

دانشگاه صنعتی خواجه نصیرالدین طوسی
دانشکده مهندسی برق - گروه مهندسی کنترل

درس مبانی سیستم‌های هوشمند پاسخ مینی پروژه اول

نام و نام خانوادگی	فریما ایران‌بخش
شماره دانشجویی	۹۸۱۹۸۸۳
تاریخ	آبان ۱۴۰۲



فهرست مطالب

۷	۱ سوال اول
۷	۱.۱ با استفاده از datasets.sklearn یک دیتاست با ۱۰۰۰ نمونه، ۲ کلاس و ۲ ویژگی تولید کنید.
	۲.۱ با استفاده از حداقل دو طبقه بند آماده پایتون و در نظر گرفتن فرآیندهای مناسب، دو کلاس موجود در دیتاست قسمت قبلی را از هم تفکیک کنید. ضمن توضیح روند انتخاب فرآیندها (مانند تعداد دوره آموزش و نرخ یادگیری)، نتیجه دقت آموزش و ارزیابی را نمایش دهید. برای بهبود نتیجه از چه تکنیک‌هایی استفاده کردید؟ .
۸	۳.۱ 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 نمایش دهید.
۹	۴.۱ از چه طریقی می‌توان دیتاست تولیدشده در قسمت «۱» را چالش‌برانگیزتر و سخت‌تر کرد؟ این کار را انجام داده و قسمت‌های «۲» و «۳» را برای این داده‌های جدید تکرار و نتایج را مقایسه کنید.
۱۰	۵.۱ اگر یک کلاس به داده‌های تولیدشده در قسمت «۱» اضافه شود، در کدام قسمت‌ها از بلوک دیاگرام آموزش و ارزیابی تغییری 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.
۱۴	
۱۷	۲ سوال دوم
	۱.۲ 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.
۱۷	۲.۲ ضمن توضیح اهمیت فرآیند بر زدن (مخلوط کردن)، داده‌ها را مخلوط کرده و با نسبت تقسیم دلخواه و معقول به دو بخش «آموزش» و «ارزیابی» تقسیم کنید.
۱۸	۳.۲ 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 عمل کرد مدل نظر داد؟ چرا و اگر نمی‌توان، راه حل چیست؟
۱۹	۴.۲ 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؟
۲۳	۵.۲ تمام قسمت‌های «۱» تا «۳» را با استفاده از داده‌های نرمال شده تکرار کنید و نتایج پیش‌بینی مدل را برای پنج نمونه داده نشان دهید.
۲۴	۶.۲ 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.
۲۷	۷.۲ فرآیند آموزش و ارزیابی مدل را با استفاده از یک طبقه‌بند آماده پایتونی انجام داده و این بار در این حالت چالش عدم تعادل داده‌های کلاس‌ها را حل کنید.
۳۱	
۳۲	۳ سوال سوم



- ۱.۳ 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 دانلودشده دیتاست را روی گوگل درایو خود قرار داده و با استفاده از دستور gdown آن را در محیط گوگل کولب بارگذاری کنید. ۳۲
- ۲.۳ ضمن توجه به محل قرارگیری هدف و ویژگی‌ها، دیتاست را به صورت یک دیتافریم درآورده و با استفاده از دستورات پایتونی، ۱۰۰ نمونه داده مربوط به کلاس «۱» و ۱۰۰ نمونه داده مربوط به کلاس «۰» را در یک دیتافریم جدید قرار دهید و در قسمت‌های بعدی با این دیتافریم جدید کار کنید. ۳۳
- ۳.۳ با استفاده از حداقل دو طبقه بند آماده پایتون و در نظر گرفتن فرایارامترهای مناسب، دو کلاس موجود در دیتاست را از هم تفکیک کنید. نتیجه دقت آموزش و ارزیابی را نمایش دهید. ۳۴
- ۴.۳ در حالت استفاده از دستورات آماده سایکیت لرن، آیا راهی برای نمایش نمودار تابع اتلاف وجود دارد؟ پیاده سازی کنید. ۳۵
- ۵.۳ یک شاخصه ارزیابی غیر از (Accuracy) تعریف کنید و بررسی کنید که از چه طریقی می توان این شاخص جدید را در ارزیابی داده های تست نمایش داد. پیاده سازی کنید. ۳۶



فهرست تصاویر

۸	شکل شماره ۱	۱
۱۱	شکل شماره ۲	۲
۱۲	شکل شماره ۳	۳
۱۳	شکل شماره ۴	۴
۱۵	شکل شماره ۵	۵
۱۶	شکل شماره ۶	۶
۱۸	شکل شماره ۷	۷
۱۸	شکل شماره ۸	۸
۲۱	شکل شماره ۹	۹
۲۲	شکل شماره ۱۰	۱۰
۲۴	شکل شماره ۱۱	۱۱
۲۶	شکل شماره ۱۲	۱۲
۳۰	شکل شماره ۱۳	۱۳
۳۶	شکل شماره ۱۴	۱۴
۳۸	شکل شماره ۱۵	۱۵



فهرست جداول



فهرست برنامه‌ها

۷ (Python) libraries import	۱
۷ (Python) dataset Make	۲
۸ (Python) LogisticRegression	۳
۸ (Python)	۴
۹ (Python) SGDRegression	۵
۹ (Python)	۶
۹ (Python)	۷
۱۰ (Python) test and train	۸
۱۱ (Python)	۹
۱۴ (Python) test and train	۱۰
۱۵ (Python) test and train	۱۱
۱۷ (Python)	۱۲
۱۷ (Python)	۱۳
۱۷ (Python)	۱۴
۱۷ (Python)	۱۵
۱۹ (Python)	۱۶
۱۹ (Python)	۱۷
۲۰ (Python)	۱۸
۲۰ (Python)	۱۹
۲۰ (Python)	۲۰
۲۰ (Python)	۲۱
۲۲ (Python)	۲۲
۲۲ (Python)	۲۳
۲۳ (Python)	۲۴
۲۴ (Python)	۲۵
۲۵ (Python)	۲۶
۲۵ (Python)	۲۷
۲۶ (Python)	۲۸
۲۷ (Python)	۲۹
۲۸ (Python)	۳۰
۲۸ (Python)	۳۱
۲۹ (Python)	۳۲
۲۹ (Python)	۳۳
۳۱ (Python)	۳۴



۳۱	(Python)	۳۵
۳۲	(Python)	۳۶
۳۳	(Python)	۳۷
۳۳	(Python)	۳۸
۳۴	(Python)	۳۹
۳۴	(Python)	۴۰
۳۴	(Python)	۴۱
۳۵	(Python)	۴۲
۳۶	(Python)	۴۳



۱ سوال اول

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

ابتدا کتابخانه‌های مورد نیاز خود را در این قسمت import می‌کنیم:

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import train_test_split
5 from sklearn.linear_model import LogisticRegression, SGDClassifier
6 from sklearn.datasets import make_classification, make_blobs, make_circles
```

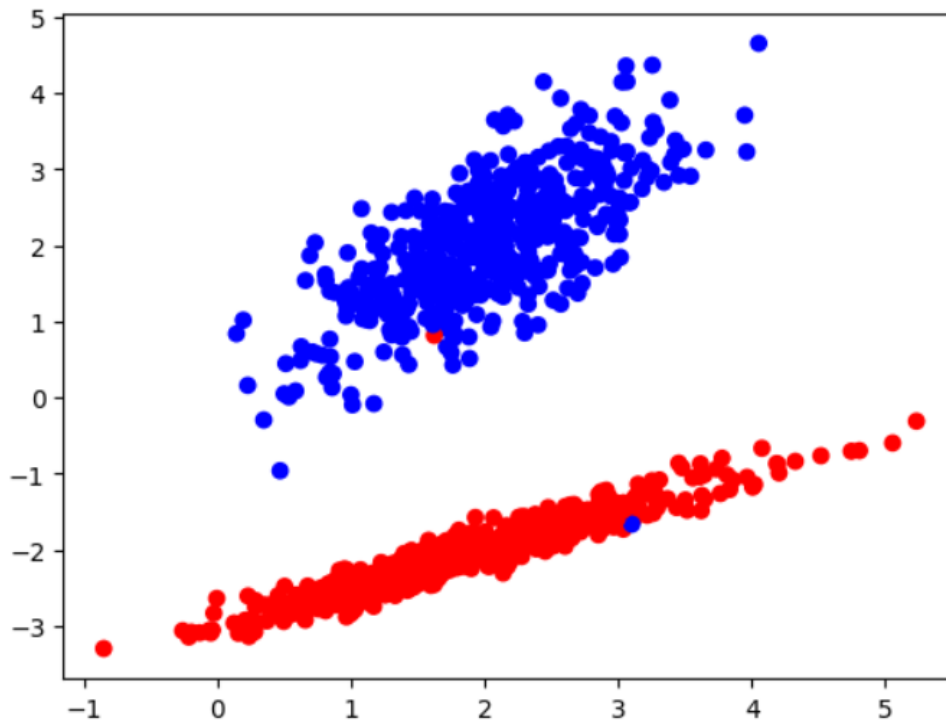
Code 1: import libraries (Python)

سپس با استفاده از تابع آماده در sklearn و با تغییر دادن مقادیر مربوط به آن، دیتاست مورد نیاز آن را تولید می‌کنیم. برای تعداد نمونه‌ها با n-samples و برای تعداد کلاس‌ها با n-classes و برای تعداد ویژگی‌ها با n-features کار می‌کنیم و برابر با مقادیر خواسته شده در صورت سوال، قرارشان می‌دهیم. با توجه به شماره دانشجویی، عبارت random-state را برای گرفتن یک خروجی به ازای هر بار، برابر ۸۳ قرار می‌دهیم. ما ۱۰۰۰ عدد دیتا داریم که در ۲ کلاس طبقه‌بندی شده‌اند و دارای ۲ حالت می‌باشند. class-sep میزان تفکیک داده‌ها را از هم مشخص می‌کند. اگر کمتر از یک باشد، داده‌ها به هم نزدیک‌تر هستند و اگر بیشتر از یک باشد، داده‌ها از هم دورتر می‌شوند. برای راحتی کار در این قسمت سوال آن را برابر ۲ قرار می‌دهیم. n-cluster-per-class تعداد خوشه‌های مربوط به هر کلاس را مشخص می‌کند. وقتی آن را برابر یک قرار می‌دهیم، یعنی هر کلاس یک خوشه جدا برای خود دارد.

```
1 X, y= make_classification(n_samples=1000,
2                           n_features=2,
3                           n_redundant=0,
4                           n_classes=2,
5                           n_clusters_per_class=1,
6                           class_sep=2,
7                           random_state=83)
8
9 colors = np.array(['blue','red'])
10 plt.scatter(X[:,0],X[:,1], c=colors[y])
11 plt.show
```

Code 2: Make dataset (Python)

خروجی آن:



شکل ۱: شکل شماره ۱

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

با استفاده از تابع `train-test-split` که داده‌ها را برای قسمت‌های آموزش و تست با نسبت‌های ۸۰ و ۲۰ جدا می‌کند، داده‌ها را تفکیک می‌کنیم. برای حالت طبقه بندی اول از `LogisticRegression` استفاده می‌کنیم و الگوریتم `optimization` آن را برابر `'sag'` که بر پایه گرادیان نزولی است و `max-iter` که تعداد تکرارها را مشخص می‌کند، برابر ۲۰۰ قرار می‌دهیم. تعداد `max-iter` ها را باید به حدی بالا بگذاریم که الگوریتم همگرا شود، اما نباید آنقدری بالا باشد که محاسبات زیادی و وقت گیر بیشتری انجام دهد.

```
1 x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
2
3 model = LogisticRegression(solver='sag', max_iter=200, random_state=83)
4 model.fit(x_train, y_train)
5 model.predict(x_test), y_test
```

Code 3: LogisticRegression (Python)

نتیجه دقت آموزش و ارزیابی به ترتیب برابر زیر است:

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



```
2 #0.9975
3
4 model.score(x_test, y_test)
5 #0.995
```

Code 4: (Python)

طبقه بند دوم آماده در پایتونی که استفاده کردیم، طبقه بند SGDClassifier است که بر پایه‌ی گرادیان نزولی است. قسمت loss تابع استفاده شده در optimization را نشان می‌دهد که در اینجا از تابع لگاریتمی استفاده شده است.

```
1 model1 = SGDClassifier(loss= 'log_loss', random_state=83)
2 model1.fit(x_train, y_train)
```

Code 5: SGDRegression (Python)

نتیجه دقت آموزش و ارزیابی به ترتیب برابر زیر است:

```
1 model1.score(x_train, y_train)
2 #0.99875
3
4 model1.score(x_test, y_test)
5 #0.995
```

Code 6: (Python)

برای بهبود در نتیجه تعداد آموزش‌ها را زیاد کردیم.

۳.۱ 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 نمایش دهید.

ابتدا مینیمم و ماکزیمم داده‌ها برای هر دو کلاس را مشخص می‌کنیم و خطی با مینیمم و ماکزیمم کلاس اول و کلاس دوم با تعداد ۵۰۰ مقدار می‌کشیم. سپس جایی را که مخالف حدس شبکه ما بود را به عنوان missclassified در نظر می‌گیریم. و می‌گوییم هر جا در شبکه، اگر حدس شبکه با کلاس واقعی آن متفاوت بود علامت ضربدر را نشان دهد و در آخر سه خط با لول‌های یک و صفر و منفی یک برای آن می‌کشیم و شکل را نمایش می‌دهیم:

```
1 x1_min, x2_min = X.min(0)
2 x1_max, x2_max = X.max(0)
3
4 n=500
5 x1r = np.linspace(x1_min, x1_max, n)
6 x2r = np.linspace(x2_min, x2_max, n)
7 x1m , x2m = np.meshgrid(x1r, x2r)
```



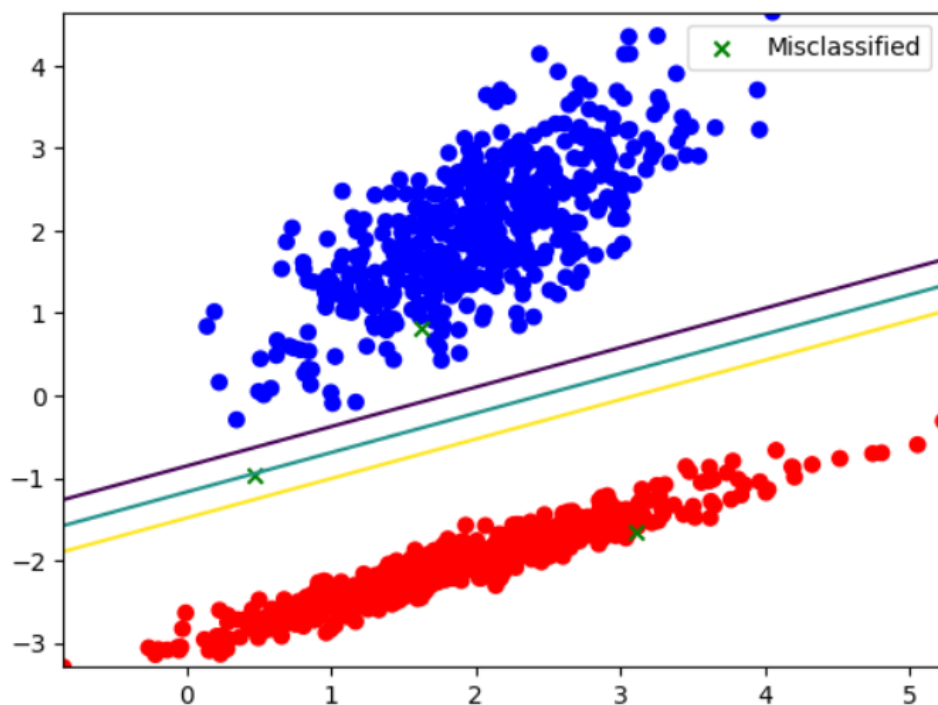
```
8
9 xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
10 ym = model.decision_function(xm)
11
12 predictions = model.predict(X)
13 misclassified_indices = np.where(predictions != y)[0]
14 colors = np.array(['blue', 'red'])
15
16 plt.scatter(X[predictions == y, 0], X[predictions == y, 1], c=colors[y[
    predictions == y]])
17 plt.scatter(X[misclassified_indices, 0], X[misclassified_indices, 1], marker='
    x', c='green', label='Misclassified')
18
19 plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels=[-1, 0, 1])
20
21 plt.legend()
22 plt.show()
```

Code 7: (Python)

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

برای سخت تر کردن کار، می توانیم مقدار class-sep را کمتر کنیم. هر چه مقدار این پارامتر را کمتر کنیم، توهم رفتگی داده ها نیز بیشتر می شود و همچنین می توانیم مقدار n-cluster-per-class را هم برای سخت تر شدن کار، بیشتر کنیم.

```
1 X, y= make_classification(n_samples=1000,
2                             n_features=2,
3                             n_redundant=0,
4                             n_classes=2,
5                             n_clusters_per_class=2,
6                             class_sep=0.8,
7                             random_state=83)
8
9 colors = np.array(['blue', 'red'])
```



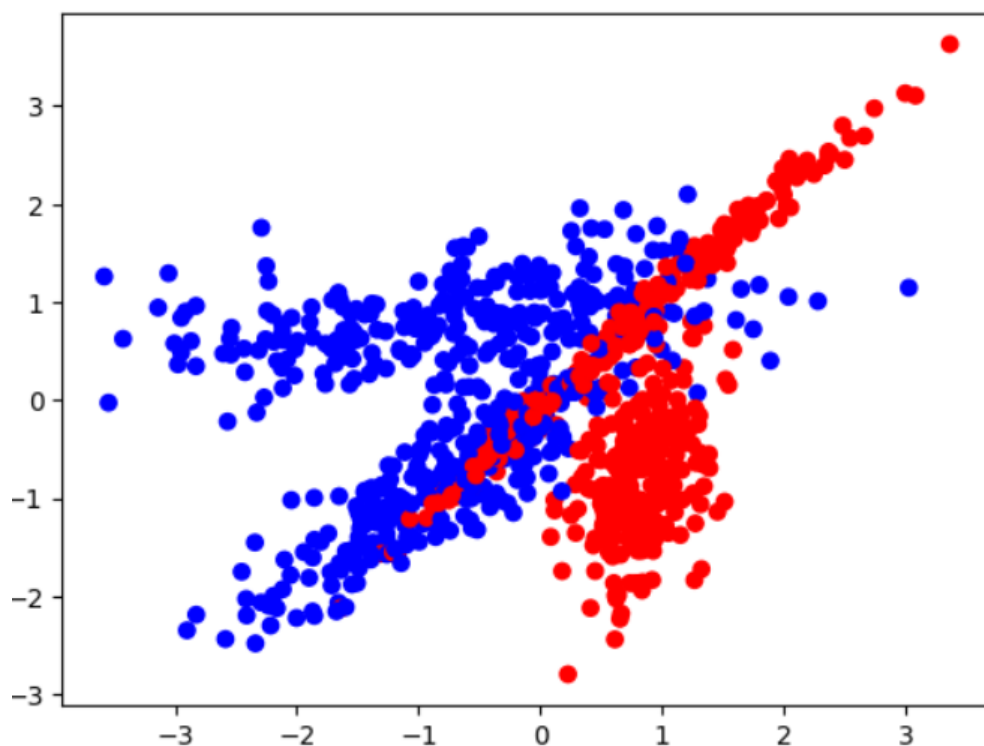
شکل ۲: شکل شماره ۲

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

Code 8: train and test (Python)

خروجی آن به صورت شکل ۳ زیر می‌شود:

