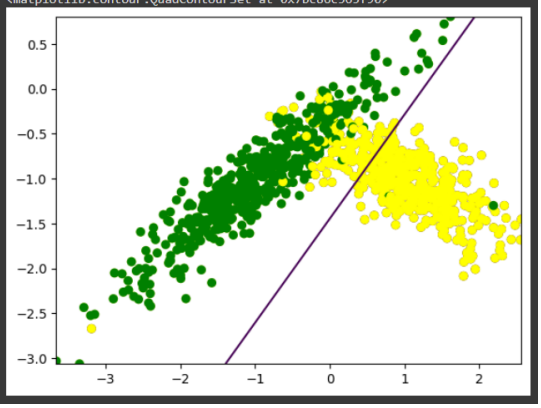
```
the accuracy of LogisticRegression is: 0.9346666666666666

the accuracy of Perceptron is: 0.9106666666666666

the accuracy of SGDClassifier is: 0.9106666666666666

<matplotlib.contour.QuadContourSet at 0x7bc86c299540>
```



شکل19: مرز و نواحی تصمیم‌گیری طبقه‌بندی LogisticRegression

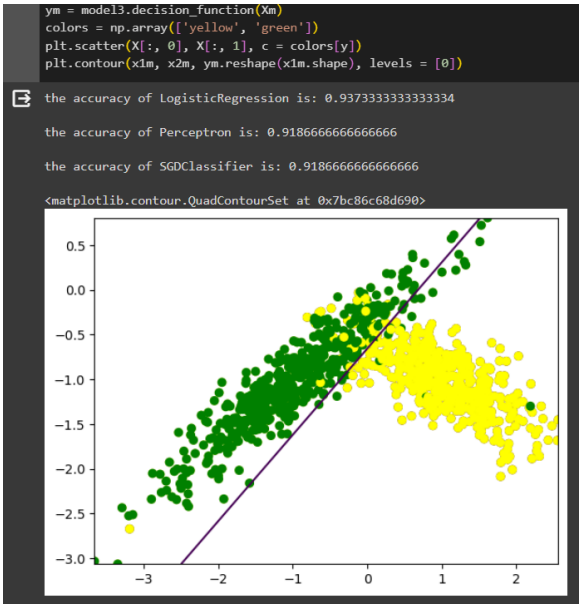
```
ym = model2.decision_function(Xm)
colors = np.array(['yellow', 'green'])
plt.scatter(X[:, 0], X[:, 1], c = colors[y])
plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels = [0])
```

```
the accuracy of LogisticRegression is: 0.9226666666666666

the accuracy of Perceptron is: 0.8773333333333333

the accuracy of SGDClassifier is: 0.8773333333333333

<matplotlib.contour.QuadContourSet at 0x7bc86c565f90>
```



شکل20: مرز و نواحی تصمیم‌گیری طبقه‌بندی Perceptron

همانطور که دیده می‌شود با تغییر پارامتر class_sep از 2.0 به 1.0، دقت طبقه‌بندی کاهش پیدا کرده است و اگر به نمودارها توجه شود، دیده می‌شود که مرز و نواحی تصمیم‌گیری دارای اندکی خطا هستند و نمونه‌های بیشتری نسبت به بخش قبل، دارای طبقه‌بندی نادرست هستند.

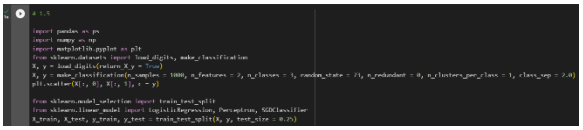


شکل21: مرز و نواحی تصمیم‌گیری طبقه‌بندی SGDClassifier

---

5. اگر یک کلاس به داده‌های تولید شده در قسمت1 اضافه شود، در کدام قسمت‌ها از بلوک دیاگرام آموزش و ارزیابی تغییراتی ایجاد می‌شود؟ در مورد این تغییرات توضیح دهید. آیا می‌توانید در این حالت پیاده‌سازی را به راحتی و با استفاده از کتابخانه‌ها و کدهای آماده پایتونی انجام دهید؟ پیاده‌سازی کنید.

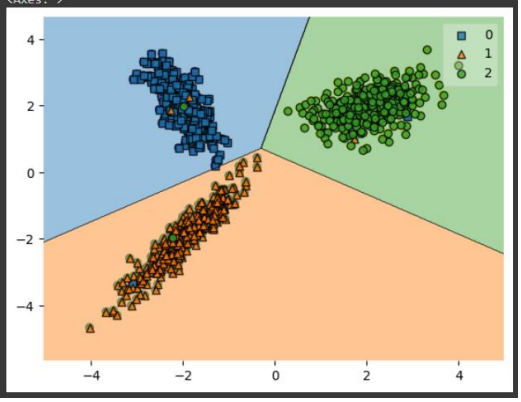
برای اضافه کردن یک کلاس به داده‌ها، در تابع make_classification مقدار پارامتر n_classes را به 3 تغییر می‌دهیم (شکل22).



شکل22: تعریف دیتاست با 3 کلاس

```
# LogisticRegression
model1 = LogisticRegression(random_state = 73)
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + '\n')

from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X, y, clf = model1)
```

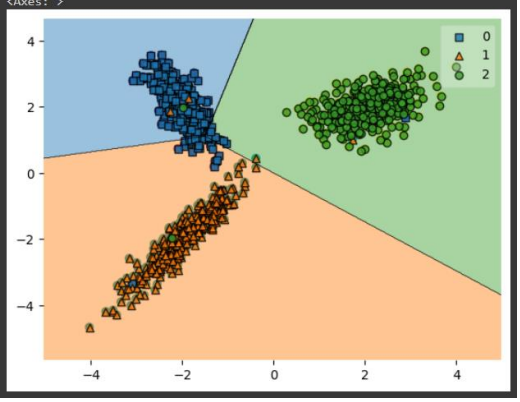
```
the accuracy of LogisticRegression is: 0.9906666666666667

<Axes: >
```



شکل23: طبقه‌بندی LogisticRegression

```
# Perceptron
model2 = Perceptron(random_state = 73, penalty = '12')
model2.fit(X_train, y_train)
model2.predict(X_test), y_test
a2 = model2.score(X_train, y_train)
print("the accuracy of Perceptron is: " + str(a2) + '\n')

from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X, y, clf = model2)
```

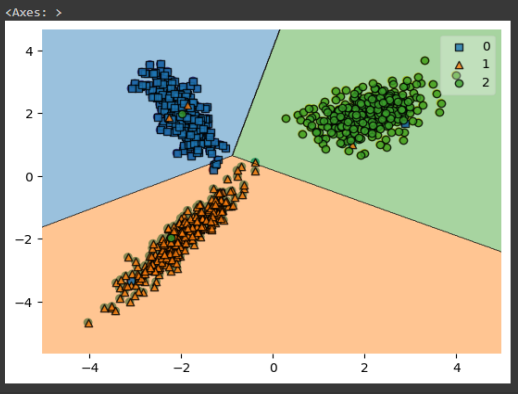
```
the accuracy of LogisticRegression is: 0.988

the accuracy of Perceptron is: 0.9533333333333334

<Axes: >
```