```
1 model = LogisticRegression()
2 model.fit(X,y)
3
4 x_train,x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
5
6 model = LogisticRegression(solver='sag', max_iter=200, random_state=27)
7 model.fit(x_train, y_train)
8 model.predict(x_test), y_test
9 model.score(x_train, y_train)
10 #0.86375
11 model.score(x_test, y_test)
12 0.83
13 model1 = SGDClassifier(loss= 'log_loss', random_state=27)
```



شکل ۳: شکل شماره ۳

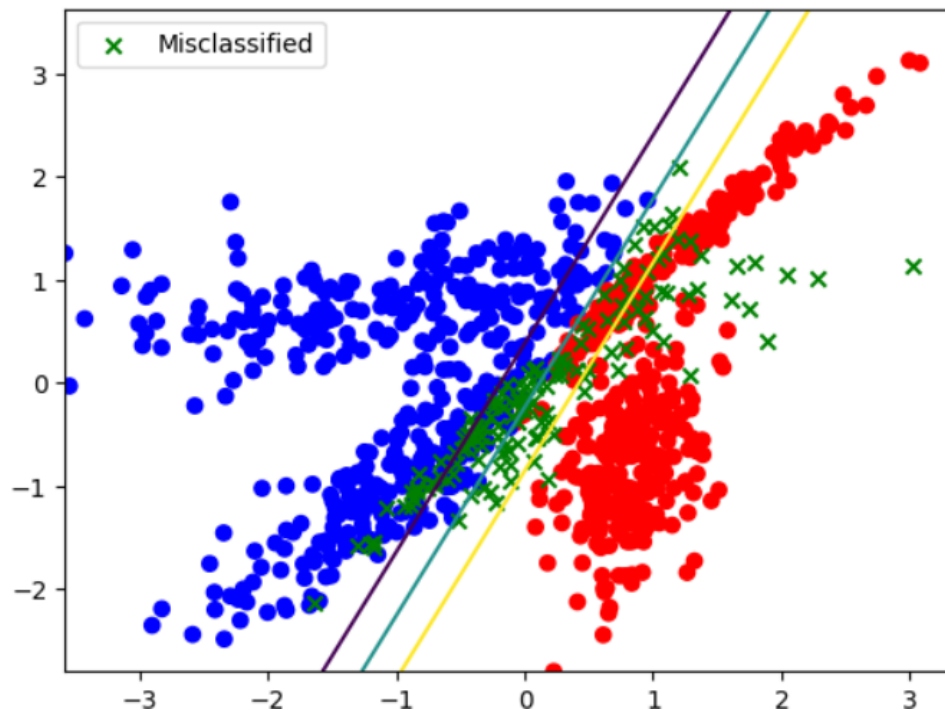
```
14 model1.fit(x_train, y_train)
15 model1.score(x_train, y_train)
16 #0.865
17 model1.score(x_test, y_test)
18 #0.825
19
20
21 x1_min, x2_min = X.min(0)
22 x1_max, x2_max = X.max(0)
23
24 n=500
25 x1r = np.linspace(x1_min, x1_max, n)
26 x2r = np.linspace(x2_min, x2_max, n)
27 x1m , x2m = np.meshgrid(x1r, x2r)
28
29 xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
30 ym = model.decision_function(xm)
```



```
31
32 predictions = model.predict(X)
33 misclassified_indices = np.where(predictions != y)[0]
34 colors = np.array(['blue', 'red'])
35
36 plt.scatter(X[predictions == y, 0], X[predictions == y, 1], c=colors[y[
    predictions == y]])
37 plt.scatter(X[misclassified_indices, 0], X[misclassified_indices, 1], marker='
    x', c='green', label='Misclassified')
38
39 plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels=[-1, 0, 1])
40
41 plt.legend()
42 plt.show()
```

Code 9: (Python)

همانطور که در شکل ۴ نیز مشاهده می‌شود مقدار داده‌هایی که به اشتباه طبقه‌بندی شده‌اند افزایش یافته است و نتایج برای داده‌های



شکل ۴: شکل شماره ۴

آموزش در روش logisticregression و sgd نیز به ترتیب از ۹۹۷۵.۰ به ۸۶۳۷۵.۰ و از ۹۹۵.۰ به ۸۶۵.۰ و برای داده‌های ارزیابی برای روش‌های ذکر شده از ۹۹۸۷۵.۰ به ۸۶۵.۰ و از ۹۹۵.۰ به ۸۲۵.۰ کاهش یافته است و دقت کلاسیفایرها کم شده است. همانطور



که انتظار داشته ایم، با توهم رفتگی داده‌ها امکان جدا سازی و طبقه بندی آن‌ها سخت تر می‌شود و همچنین دقت شبکه عصبی نیز کاهش می‌یابد.

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

اگر با تعداد نمونه‌های یکسان، یک کلاس به داده‌ها اضافه شود مقدار داده‌های هر کلاس کمتر می‌شود و به طبع آن مقدار داده‌های بخش‌های train و test نیز کاهش می‌یابد و یادگیری شبکه عصبی با داده‌های کمتری صورت می‌گیرد و ممکن است نتایج ما به خوبی زمانی که تعداد داده‌ها برای آموزش شبکه عصبی بیشتر بوده‌اند، نباشد. در بلوک دیاگرام ارزیابی با توجه به آن که مقدار ویژگی‌های ما ثابت مانده است و همچنان ۲ ویژگی داریم، اما یک کلاس به آن اضافه شده است، افزودن یک کلاس جدید ممکن است توزیع خطاها و نتایج معیارهای ارزیابی مانند دقت را تغییر دهد. به طور کلی، افزودن یک کلاس جدید ممکن است توزیع داده‌ها، معیارهای ارزیابی و نمودارهای مربوط به ارزیابی مدل را تغییر دهد.

یک کلاس به داده‌ها اضافه می‌کنیم و نتایج به شرح زیر می‌شود:

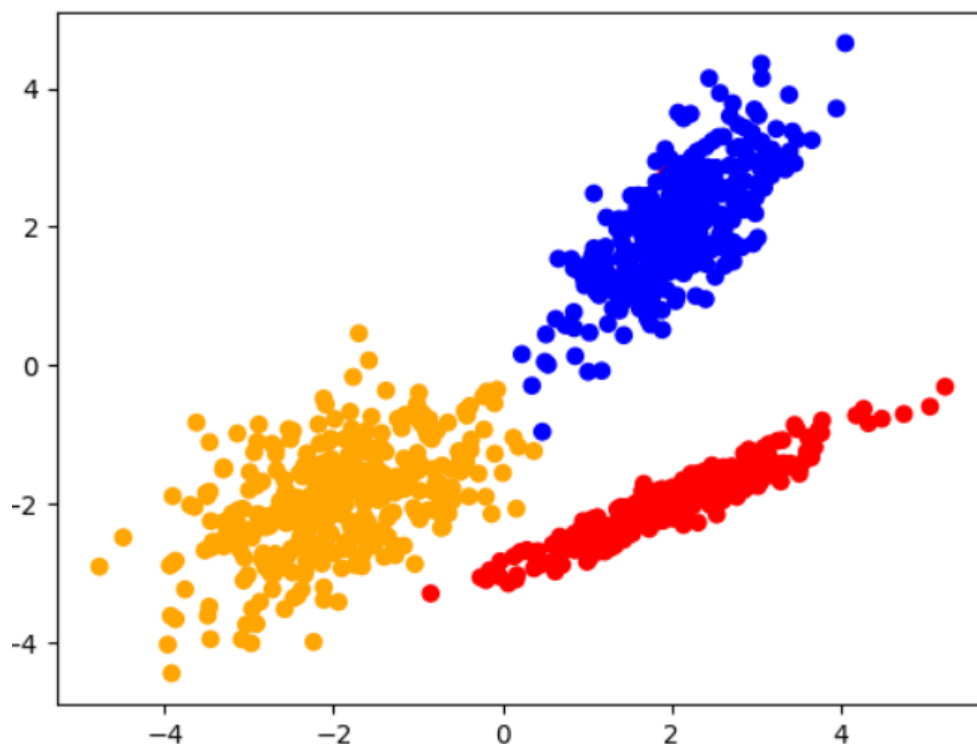
```

1 X, y= make_classification(n_samples=1000,
2                             n_features=2,
3                             n_redundant=0,
4                             n_classes=3,
5                             n_clusters_per_class=1,
6                             class_sep=2,
7                             random_state=83)
8
9 colors = np.array(['blue','red', 'orange'])
10 plt.scatter(X[:,0],X[:,1], c=colors[y])
11 plt.show
12
13 x_train,x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
14
15 model = LogisticRegression(solver='sag', max_iter=200, random_state=27)
16 model.fit(x_train, y_train)

```

Code 10: train and test (Python)

در این حالت پیاده‌سازی سخت‌تر از حالت قبل است. برای پیاده‌سازی طبقه‌بندی، ابتدا مینیمم و ماکزیمم هر ویژگی را محاسبه می‌کنیم و برای ایجاد حاشیه عدد ۱ را از آن کم می‌کنیم. در خط بعدی شبکه‌ای از نقاط را بر اساس محدوده‌های مینیمم و ماکزیمم محاسبه شده، در فضای دو بعدی ایجاد می‌کنیم. سپس مدل با صاف کردن نقاط شبکه و استفاده از مدل آموزش دیده، کلاس را برای هر نقطه در شبکه پیش‌بینی می‌کند. و سپس آن را برای مطابقت داشتن با شکل شبکه، تغییر شکل می‌دهد. و سپس با استفاده از نقاط مش‌گردد و



شکل ۵: شکل شماره ۵

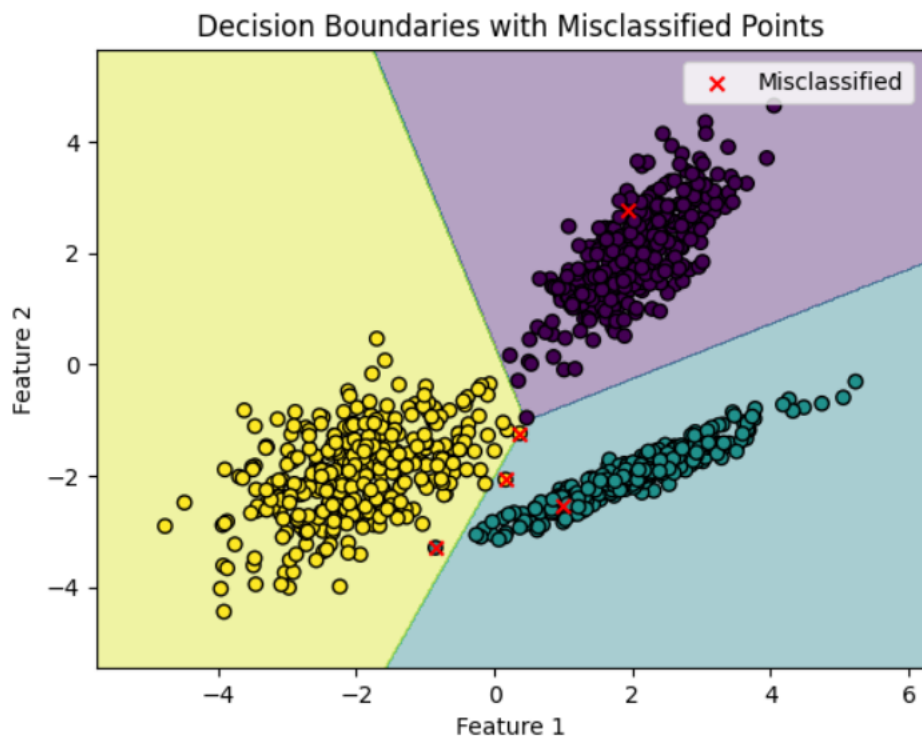
کلاس‌های پیش‌بینی‌شده‌شان، یک نمودار کانتور پر شده ایجاد می‌کنیم که مرزهای تصمیم‌گیری بین کلاس‌های مختلف را نشان می‌دهد. آلفا سطح شفافیت خطوط پر شده را نشان می‌دهد. در آخر نقاط واقعی داده را بر اساس لیبل‌هایشان، رنگی نشان می‌دهیم و داده‌هایی که اشتباه طبقه‌بندی شده‌اند را نیز با رنگ متفاوت نمایش می‌دهیم.

```
1 x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
2 x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
3 xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max, 500),
4                           np.linspace(x2_min, x2_max, 500))
5
6 Z = model.predict(np.c_[xx1.ravel(), xx2.ravel()])
7 Z = Z.reshape(xx1.shape)
8
9 plt.contourf(xx1, xx2, Z, alpha=0.4)
10 plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k')
11
12 predicted_classes = model.predict(X)
13 misclassified_indices = np.where(predicted_classes != y)[0]
14 misclassified_points = X[misclassified_indices]
```




```
15
16 plt.scatter(misclassified_points[:, 0], misclassified_points[:, 1], marker='x'
17             , c='red', label='Misclassified')
18
19 plt.xlabel('Feature 1')
20 plt.ylabel('Feature 2')
21 plt.title('Decision Boundaries with Misclassified Points')
22 plt.legend()
23 plt.show()
```

Code 11: train and test (Python)



شکل ۶: شکل شماره ۶



۲ سوال دوم

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

دیتای موجود در لینک در رابطه با تصاویری است که برای ارزیابی یک روش احراز هویت برای اسکناس‌های بانکی، گرفته شده‌اند. این دیتا دارای ۱۳۷۲ نمونه است و ۵ ستون دارد که مقادیر آن شامل Role, Type, Demographic, Description, Units هست. هر کدام از مقادیر به ترتیب نشان‌دهنده‌ی واریانس عکس تبدیل موجک (یک روش پردازش سیگنال که برای تجزیه یک سیگنال یا تصاویر به قطعات کوچک‌تر مورد استفاده قرار می‌گیرد)، انحراف معیار آن‌ها، کرتوزیس آن‌ها که نشان‌دهنده‌ی میزان مرکزی داده‌هاست که در ارتباط با شیب و انحراف از مرکز است و نشان می‌دهد داده‌ها چقدر متمرکز یا پراکنده در اطراف میانگین مرکزی خود هستند، آنتروپی تصویر که به مفهوم اطلاعات موجود در یک تصویر و تنوع پیکسل‌های آن اشاره دارد و به طور کلی میزان ناهمواری و ناپیوستگی در یک سیستم را اندازه‌گیری می‌کند و در آخر ستون پنجم کلاس آن‌ها را بیان می‌کند. چهار ستون اول شامل اعداد پیوسته و ستون آخر اعداد صحیح را در بردارد. با استفاده از قرار دادن دیتا در گوگل کولب و قرار دادن لینک آن در دستور gdown دیتا را لود می‌کنیم:

```
1 !pip install --upgrade --no-cache-dir gdown
2 !gdown 1Jc2kXnjGemRVUN_e0k0Ci0G1oTC1R0KF
```

Code 12: (Python)

دیتای خود را با دستور زیر فراخوانی می‌کنیم و چون دیتای ما دارای اسم برای ستون خود نیست، header آن را برابر None می‌گذاریم تا برای ستون آن‌ها در سطر اول عدد گذاری کند و سپس ۵ سطر اول دیتا فریم خود را مشاهده می‌کنیم:

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

Code 13: (Python)



خروجی ما به صورت زیر می‌شود و مشاهده می‌کنیم که نامی برای ستون‌ها وجود ندارد. با دستور زیر دیتا را مرتب کرده و طبق نام ستون‌ها، که در سایت وجود داشت، برای آن‌ها اسم گذاری می‌کنیم:

```
1 columns_name = ['Role', 'Type', 'Demographic', 'Description', 'Units']
2
3 df.columns = columns_name
4 df.head()
```

Code 14: (Python)



ستون‌های ۱ تا ۴ برای ویژگی‌های ما است و به عنوان X انتخاب می‌شوند و ستون ۵ام لیبیل ما و خروجی ما ستو به عنوان y انتخاب می‌شود:

```
1 X = df.iloc[:,0:4]
```

	0	1	2	3	4
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

شکل ۷: شکل شماره ۷

	Role	Type	Demographic	Description	Units
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

شکل ۸: شکل شماره ۸

```
2 y = df.iloc[:, -1]
```

Code 15: (Python)

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

مخلوط کردن داده‌ها می‌تواند منجر به بهبود دقت مدل‌های پیش‌بینی و یادگیری ماشین شود. این فرآیند اغلب منجر به داده‌هایی با تنوع و اطلاعات بیشتری می‌شود که باعث افزایش دقت و قابلیت پیش‌بینی مدل‌های آموزش داده شده می‌شود. ادغام داده‌ها اغلب منجر به افزایش تنوع و کمیت داده‌های موجود می‌شود که این امر می‌تواند منجر به بهتر شدن تحلیل‌ها، کاهش انحرافات غیرمنتظره و افزایش اطمینان در تصمیم‌گیری‌ها شود. مخلوط کردن داده‌ها ممکن است اطلاعات جدید و مفیدی که از یک ترکیب منابع به دست می‌آید را ارائه دهد که ممکن است در انجام تحلیل‌های پیشرفته و تصمیم‌گیری‌های مهم مورد استفاده قرار گیرد.



این کار با کتابخانه آماده sickitlearn قابل انجام است و آن را مانند کد زیر فراخوانی می‌کنیم. نسبت تقسیم داده‌ها را ۲۰٪ در نظر می‌گیریم که یعنی به نسبت ۸۰ درصد داده‌ی آموزش و ۲۰ درصد داده‌ها، داده‌ی ارزیابی می‌شوند و در هر دوره epoch، داده‌ها با این نسبت تبدیل تقسیم می‌شوند. همانطور که مشاهده می‌کنیم، اندازه داده‌های آموزش و ارزیابی به شکل زیر می‌شود:

```
1 from sklearn.model_selection import train_test_split
2
3 x_train,x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
4 x_train.shape, y_train.shape, x_test.shape, y_test.shape
5
6 #((1097, 4), (1097,), (275, 4), (275,))
```

Code 16: (Python)

۳.۲ 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 عمل کرد مدل نظر داد؟ چرا و اگر نمی‌توان، راه حل چیست؟

طبق تابع‌های موجود برای ایجاد \hat{y} با استفاده از تابع sigmoid به شکل زیر عمل می‌کنیم و ابتدا تابع sigmoid را تعریف می‌کنیم و سپس \hat{y} را ایجاد می‌کنیم. با استفاده از \hat{y} تابع logisticregression که ضرب x ها و w هاست را تشکیل می‌دهیم و خروجی آن \hat{y} می‌شود که همان y ای است که ما در شبکه خود ایجاد کرده‌ایم. سپس تابع اتلاف را با $\log\text{-loss}$ معرفی کرده و گرادین آن را محاسبه می‌کنیم و η که همان ضریب یادگیری است را در grads که حاصل ضرب x در اختلاف y اصلی ما با \hat{y} بدست آمده است، ضرب کرده و w را هر سری با آن آپدیت می‌کنیم تا به w درست، برسیم.

```
1 def sigmoid(x):
2     return 1 / (1+np.exp(-x))
3
4 def logistic_regression(x,w):
5     y_hat = sigmoid(x @ w)
6     return y_hat
7
8 def bce(y, y_hat):
9     loss = -(np.mean(y*np.log(y_hat)+(1-y)*np.log(1-y_hat)))
10    return loss
```



```

11
12 def gradient(x, y, y_hat):
13     grads = (x.T @ (y_hat - y)) / len(y)
14     return grads
15
16 def gradient_descent(w, eta, grads):
17     w -= eta*grads
18     return w

```

Code 17: (Python)

برای رسم نمودار تابع اتلاف داده‌های آموزش به شرح زیر عمل می‌کنیم:
 x-train را به آرایه تبدیل می‌کنیم و یک ستون تماماً یک به آن اضافه می‌کنیم تا بایاس را نیز در نظر بگیریم.