شکل24: طبقه‌بندی Perceptron

```
#SGDClassifier
model3 = SGDClassifier(random_state = 73, loss = 'hinge')
model3.fit(X_train, y_train)
model3.predict(X_test), y_test
a3 = model2.score(X_train, y_train)
print("the accuracy of SGDClassifier is: " + str(a3) + '\n')

from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X, y, clf = model2)
```

```
the accuracy of LogisticRegression is: 0.9893333333333333

the accuracy of Perceptron is: 0.984

the accuracy of SGDClassifier is: 0.984

<Axes: >
```



شکل25: طبقه‌بندی SGDClassifier

سوال دوم

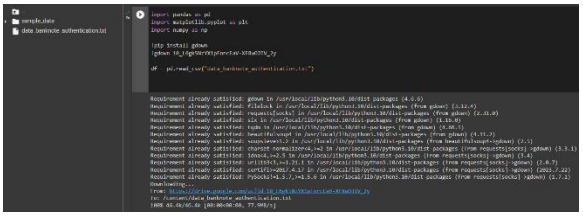
1. با مراجعه به این پیوند با یک دیتاست مربوط به حوزه ((بانکی)) آشنا شوید و ضمن توضیح کوتاه اهداف و ویژگی‌هایش، فایل آن را دانلود کرده و پس از بارگذاری در گوگل درایو خود، آن را با دستور gdown در محیط گوگل کولب قرار دهید. اگر تغییر فرمتی برای فایل این دیتاست نیاز می‌بینید، این کار را با دستورهای پایتونی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

دیتاست مربوط به حوزه بانکی

هر زمان که برای واریز پول به بانک مراجعه می‌شود، صندوقدار اسکناس‌ها را در دستگاهی قرار می‌دهد تا مشخص کند آیا اسکناس‌ها واقعی هستند یا خیر. این مسئله، یک مسئله‌ی طبقه‌بندی است که در آن تعدادی داده ورودی داده می‌شود و باید ورودی را در یکی از چندین دسته از پیش تعیین شده قرار دهیم.

داده‌ها از تصاویری که از نمونه‌های اسکناس اصلی و جعلی گرفته شده‌اند، استخراج می‌شود. به منظور دیجیتالی کردن، از دوربین صنعتی که معمولا برای بازرسی چاپ استفاده می‌شود، استفاده می‌گردد. تصاویر نهایی دارای 400×400 پیکسل هستند.

ابتدا کتابخانه‌های موردنیاز مانند pandas، matplotlib و numpy را تعریف کرده و سپس پکیج gdown را با استفاده از دستور pip install gdown نصب می‌کنیم. سپس آدرس فایلی که در گوگل درایو ذخیره شده است را به کمک دستور gdown وارد کرده و دیتاست را بارگذاری می‌کنیم. برای خواندن دیتاهای درون دیتاست، از دستور pd.read_csv استفاده می‌کنیم (شکل26).



شکل26: بارگذاری دیتاست مربوط به حوزه بانکی

2. ضمن توضیح اهمیت فرایند برزدن (مخلوط کردن)، داده‌ها را مخلوط کرده و با نسبت تقسیم دلخواه و معقول به دو بخش ((آموزش)) و ((ارزیابی)) تقسیم کنید.

با مخلوط کردن داده‌ها اطمینان حاصل می‌شود که مدل در هر دوره در معرض الگوهای مختلف قرار می‌گیرد که به آن کمک می‌کند که ویژگی‌های قوی‌تری را که مرتبط با مشکل موردنظر هستند، بیاموزد. برزدن داده‌ها به کاهش سوگیری که ممکن است به دلیل الگوها یا نظم‌های خاص در مجموعه داده ایجاد شود، کمک می‌کند. این موضوع تضمین می‌کند که مدل هیچ دنباله خاصی را در داده‌ها یاد نمی‌گیرد یا ترجیح نمی‌دهد.

به منظور برزدن داده‌های ذخیره شده در دیتافریم، از دستور df.sample استفاده می‌کنیم و مقدار پارامتر frac را برابر 1 و مقدار پارامتر random_state را برابر 73 انتخاب می‌کنیم. پارامتر frac تعیین می‌کند که چه درصدی از سطرها برای برزدن انتخاب شود و هنگامی که مقدار آن را 1 انتخاب می‌کنیم، یعنی تمام سطرها بر زده شوند. همچنین عبارت reset_index(drop = True) اندیس داده‌ها را پس از برخوردن، مجددا مقداردهی می‌کند.

اگر دیتاست را نمایش دهیم (شکل27) خواهیم دید که ماتریسی با پنج ستون داریم که ستون‌های 1 تا 4 هر سطر، مربوط به ویژگی‌ها و ستون آخر مربوط به تارگت است. بنابراین باید این ستون‌ها را به طور جداگانه در X و y قرار دهیم.

سپس مانند سوال قبل، از کتابخانه sklearn.model_selection تابع test_train_split را فراخوانی و با نسبت 0.3، داده‌ها را به دو بخش آموزش و ارزیابی تقسیم می‌کنیم (شکل28).

```
        3.6216   8.6661  -2.8073  -0.44699  0
0    -0.24037  -1.7837   2.1350   1.24180  1
1    -4.24400 -13.0634  17.1116  -2.80170  1
2    -0.28015   3.0729  -3.3857  -2.91550  1
3     2.97420   8.9600  -2.9024  -1.03790  0
4    -2.91380  -9.4711   9.7668  -0.60216  1
...       ...      ...      ...       ... ..
1366 -1.73440   2.0175   7.7618   0.93532  0
1367  3.52570   1.2829   1.9276   1.79910  0
1368  1.26160   4.4303  -1.3335  -1.75170  0
1369  1.74960  -0.1759   5.1827   1.29220  0
1370  0.89512   4.7738  -4.8431  -5.59090  1

[1371 rows x 5 columns]
```

شکل27: نمایش دیتاست

```
#2.2

from sklearn.model_selection import train_test_split
df_ = df.sample(frac = 1, random_state = 73).reset_index(drop = True)
X = df_.values[:, : -1]
y = df_.values[:, -1]
y = y.reshape((-1, 1))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

شکل28: تقسیم دیتاست به دو بخش آموزش و تست

3. بدون استفاده از کتابخانه‌های آماده پایتون، مدل، تابع اتلاف و الگوریتم یادگیری و ارزیابی را کدنویسی کنید تا دو کلاس موجود در دیتاست به خوبی از یکدیگر تفکیک شوند. نمودار تابع اتلاف را رسم کنید و نتیجه دقت ارزیابی روی داده‌های تست را محاسبه کنید. نمودار تابع اتلاف را تحلیل کنید. آیا می‌توان از روی نمودار تابع اتلاف و قبل از مرحله ارزیابی با قطعیت در مورد عملکرد مدل نظر داد؟ چرا و اگر نمی‌توان، راه‌حل چیست؟

برای پیاده‌سازی مدل، تابعی به نام predict تعریف می‌کنیم که مقدار دیتا، مقدار بایاس و وزن هر دیتا را گرفته و مقدار پیش‌بینی شده‌ی y_hat را تحویل می‌دهد. از آنجایی که مقدار تارگت 0 یا 1 است، تابع sigmoid را انتخاب کرده و ضابطه آن را در پایتون پیاده سازی می‌کنیم (شکل29).

در قدم بعد، تابع اتلاف Binary Cross Entropy را پیاده‌سازی می‌کنیم (شکل30).

در قدم بعد، باید توابع گرادیان و گرادیان نزولی را تعریف کرد (شکل31). در تابع grads با دستور x.T ماتریس داده‌ها ترانهاده شده و با دستور @ ضرب داخلی انجام می‌شود. هدف از تعریف تابع گرادیان نزولی نیز آپدیت کردن مقدار w در هر مرحله است. در فرمول تعریف شده برای گرادیان نزولی، eta گام آموزشی است.

در آخرین قدم باید تابعی برای ارزیابی دقت مدل تعریف شود (تابع accuracy). می‌دانیم که مقدار y_hat پس از تمام عملیاتی که روی آن انجام می‌گیرد ممکن است عددی اعشاری باشد؛ بنابراین با استفاده از تابع np.round(y_hat) آن را به نزدیک‌ترین عدد صحیح گرد می‌کنیم. سپس مقدار گرد شده y_hat را با مقدار y مقایسه کرده و در صورت برابری، نتیجه را با استفاده از تابع np.sum جمع می‌کنیم و در آخر بر اندازه y تقسیم می‌کنیم. به این ترتیب تابع accuracy پیاده‌سازی می‌شود (شکل32).

```python
def predict(x, w):
  y_ = x @ w
  y_hat = 1 / (1 + np.exp(-(y_)))
  return y_hat
```

شکل29: تابع predict

```python
def BCE(y, y_hat):
  loss = -(np.mean(y*np.log(y_hat) + (1 - y)*np.log(1 - y_hat)))
  return loss
```

شکل30: تابع اتلاف BCE

```python
def gradient(x, y, y_hat):
  g = (x.T @ (y_hat - y)) / len(y)
  return g

def gradient_descent(w, eta, g):
  w -= eta*g
  return w
```

شکل31: توابع گرادیان و گرادیان نزولی

```python
def accuracy(y, y_hat):
  a = np.sum(y == np.round(y_hat)) / len(y)
  return a
```

شکل32: تابع accuracy

اینک زمان پیاده‌سازی بخش‌های آموزش و تست است.

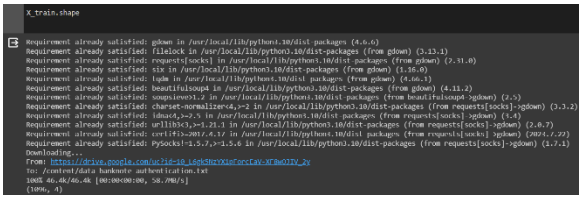
بخش آموزش:

برای تولید ماتریس w، ابتدا لازم است بدانیم که ابعاد ماتریس دیتاست مربوط به آموزش چند در چند است (شکل33). همانطور که دیده می‌شود، ماتریس X_train، 4×1096 است. پس ابعاد ماتریس w باید 1×4 باشد. با استفاده از تابع np.random.randn(4, 1) ماتریسی با مقادیر تصادفی با ابعاد گفته شده برای w تولید می‌کنیم.

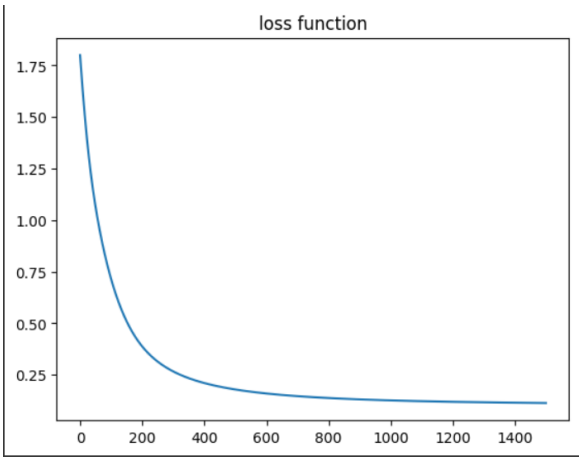
گام آموزشی eta را به دلخواه 0.01 و مقدار epock را 1500 انتخاب می‌کنیم. سپس در هر epock مقدار y_hat، مقدار تابع اتلاف، گرادیان، گرادیان نزولی و مقدار آپدیت شده w را محاسبه می‌کنیم.

همچنین برای رسم تابع اتلاف، آرایه‌ای به نام error_hist[] تعریف کرده و مقدار خطای میان y و y_hat را در هر epock درون آن قرار می‌دهیم.

نمودار تابع اتلاف در شکل34 رسم شده است.



شکل33: ابعاد ماتریس X_train



شکل34: نمودار تابع اتلاف

بخش تست:

اینک دیتاهای X_test را به همراه w آپدیت شده به تابع predict می‌دهیم تا y_hat آنها محاسبه شود. سپس مقادیر y_test و y_hat را به تابع accuracy می‌دهیم تا دقت را نشان دهد (شکل‌های35 و 36).

```
the accuracy is: 0.9514563106796117
```

شکل36: دقت ارزیابی روی داده‌های تست

```
y_hat2 = predict(X_test, w)
acc = accuracy(y_test, y_hat2)
print("the accuracy is: " + str(acc))
```

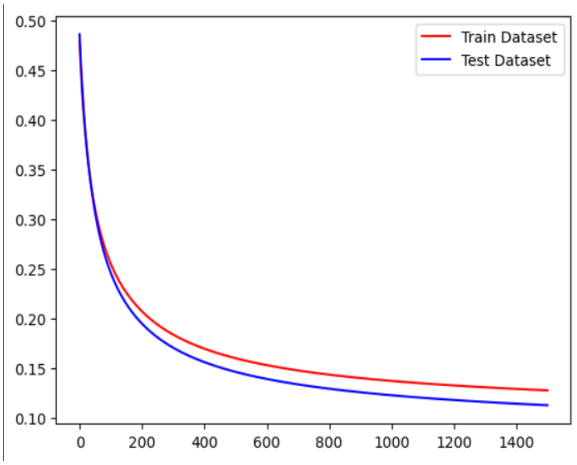
شکل35: بخش تست

همانطور که در تابع اتلاف دیده می‌شود، پس از گذشت چند epock، مدل با سرعت قابل قبولی به سمت یک مقدار بهینه همگرا می‌شود.

البته قبل از مرحله ارزیابی نمی‌توان با قطعیت راجع به عملکرد مدل نظر داد؛ بلکه باید تابع اتلاف داده‌های تست نیز رسم شود و بررسی شود که آیا تابع اتلاف داده‌های آموزش و تست تقریبا به یکدیگر نزدیک هستند یا خیر. بدین منظور، داده‌های تست را نیز از epockها می‌گذرانیم (شکل37) و تابع اتلاف آن را رسم می‌کنیم (شکل38). شکل38 نشان می‌دهد که نمودار اتلاف دیتاست آموزش و تست تقریبا مشابه هم هستند و هر دو همگرا به مقداری مشخص هستند.