```

1 x_train = np.asarray(x_train)
2
3 x_train = np.hstack((np.ones((len(x_train), 1)), x_train))

```

Code 18: (Python)

همچنین y-train را نیز به آرایه تبدیل می‌کنیم تا بتوانیم از آن استفاده کنیم و با استفاده از reshape آن را به آرایه دو بعدی تبدیل می‌کنیم تا بتواند ضرب ماتریسی روی آن صورت گیرد:

```

1 y_train = np.array(y_train)
2 y_train = y_train.reshape(-1,1)

```

Code 19: (Python)

ما ماتریسی رندوم با تعداد سطرهای ویژگی+بایاس ما است (۵) و تعداد ستون آن برابر عدد ۱ می‌شود. ضریب آموزش را ۰.۱ و تعداد دورها را ۲۰۰۰ در نظر می‌گیریم:

```

1 w = np.random.randn(5,1)
2
3 eta = 0.01
4 n_epochs = 2000

```

Code 20: (Python)

تابع اتلاف ما به شکل زیر تعریف می‌شود و به تعداد epochهای w را آپدیت می‌کند تا به مقدار نهایی و نزدیک‌ترین جواب برسیم.
 error-hist خالی ای را تعریف می‌کنیم تا در هر بار که تابع اتلاف را محاسبه کردیم، مقدار آن را ذخیره کنیم. نتایج‌های w آپدیت شده را در هر ۱۰۰ دور نمایش می‌دهیم و به شکل زیر است:

```

1 error_hist = []

```



```

2
3 for epoch in range(n_epochs):
4     y_hat = logistic_regression(x_train, w)
5
6     e = bce(y_train, y_hat)
7     error_hist.append(e)
8
9     grads = gradient(x_train, y_train, y_hat)
10
11     w = gradient_descent(w, eta, grads)
12
13     if (epoch+1) % 100 == 0:
14         print(f'Epoch={epoch}, \t E={e:.4f}, \t w={w.T[0]}')
15
16 plt.plot(error_hist)

```

Code 21: (Python)

```

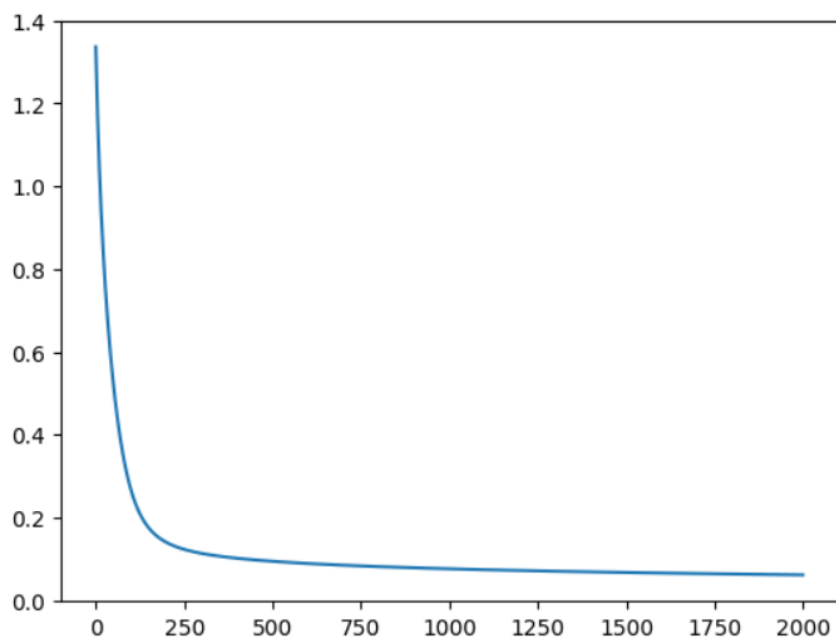
Epoch=99,      E=0.2679,      w=[ 1.11837151 -0.71046428 -0.40289773 -0.12079813 -0.39301001]
Epoch=199,     E=0.1400,      w=[ 1.07647158 -0.81522472 -0.48112187 -0.41577229 -0.2887773 ]
Epoch=299,     E=0.1131,      w=[ 1.07157463 -0.89555113 -0.53703534 -0.53248299 -0.23748836]
Epoch=399,     E=0.1014,      w=[ 1.08004019 -0.96603331 -0.57783351 -0.59698588 -0.21093989]
Epoch=499,     E=0.0940,      w=[ 1.09423111 -1.02797062 -0.60914837 -0.64200144 -0.19612685]
Epoch=599,     E=0.0887,      w=[ 1.1111642  -1.08250866 -0.63491392 -0.67775401 -0.18726061]
Epoch=699,     E=0.0846,      w=[ 1.12948699 -1.13089608 -0.65725507 -0.70822577 -0.18164765]
Epoch=799,     E=0.0812,      w=[ 1.14851275 -1.17423676 -0.67730425 -0.73526954 -0.17794137]
Epoch=899,     E=0.0783,      w=[ 1.16786195 -1.21342956 -0.69569808 -0.75986914 -0.17541785]
Epoch=999,     E=0.0759,      w=[ 1.18731109 -1.24918648 -0.71281917 -0.7826035  -0.17366185]
Epoch=1099,    E=0.0737,      w=[ 1.20672226 -1.28206859 -0.72891307 -0.80384267 -0.17242152]
Epoch=1199,    E=0.0717,      w=[ 1.22600753 -1.31251995 -0.74414705 -0.82383964 -0.17153675]
Epoch=1299,    E=0.0700,      w=[ 1.24510964 -1.34089504 -0.75864123 -0.84277708 -0.17090166]
Epoch=1399,    E=0.0684,      w=[ 1.26399101 -1.36747962 -0.77248582 -0.86079279 -0.17044389]
Epoch=1499,    E=0.0670,      w=[ 1.2826271  -1.39250651 -0.78575114 -0.87799449 -0.17011269]
Epoch=1599,    E=0.0656,      w=[ 1.30100222 -1.41616729 -0.7984938  -0.89446879 -0.16987166]
Epoch=1699,    E=0.0644,      w=[ 1.31910685 -1.43862116 -0.8107605  -0.91028691 -0.16969425]
Epoch=1799,    E=0.0633,      w=[ 1.33693586 -1.46000157 -0.8225906  -0.92550858 -0.16956083]
Epoch=1899,    E=0.0622,      w=[ 1.35448721 -1.48042137 -0.8340178  -0.94018459 -0.16945673]
Epoch=1999,    E=0.0612,      w=[ 1.37176117 -1.49997671 -0.84507143 -0.95435874 -0.16937093]
[<matplotlib.lines.Line2D at 0x7ec3d21a3250>]

```

شکل ۹: شکل شماره ۹

تابع اتلاف نیز به شکل زیر است:

همانطور که در تابع اتلاف مشاهده می‌شود، با هر بار آموزش مقدار اختلاف y -hat که همان خروجی مد نظر ما است از y -train که خروجی شبکه عصبی ما است، کمتر می‌شود و مقدار خطای ما کمتر می‌شود و به حدود صفر می‌رسد و نشان دهنده آن است که آموزش شبکه ما به درستی کار کرده است و رو به بهتر شدن می‌رود. هدف اصلی در این مسئله کمینه کردن مقدار تابع اتلاف است، به طوری که مدل توانایی خود را در پیش‌بینی یا تطبیق با داده‌های ورودی بهینه کند. در طول فرآیند آموزش، مدل بهبود می‌یابد و توانایی پیش‌بینی بهتری را نسبت به داده‌های ورودی پیدا می‌کند. این به معنای این است که تابع اتلاف کاهش می‌یابد و مدل به سمت کمینه کردن خطا یا اختلاف بین خروجی مدل و مقادیر واقعی هدایت می‌شود.



شکل ۱۰: شکل شماره ۱۰

تمام مراجل بالا را برای داده‌های ارزیابی نیز تکرار می‌کنیم با این تفاوت که این بار w ها آپدیت نمی‌شوند و مقدار آخرین w بدست آمده در فرآیند آموزش را به عنوان ورودی برای داده‌های ارزیابی لحاظ می‌کنیم.