```
y_hat2 = predict(X_test, w)
acc = accuracy(y_test, y_hat2)
print("the accuracy is: " + str(acc))

w = np.random.randn(4, 1)
error_hist = []
error_hist_train = []
for i in range(0, epock):
  y_hat = predict(X_train, w)
  e_train = BCE(y_train, y_hat)
  error_hist.append(e_train)
  g = gradient(X_train, y_train, y_hat)
  w = gradient_descent(w, eta, g)
  y_hat2 = predict(X_test, w)
  e_test = BCE(y_test, y_hat2)
  error_hist_train.append(e_test)


plt.plot(error_hist, label = "Train Dataset", color = 'r')
plt.plot(error_hist_train, label = "Test Dataset", color = 'b')
plt.legend()
plt.show()
```

شکل37: گذراندن داده‌های تست از epockها



شکل38: نمودار تابع اتلاف داده‌های Train و Test

4. حداقل دو روش برای نرمال‌سازی داده‌ها را با ذکر اهمیت این فرآیند توضیح دهید و با استفاده از یکی از این روش‌ها، داده‌ها را نرمال کنید. آیا از اطلاعات بخش ((ارزیابی)) در فرآیند نرمال‌سازی استفاده کردید؟ چرا؟

نرمال‌سازی داده‌ها کمک می‌کند تا مطمئن شویم تمام مقادیر در یک بازه‌ی مشخص قرار دارند. نرمال‌سازی داده‌ها می‌تواند دقت تجزیه و تحلیل آماری را با اطمینان از اینکه تمام داده‌ها در یک مقیاس هستند، بهبود ببخشد؛ همچنین امکان مقایسه میان متغیرهای مختلف را فراهم می‌کند. این موضوع تضمین می‌کند که واحدها یا مقیاس‌های اندازه‌گیری بر فرآیند تحلیل تاثیری نمی‌گذارد. نرمال‌سازی می‌تواند به همگرایی سریع‌تر و رفتار پایدار الگوریتم‌هایی مانند گرادیان نزولی کمک کند.

• عادی‌سازی کمینه-بیشینه (Min-Max Normalization)

نرمال‌سازی کمینه-بیشینه داده‌ها را در یک محدوده ثابت، معمولا بین صفر و یک مقیاس می‌کند. فرمول آن به صورت زیر است:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

که در آن X داده اصلی، X_{min} مینیمم مقدار در دیتاست و X_{max} بیشینه مقدار در دیتاست است.

این تکنیک زمانی مفید است که بخواهیم شکل توزیع و مقدار بیشینه و کمینه دیتاست را حفظ کنیم. با این حال، این تکنیک به داده‌های پرت حساس است و اگر دیتاست دارای مقادیر شدید باشد، ممکن است به خوبی کار نکند.

• نرمال‌سازی Z-score

نرمال‌سازی Z-score که استانداردسازی نیز نامیده می‌شود، داده‌ها را به گونه‌ای مقیاس می‌دهد که میانگین 0 و انحراف معیار 1 داشته باشد. فرمول آن به صورت زیر است:

$$X_{norm} = \frac{X - mu}{sigma}$$

که در آن X داده اصلی، mu میانگین داده‌های دیتاست و sigma انحراف معیار داده‌های دیتاست است.

این تکنیک زمانی مفید است که می‌خواهیم نقاط داده را در میان مجموعه داده‌های مختلف مقایسه کنیم یا زمانی که می‌خواهیم نقاط پرت را شناسایی کنیم. با این حال ممکن است شکل توزیع را حفظ نکند و تفسیر مقادیر اصلی را دشوار سازد.

• تبدیل لگاریتمی (Log Transformation)

تبدیل لگاریتمی، داده‌ها را با گرفتن لگاریتم آنها مقیاس می‌کند. فرمول آن به صورت زیر است:

$X_{norm} = \log(X)$

که در آن X داده اصلی است.

این تکنیک زمانی مفید است که داده‌ها مورب هستند و یا زمانی که می‌خواهیم طیف وسیعی از مقادیر را در محدوده کوچکتری فشرده کنیم. با این حال ممکن است این تکنیک زمانی که داده‌ها منفی یا صفر هستند کار نکند.

معمولا توصیه می‌شود که داده‌ها پس از تقسیم شدن به دو بخش آموزش و تست نرمال‌سازی شوند. منطق پشت این توصیه، جلوگیری از نشت اطلاعات از مجموعه تست به مجموعه آموزشی است که می‌تواند منجر به نتایج بیش از حد خوشبینانه و ارزیابی عملکرد غیرواقعی شود.

اگر الگوریتم با داده‌های آموزشی نرمالیزه شده کار می‌کند، باید نرمال‌سازی برای داده‌های تست نیز اعمال شود. همچنین باید توجه شود که دقیقا همان مقیاس‌بندی استفاده شده برای داده‌های آموزشی، بر روی داده‌های تست اعمال شود.

در این بخش از تکنیک نرمال‌سازی Z-score استفاده می‌کنیم. بدین منظور از کتابخانه آماده sklearn.preprocessing تابع StandardScaler را فراخوانی می‌کنیم. سپس یک object به نام scaler تعریف کرده و با استفاده از تابع fit که بر روی داده‌های آموزش اعمال شود، پارامترهای نرمال‌سازی محاسبه شده و بر روی داده‌ها اعمال می‌شود. در نهایت با استفاده از تابع transform، پارامترهای نرمال‌سازی محاسبه شده به داده‌های آموزش و تست منتقل می‌شود (شکل39).

برای بررسی صحت نرمال‌سازی، می‌توان داده‌های آموزش را قبل و بعد از نرمال‌سازی نمایش داد (شکل40).

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

شکل39: نرمال‌سازی داده‌ها به روش StandardScaler

```
Train Dataset befor normalization:
[[ 1.3518e+00  1.0595e+00 -2.3437e+00  3.9998e-01]
 [-1.2852e-03  1.3863e-01 -1.9651e-01  8.1754e-03]
 [ 1.4378e+00  6.6837e-01 -2.0267e+00  1.0271e+00]
 ...
 [ 6.6129e-02  2.4914e+00 -2.9401e+00 -6.2156e-01]
 [-3.8167e+00  5.1401e+00 -6.5063e-01 -5.4306e+00]
 [ 3.0009e+00  5.8126e+00 -2.2306e+00 -6.6553e-01]]


Train Dataset after normalization:
[[ 0.306378   -0.1671621  -0.84980231  0.78567851]
 [-0.16235549 -0.32411117 -0.35317782  0.59882324]
 [ 0.33616998 -0.23382459 -0.77648325  1.08475787]
 ...
 [-0.13900198  0.0768847  -0.98774392  0.29849656]
 [-1.48408505  0.52831758 -0.45821142 -1.9949794 ]
 [ 0.87765644  0.64293555 -0.82364336  0.27752686]]
```

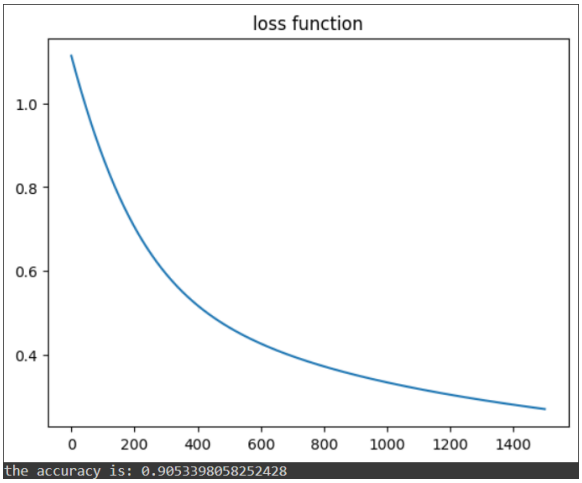
شکل40: نمایش داده‌های آموزش قبل و بعد از نرمال‌سازی

---

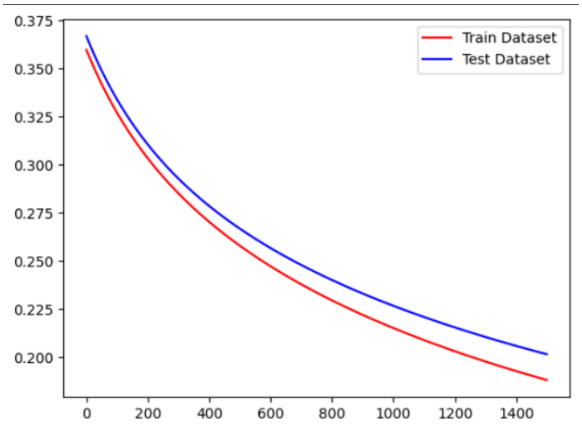
5. تمام قسمت‌های 1 تا 3 را با استفاده از داده‌های نرمال شده تکرار کنید و نتایج پیش‌بینی مدل را برای پنج نمونه داده نشان دهید.

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

!pip install gdown
!gdown 10_L6gk5NzYX1pForcEaV-XF8wOJIV_2y

df = pd.read_csv("data_banknote_authentication.txt")

from sklearn.model_selection import train_test_split
df_ = df.sample(frac = 1, random_state = 73).reset_index(drop = True)
X = df_.values[:, : -1]
y = df_.values[:, -1]
y = y.reshape((-1, 1))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

w = np.random.randn(4, 1)

eta = 0.01
epock = 1500
error_hist = []

for i in range(0, epock):
  y_hat = predict(X_train, w)
  e = BCE(y_train, y_hat)
  error_hist.append(e)
  g = gradient(X_train, y_train, y_hat)
  w = gradient_descent(w, eta, g)

plt.plot(error_hist)
plt.title("loss function")
plt.show()

y_hat2 = predict(X_test, w)
acc = accuracy(y_test, y_hat2)
print("the accuracy is: " + str(acc))
```

شکل41: نرمال‌سازی بعد از تقسیم به دو بخش train و test



شکل42: تابع اتلاف بعد از نرمال‌سازی



شکل43: نمودار تابع اتلاف داده‌های Train و Test

---

6. با استفاده از کدنویسی پایتون وضعیت تعادل داده‌ها در دو کلاس موجود در دیتاست را نشان دهید. آیا تعداد نمونه‌های کلاس‌ها با هم برابر است؟ عدم تعادل در دیتاست می‌تواند منجر به چه مشکلاتی شود؟ برای حل این موضوع چه اقداماتی می‌توان انجام داد؟ پیاده‌سازی کرده و نتیجه را مقایسه و گزارش کنید.

برای یافتن تعداد نمونه‌ها در دو کلاس، کلاس1 را کلاسی با target = 0 و کلاس2 را کلاسی با target = 1 تعریف می‌کنیم. سپس تعداد نمونه‌های هریک را به سادگی با استفاده از for محاسبه کرده (شکل44) و نمایش می‌دهیم (شکل45). همانطور که مشاهده می‌شود تعداد نمونه‌های دو کلاس با هم برابر نیست.

```
#2.6

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

!pip install gdown
!gdown 10_L6gk5NzYX1pForcEaV-XF8wOJIV_2y

df = pd.read_csv("data_banknote_authentication.txt")

from sklearn.model_selection import train_test_split
df_ = df.sample(frac = 1, random_state = 73).reset_index(drop = True)
X = df_.values[:, : -1]
y = df_.values[:, -1]

n_class1 = 0 #target = 0
n_class2 = 0 #target = 1

for i in range(0, len(y)):
  if y[i] == 0:
    n_class1 = n_class1 + 1
  else:
    n_class2 = n_class2 + 1

print("The number of samples in class1 is: " + str(n_class1))
print("The number of samples in class2 is: " + str(n_class2))
```

شکل44: یافتن تعداد نمونه‌های دو کلاس

```
The number of samples in class1 is: 761
The number of samples in class2 is: 610
```

شکل45: نمایش تعداد نمونه‌های دو کلاس

عدم تعادل در یک دیتاست ممکن است باعث بیش برازش (overfitting) شود. یعنی مدل تنها به دنبال تشخیص یک کلاس با تعداد نمونه بیشتر باشد و از کلاس‌های دیگر غافل شود. مجموعه داده‌های نامتعادل ممکن است باعث شود که مدل‌ها دارای یک سوگیری پیش‌بینی کلاس اکثریت باشند. از طرفی مدل‌هایی که از طریق داده‌های نامتعادل آموزش داده شده‌اند، ممکن است به خوبی نتوانند داده‌های تست را پیش‌بینی کنند.

برای حل مشکل عدم تعادل در دیتاست می‌توان اقدامات زیر را انجام داد:

• Random Under-Sampler (RU)

در این روش، از میان نمونه‌های موجود در کلاسی با تعداد نمونه بیشتر، مجددا نمونه‌برداری می‌شود و تعدادی نمونه به طور تصادفی از این کلاس انتخاب می‌شوند. به طور کلی، RU تضمین می‌کند که هیچ داده‌ای به طور مصنوعی تولید نمی‌شود و تمام داده‌های حاصل، زیرمجموعه‌ای از مجموعه داده ورودی اصلی هستند. با این وجود، برای درجات بالای عدم تعادل، این تکنیک معمولا منجر به از دست دادن تعداد زیادی از داده‌های آموزشی می‌شود و در نهایت عملکرد مدل را کاهش می‌دهد.

• Random Over-Sampler (RO)

تکنیک RO مشابه الگوریتم RU است، با این تفاوت که در جهت مخالف حرکت می‌کند؛ به این معنی که در این روش، کلاس‌های با نمونه کمتر، بیشتر نمونه‌برداری می‌شوند تا زمانی که اندازه نمونه کلاس‌ها برابر شود. با نمونه‌برداری بیش از حد، هر نمونه می‌تواند چندین بار در کلاس تکرار شود.

در این سوال، ما از تکنیک RU استفاده می‌کنیم و به طور تصادفی، تعدادی داده از کلاسی با نمونه بیشتر انتخاب می‌کنیم؛ به طوریکه تعداد نمونه‌های هر دو کلاس برابر شوند.

بدین منظور، ویژگی‌ها با تارگت برابر صفر را در کلاس1 و ویژگی‌ها با تارگت برابر یک را در کلاس2 می‌ریزیم. سپس برای کلاس1 که تعداد نمونه‌های بیشتری دارد، عملیات downsampling انجام می‌دهیم. برای این کار از تابع resample در کتابخانه sklearn.utils استفاده کرده و پارامتر n_samples را برابر با تعداد نمونه‌های کلاس2 (610) قرار می‌دهیم. همچنین اگر پارامتر replace در تابع resample، False باشد، نمونه‌هایی با جایگشت‌های

تصادفی انتخاب می‌شوند (مقدار پیش‌فرض این پارامتر True است). بعد از این مرحله، ویژگی‌های دو کلاس و تارگت‌های دو کلاس را با استفاده از دستور np.concatenate به یکدیگر می‌چسبانیم که یک آرایه کلی X و یک آرایه کلی y داشته باشیم (شکل46).

سپس با استفاده از توابع تعریف شده برای کلاس‌بندی، داده‌ها را از یکدیگر تفکیک کرده(شکل‌های47 و 48) و نتیجه را گزارش می‌دهیم (شکل49).

```
X_class1 = df.values[0:761, :-1]
X_class2 = df.values[761: , :-1]
y_class1 = df.values[0:761, -1]
y_class2 = df.values[761: , -1]

from sklearn.utils import resample
X_class1 = resample(X_class1, n_samples = 610, random_state = 73, replace = False)
y_class1 = resample(y_class1, n_samples = 610, random_state = 73, replace = False)

X = np.concatenate((X_class1, X_class2))
y = np.concatenate((y_class1, y_class2))
```

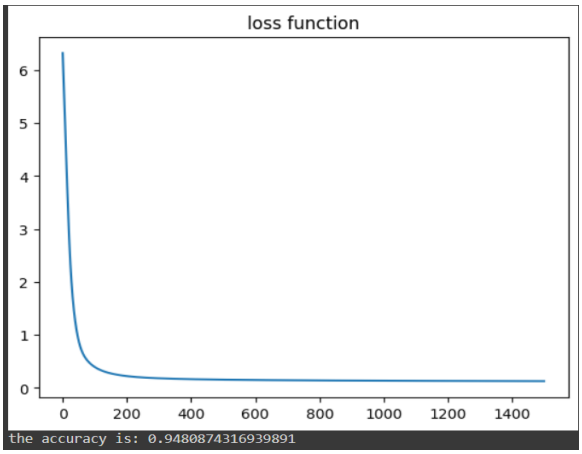
شکل46: انجام عملیات downsampling و ایجاد آرایه X و y جدید

```
from sklearn.model_selection import train_test_split
y = y.reshape((-1, 1))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

w = np.random.randn(4, 1)

eta = 0.01
epock = 1500
error_hist = []

def predict(x, w):
  y_ = x @ w
  y_hat = 1 / (1 + np.exp(-(y_)))
  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):
  g = (x.T @ (y_hat - y)) / len(y)
  return g

def gradient_descent(w, eta, g):
  w -= eta*g
  return w

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

شکل47: کدنویسی تفکیک کلاس‌ها از یکدیگر

```
for i in range(0, epock):
  y_hat = predict(X_train, w)
  e = BCE(y_train, y_hat)
  error_hist.append(e)
  g = gradient(X_train, y_train, y_hat)
  w = gradient_descent(w, eta, g)

plt.plot(error_hist)
plt.title("loss function")
plt.show()

y_hat2 = predict(X_test, w)
acc = accuracy(y_test, y_hat2)
print("the accuracy is: " + str(acc))
```

شکل48: کدنویسی تفکیک کلاس‌ها از یکدیگر - ادامه



شکل49: نتیجه ارزیابی و دقت کلاس‌بندی پس از downsampling و متعادل کردن کلاس‌ها

همانطور که در شکل49 دیده می‌شود، میزان دقت تقریبا 0.9481 است که اندکی بیشتر از زمانی است که دیتاها imbalanced بودند.

7. فرآیند آموزش و ارزیابی مدل را با استفاده از یک طبقه‌بندی آماده پایتونی انجام داده و اینبار در این حالت چالش عدم تعادل داده‌های کلاس را حل کنید.

در این سوال از طبقه‌بندی آماده LogisticRegression استفاده می‌کنیم. توضیح کدهای نوشته شده دقیقا مشابه سوال اول است (شکل50). دقت این طبقه‌بندی در شکل51 نشان داده شده است که این دقت، بسیار بالا است!

```
the accuracy of LogisticRegression is: 0.9916579770594369
```

شکل51: دقت طبقه‌بندی LogisticRegression در حالت کلاس‌های نامتعادل

```
#2.7

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

!pip install gdown
!gdown 10_L6gk5NzYX1pForcEaV-XF8wOJIV_2y

df = pd.read_csv("data_banknote_authentication.txt")
X = df.values[:, : -1]
y = df.values[:, -1]

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

# LogisticRegression
model1 = LogisticRegression(random_state = 73)
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + '\n')
```

شکل50: استفاده از طبقه‌بندی آماده LogisticRegression در حالت کلاس‌های نامتعادل

برای متعادل کردن داده‌ها در حالت استفاده از دستور آماده طبقه‌بندی LogisticRegression، باید مقدار پارامتر class_weight (که مقدار پیش فرض آن ‘none’ است) را به مقدار ‘balance’ تغییر دهیم. در این حالت وزن کلاس‌ها به صورت خودکار تنظیم می‌شود (شکل52). دقت این روش در شکل53 نشان داده شده است که کمی بالاتر از حالتی است که دیتاها imbalanced هستند.

```
the accuracy of LogisticRegression is: 0.9927007299270073
```

شکل53: دقت طبقه‌بندی LogisticRegression در حالت کلاس‌های متعادل

```
model1 = LogisticRegression(random_state = 73, class_weight = 'balanced')
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + '\n')
```

شکل52: متعادل کردن کلاس‌ها و استفاده از طبقه‌بندی LogisticRegression

سوال سوم

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

دیتاست Heart_Diseases_Indicators شامل شاخص‌های مختلف مرتبط با سلامت برای تعدادی از افراد است. این مجموعه داده شامل انواع اطلاعات مرتبط با سلامتی، عوامل سبک زندگی و اطلاعات جمعیتی برای گروهی از افراد است که با استفاده از آن می‌توان عوامل خطر بالقوه برای بیماری قلبی و شرایط سلامتی را بررسی کرد. در اینجا توضیحی از هر ستون آورده شده است:

HeartDiseaseorAttack: نشان می‌دهد که آیا فرد دچار حمله قلبی شده است یا خیر (بله = 1، خیر = 0).

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

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

CholCheck: دفعات بررسی کلسترول (طبقه‌ای).

BMI: شاخص توده‌ای بدن (مقداری پیوسته).

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

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

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

PhysActivity: سطح فعالیت بدنی (طبقه‌ای).

Friuts: فراوانی مصرف میوه (طبقه‌ای).

Veggies: فراوانی مصرف سبزیجات (طبقه‌ای).

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

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

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

GenHlth: ارزیابی سلامت عمومی (طبقه‌ای).

MentHlth: ارزیابی سلامت روان (طبقه‌ای).

PhysHlth: ارزیابی سلامت جسمانی (طبقه‌ای).

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

Sex: جنسیت فرد (مرد = 1، زن = 0).

Age: سن فرد (مقداری پیوسته).

Education: مقطع تحصیلی (طبقه‌ای).

Income: سطح درآمد (طبقه‌ای).

همانطور که در بخش اول سوال دوم هم توضیح داده شد، مطابق شکل54 می‌توان دیتاست را در محیط گوگل کولب بارگذاری کرد.

```
#3.1

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

!pip install gdown
!gdown 1lGxWvBHuLZoujt02neNMuJ2bZ8pyOG9t

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

شکل54: بارگذاری دیتاست بیماری قلبی

2. ضمن توجه به محل قرارگیری هدف و ویژگی‌ها، دیتاست را به صورت یک دیتافریم درآورده و با استفاده از دستورات پایتونی، 100 نمونه داده مربوط به کلاس1 و 100 نمونه داده مربوط به کلاس0 را در یک دیتافریم جدید قرار دهید و در قسمت‌های بعدی با این دیتافریم جدید کار کنید.

برای تفکیک کلاس‌ها از یکدیگر، ابتدا همه داده‌هایی که تارگت (HeartDiseaseorAttack) آنها 1 است را با استفاده از دستور df[df["HeartDiseaseorAttack"] == 1] در کلاس1 و همه داده‌هایی که تارگت آنها 0 است را با استفاده از دستور df[df["HeartDiseaseorAttack"] == 0] در کلاس0 قرار می‌دهیم. سپس با استفاده از دستور df.sample(n = 100) برای هر کلاس، صد نمونه از هر کلاس جدا کرده و با استفاده از دستور reset_index(drop = True) اندیس‌های نمونه‌ها را شماره‌گذاری مجدد و مرتب می‌کنیم. اینک دو کلاس مجزا از هم را به کمک دستور append به هم می‌چسبانیم و در دیتافریم جدید df_new قرار می‌دهیم. در مرحله آخر دیتاهای موجود در df_new را بر می‌زنیم (شکل55).

```
#3.1

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

!pip install gdown
!gdown 1lGxWVBHuLZoujt02neNMuJ2bZ8pyOG9t

df = pd.read_csv("heart_disease_health_indicators.csv")

df_class1 = df[df["HeartDiseaseorAttack"] == 1]
df_class0 = df[df["HeartDiseaseorAttack"] == 0]
df_class1 = df_class1.sample(n = 100, random_state = 73).reset_index(drop = True)
df_class0 = df_class0.sample(n = 100, random_state = 73).reset_index(drop = True)
df_new = df_class1.append(df_class0).reset_index(drop = True)
df_new = df_new.sample(frac = 1, random_state = 73).reset_index(drop = True)
```

شکل55: کلاس‌بندی و ساختن دیتافریم

---

3. با استفاده از حداقل دو طبقه‌بندی 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.

مانند سوال اول، سه طبقه‌بندی آماده LogisticRegression، Perceptron و SGDclassifier را از کتابخانه sklearn.linear_model فراخوانی می‌کنیم. سپس ویژگی‌ها را در X و تارگت‌ها را در y قرار داده و با استفاده از دستور train_test_split از کتابخانه sklearn.model_selection، دیتاست را به دو بخش آموزش و ارزیابی با test_size = 0.3 تقسیم می‌کنیم (شکل56).

اولین طبقه‌بندی LogisticRegression (شکل57)، دومین طبقه‌بندی Perceptron (شکل58) و سومین طبقه‌بندی SGDclassifier (شکل59) است که نتیجه دقت آنها در شکل60 نشان داده شده است (توضیحات کدنویسی مشابه سوال اول است).

```
#3.3

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, Perceptron, SGDClassifier
X = df_new.values[:, 1:]
y = df_new.values[:, 0]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

شکل56: تقسیم دیتاست به دو بخش Train و Test

```
# LogisticRegression
model1 = LogisticRegression(random_state = 73)
model1.fit(X_train, y_train)
model1.predict(X_test), y_test
a1 = model1.score(X_train, y_train)
print("the accuracy of LogisticRegression is: " + str(a1) + '\n')
```

شکل57: طبقه‌بندی LogisticRegression

```
# Perceptron
model2 = Perceptron(random_state = 73, penalty = 'l2')
model2.fit(X_train, y_train)
model2.predict(X_test), y_test
a2 = model2.score(X_train, y_train)
print("the accuracy of Perceptron is: " + str(a2) + '\n')
```

شکل58: طبقه‌بندی Perceptron

```
#SGDClassifier
model3 = SGDClassifier(random_state = 73, loss = 'hinge')
model3.fit(X_train, y_train)
model3.predict(X_test), y_test
a3 = model2.score(X_train, y_train)
print("the accuracy of SGDClassifier is: " + str(a3) + '\n')
```

شکل59: طبقه‌بندی SGDclassifier

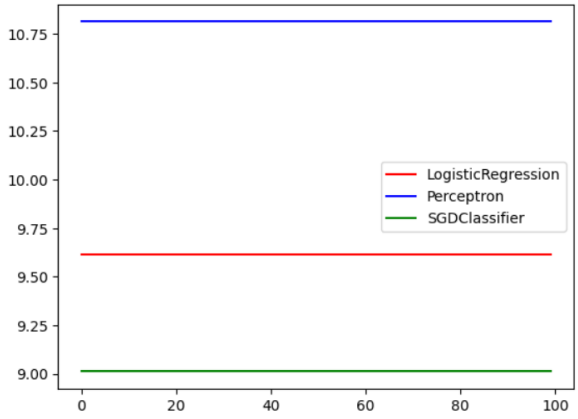
```
the accuracy of LogisticRegression is: 0.85

the accuracy of Perceptron is: 0.7142857142857143

the accuracy of SGDClassifier is: 0.7142857142857143
```

شکل60: نتیجه دقت هر طبقه‌بندی

---

4. در حالت استفاده از دستورات آماده سایکیت‌لرن، آیا راهی برای نمایش تابع اتلاف وجود دارد؟ پیاده‌سازی کنید.

برای نمایش تابع اتلاف، تعداد دوره آموزشی (epock) را 100 انتخاب می‌کنیم و در هر epock مقدار تابع اتلاف را با استفاده از تابع log_loss از کتابخانه sklearn.metrics محاسبه کرده و در آرایه‌ای به نام Lossi (i = 1, 2, 3) می‌ریزیم. در نهایت تابع اتلاف مربوط به هر طبقه‌بندی را نمایش می‌دهیم. کدنویسی این بخش در شکل61 و نمودار توابع اتلاف در شکل62 نشان داده شده است. همانطور که در شکل62 دیده می‌شود، توابع اتلاف به صورت خط هستند؛ این موضوع نشان می‌دهد که تعداد epock در بهبود تابع اتلاف طبقه‌بندی‌هایی که با استفاده از دستورات آماده سایکیت‌لرن انجام شده‌اند، تاثیری ندارد.



شکل62: نمایش توابع اتلاف طبقه‌بندی‌های آماده سایکیت‌لرن

```python
#3.4

from sklearn.metrics import log_loss

loss1 = []
loss2 = []
loss3 = []
epock = 100

for i in range(0, epock):
  model1.fit(X_train, y_train)
  y1 = model1.predict(X_test)
  l1 = log_loss(y_test, y1)
  loss1.append(l1)

for i in range(0, epock):
  model2.fit(X_train, y_train)
  y2 = model2.predict(X_test)
  l2 = log_loss(y_test, y2)
  loss2.append(l2)

for i in range(0, epock):
  model3.fit(X_train, y_train)
  y3 = model3.predict(X_test)
  l3 = log_loss(y_test, y3)
  loss3.append(l3)

plt.plot(loss1, label = "LogisticRegression", color = 'r')
plt.plot(loss2, label = "Perceptron", color = 'b')
plt.plot(loss3, label = "SGDClassifier", color = 'g')
plt.legend()
plt.show()
```

شکل61: کدنویسی برای نمایش توابع اتلاف طبقه‌بندی‌های آماده سایکیت‌لرن

---

5. یک شاخص ارزیابی (غیر از Accuracy) تعریف کنید و بررسی کنید که از چه طریقی می‌توان این شاخص جدید را در ارزیابی داده‌های تست نمایش داد. پیاده‌سازی کنید.

معیار ارزیابی MCC (Matthews correlation coefficient)

معیار MCC بیانگر کیفیت کلاس‌بندی برای یک مجموعه باینری است. MCC سنجه‌ای است که بیانگر بستگی مابین مقادیر مشاهده شده از کلاس باینری و مقادیر پیش‌بینی شده از آن است. مقادیر مورد انتظار برای این کمیت در بازه 1- تا 1 متغیر است. مقدار 1+ نشان‌دهنده پیش‌بینی دقیق و بدون خطای الگوریتم، مقدار 0 نشان‌دهنده پیش‌بینی تصادفی الگوریتم و مقدار 1- نشان‌دهنده عدم تطابق کامل مابین موارد پیش‌بینی شده از کلاس باینری و موارد مشاهده شده است.

برای ارزیابی عملکرد مدل، موارد مثبت واقعی TP، مثبت کاذب FP، منفی واقعی TN و منفی کاذب FN در نظر گرفته می‌شود. فرمول معیار ارزیابی MCC به صورت زیر است:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

این متد ارزیابی را هم می‌توان به صورت دستی در پایتون تعریف کرد و هم می‌توان از دستورات آماده سایکیت‌لرن استفاده کرد که هردو به نتایج یکسانی منجر می‌شوند.

تعریف دستی)

ابتدا ابعاد هر یک از ماتریس‌های y1 (تارگت‌های پیش‌بینی شده با طبقه‌بندی LogisticRegression)، y2 (تارگت‌های پیش‌بینی شده با طبقه‌بندی Perceptron)، y3 (تارگت‌های پیش‌بینی شده با طبقه‌بندی SGDClassifier) و y_test را تغییر می‌دهیم تا دستورات برای نوشتن حلقه for ساده‌تر شود. به منظور تعریف فرمول MCC در پایتون، پارامترهای TP، TN، FP و FN را برای هریک از طبقه‌بندی‌های انجام شده تعریف می‌کنیم. سپس در صورتی که مقدار y_test و مقدار تارگت پیش‌بینی شده هردو برابر یک باشند، یک واحد به TP اضافه، در صورتی که مقدار y_test و مقدار تارگت پیش‌بینی شده هردو برابر صفر باشند، یک واحد به TN اضافه، در صورتی که مقدار y_test یک و مقدار تارگت پیش‌بینی شده صفر باشد، یک واحد به FN اضافه و در صورتی که مقدار y_test صفر و مقدار تارگت پیش‌بینی شده یک باشد، یک واحد به FP اضافه می‌کنیم. در نهایت با داشتن مقادیر این چهار پارامتر، فرمول MCC را تعریف کرده و نتیجه ارزیابی را نمایش می‌دهیم.

مراحل کدنویسی در شکل‌های63 و 64 و نتایج ارزیابی در شکل65 نشان داده شده است.

استفاده از دستورات آماده سایکیت‌لرن)

در این حالت، تابع Matthews_corrcoef از کتابخانه sklearn.metrics را فراخوانی کرده و معیار ارزیابی MCC را برای هر طبقه‌بندی نمایش می‌دهیم (شکل‌های66 و 67).

```
y1 = y1.reshape((-1, 1))
y2 = y2.reshape((-1, 1))
y3 = y3.reshape((-1, 1))
y_test = y_test.reshape((-1, 1))
TP1 = 0
TP2 = 0
TP3 = 0
TN1 = 0
TN2 = 0
TN3 = 0
FP1 = 0
FP2 = 0
FP3 = 0
FN1 = 0
FN2 = 0
FN3 = 0

for i in range(0, len(y_test)):
  if (y_test[i, 0] == 1):
    if (y1[i, 0] == 1):
      TP1 = TP1 + 1
    elif (y1[i, 0] == 0):
      FN1 = FN1 + 1
  elif (y_test[i, 0] == 0):
    if (y1[i, 0] == 1):
      FP1 = FP1 + 1
    elif (y1[i, 0] == 0):
      TN1 = TN1 + 1
```

شکل63: تعریف معیار ارزیابی MCC به صورت دستی

```
for i in range(0, len(y_test)):
  if (y_test[i, 0] == 1):
    if (y2[i, 0] == 1):
      TP2 = TP2 + 1
    elif (y2[i, 0] == 0):
      FN2 = FN2 + 1
  elif (y_test[i, 0] == 0):
    if (y2[i, 0] == 1):
      FP2 = FP2 + 1
    elif (y2[i, 0] == 0):
      TN2 = TN2 + 1

for i in range(0, len(y_test)):
  if (y_test[i, 0] == 1):
    if (y3[i, 0] == 1):
      TP3 = TP3 + 1
    elif (y3[i, 0] == 0):
      FN3 = FN3 + 1
  elif (y_test[i, 0] == 0):
    if (y3[i, 0] == 1):
      FP3 = FP3 + 1
    elif (y3[i, 0] == 0):
      TN3 = TN3 + 1

import math
MCC1 = ((TP1*TN1) - (FP1*FN1))/(math.sqrt((TP1 + FP1)*(TP1 + FN1)*(TN1 + FP1)*(TN1 + FN1)))
MCC2 = ((TP2*TN2) - (FP2*FN2))/(math.sqrt((TP2 + FP2)*(TP2 + FN2)*(TN2 + FP2)*(TN2 + FN2)))
MCC3 = ((TP3*TN3) - (FP3*FN3))/(math.sqrt((TP3 + FP3)*(TP3 + FN3)*(TN3 + FP3)*(TN3 + FN3)))
print("Matthews correlation coefficient of LogisticRegression: " + str(MCC1) + "\t" + "(defining the formula manually)")
print("Matthews correlation coefficient of Perceptron: " + str(MCC2) + "\t" + "(defining the formula manually)")
print("Matthews correlation coefficient of SGDClassifier: " + str(MCC3) + "\t" + "(defining the formula manually)")
```

شکل64: تعریف معیار ارزیابی MCC به صورت دستی - ادامه

```
Matthews correlation coefficient of LogisticRegression: 0.5483270672886724      (defining the formula manually)
Matthews correlation coefficient of Perceptron: 0.539470955255819        (defining the formula manually)
Matthews correlation coefficient of SGDClassifier: 0.5400617248673216    (defining the formula manually)
```

شکل65: نمایش نتایج ارزیابی MCC تعریف شده به صورت دستی

```
from sklearn.metrics import matthews_corrcoef
MCC1_ = matthews_corrcoef(y_test, y1)
MCC2_ = matthews_corrcoef(y_test, y2)
MCC3_ = matthews_corrcoef(y_test, y3)
print("Matthews correlation coefficient of LogisticRegression: " + str(MCC1_) + "\t" + "(using scikit-learn)")
print("Matthews correlation coefficient of Perceptron: " + str(MCC2_) + "\t" + "(using scikit-learn)")
print("Matthews correlation coefficient of SGDClassifier: " + str(MCC3_) + "\t" + "(using scikit-learn)")
```

شکل66: تعریف معیار ارزیابی MCC با استفاده از سایکیت‌لرن

```
Matthews correlation coefficient of LogisticRegression: 0.5483270672886724      (using scikit-learn)
Matthews correlation coefficient of Perceptron: 0.539470955255819        (using scikit-learn)
Matthews correlation coefficient of SGDClassifier: 0.5400617248673216    (using scikit-learn)
```

شکل67: نمایش نتایج ارزیابی MCC تعریف شده با استفاده از سایکیت‌لرن