```

1 #test
2 x_test = np.asarray(x_test)
3 x_test = np.hstack((np.ones((len(x_test), 1))), x_test))
4
5 y_test = np.array(y_test)
6 y_test = y_test.reshape(-1,1)
7 y_hat = np.array(y_hat)
8 y_hat = y_hat.reshape(-1,1)
9
10 w=[ 1.37176117, -1.49997671, -0.84507143, -0.95435874, -0.16937093]
11 w = np.array(w)
12 w = w.reshape(-1,1)

```

Code 22: (Python)

برای محاسبه دقت ارزیابی از تابع زیر استفاده می‌کنیم:

```

1 def accuracy(y , y_hat):
2     acc = np.sum(y == np.round(y_hat)) / len(y)

```



```
3 return acc
```

Code 23: (Python)

میزان دقت ما برای داده‌های تست به اندازه زیر است:

```
1 y_hat = logistic_regression(x_test,w)
2 accuracy(y_test, y_hat)
3
4 #0.9927272727272727
```

Code 24: (Python)

نمودار تابع اتلاف که در طول زمان آموزش شبکه عصبی رسم می‌شود، اطلاعات مفیدی را ارائه می‌دهد اما تنها از روی آن نمی‌توان به طور کامل و با قطعیت ظر دقیقی در مورد عملکرد نهایی مدل ارائه داد. دلایل آن هم می‌تواند نوسان داشتن نمودار تابع اتلاف در مراحل ابتدایی آموزش که به دلیل فرآیند آموزش و تنظیم پارامترهای مدل است و ممکن است در ادامه بهبود یابند، باشد. و یا یافتن مینیمم محلی و و ناپایدار بودن نمودار تابع اتلاف و همچنین نمودار تابع اتلاف معمولاً فقط نمایانگر عملکرد مدل بر روی داده‌های آموزشی است و اطلاعاتی درباره‌ی عملکرد واقعی مدل بر روی داده‌های جدید یا داده‌هایی که مدل آن‌ها را ندیده است، فراهم نمی‌کند. راه حل آن می‌تواند شامل تقسیم داده‌ها و نسبت دادن داده‌ها به دو بخش آموزش و ارزیابی باشد. همچنین می‌توانیم عملکرد شبکه را ارزیابی کنیم. یعنی کارکرد مدل را بر روی داده‌هایی که مدل آن‌ها را ندیده است. و همچنین می‌توانیم از معیارهای ارزیابی استفاده کنیم مانند Accuracy و ماتریس درهم‌ریختگی مدل. همچنین می‌توانیم از مجموعه اعتبارسنجی یا Validation set استفاده کنیم که داده‌هایی است که جدا از مجموعه تست و آموزش است.

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

نرمال کردن داده‌ها برای منطقی کردن مقایسه پذیری و بهبود عملکرد الگوریتم‌های یادگیری ماشین و تسهیل فرآیند بهینه سازی و کمک به جلوگیری از overfitting صورت می‌گیرد. واحدها و مقیاس‌های مختلف برای ویژگی‌ها مکن است باعث کاهش دقت و عدم قابلیت مقایسه و تفسیر شود. نرمال کردن داده‌ها باعث می‌شود تمام ویژگی‌ها به یک مقیاس یا بازه مشابه تبدیل شوند که قابل مقایسه تر و تفسیرپذیرتر باشند. برای مدل‌های با پارامتر زیاد، اگر داده‌ها نرمال نشوند، احتمال overfitting بیشتر می‌شود؛ زیرا مدل ممکن است به اندازه‌ی زیادی به داده‌های با ویژگی‌های بزرگتر (به دلیل مقیاس بزرگتر) وابستگی پیدا کند و از یادگیری الگوهای کلی دور شود. روش اول: Min-Max Scaling این روش به ما کمک می‌کند تا مقدار داده‌ها بین ۰ و ۱ برود.

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (1)$$



روش دوم: Z-score Normalization این روش با توجه به میانگین و انحراف معیاری داده‌ها، آن‌ها را به صورتی نرمال می‌کند که میانگین آن‌ها صفر و انحراف معیاری یک باشد.

$$X_{normalized} = \frac{X - \mu}{\sigma} \quad (2)$$

برای نرمال کردن داده‌ها از روش اول استفاده می‌کنیم و کد آن به صورت زیر است. چون در دیتافریم داده‌ها دانه به دانه بررسی می‌شود، پی‌نیازی به اندیس‌گذاری نمی‌باشد.

```
1 normalized_df = (df - df.min()) / (df.max() - df.min())
2 normalized_df
```

Code 25: (Python)

داده‌های نرمال شده:

	Role	Type	Demographic	Description	Units
0	0.769004	0.839643	0.106783	0.736628	0.0
1	0.835659	0.820982	0.121804	0.644326	0.0
2	0.786629	0.416648	0.310608	0.786951	0.0
3	0.757105	0.871699	0.054921	0.450440	0.0
4	0.531578	0.348662	0.424662	0.687362	0.0
...
1367	0.537124	0.565855	0.165249	0.726398	1.0
1368	0.407690	0.332868	0.506753	0.808350	1.0
1369	0.237385	0.011768	0.985603	0.524755	1.0
1370	0.250842	0.201701	0.761587	0.660675	1.0
1371	0.324528	0.490747	0.343348	0.885949	1.0

1372 rows × 5 columns

شکل ۱۱: شکل شماره ۱۱

در فرآیند نرمال‌سازی تمام داده‌های تست و آموزش نرمال می‌شوند و تفاوتی بین داده‌ها برای نرمال‌سازی وجود ندارد.

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



```
1 X1 = normalized_df.iloc[:,0:4]
2 y1 = normalized_df.iloc[:,-1]
3
4 x_train1,x_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.2)
5
6 x_train1 = np.asarray(x_train1)
7 x_train1 = np.hstack((np.ones((len(x_train1), 1)), x_train1))
8
9 y_train1 = np.array(y_train1)
10 y_train1 = y_train1.reshape(-1,1)
11
12 w = np.random.randn(5,1)
13 error_hist = []
14
15 for epoch in range(n_epochs):
16     y_hat = logistic_regression(x_train1, w)
17
18     e = bce(y_train1, y_hat)
19     error_hist.append(e)
20
21     grads = gradient(x_train1, y_train1, y_hat)
22
23     w = gradient_descent(w, eta, grads)
24
25     if (epoch+1) % 100 == 0:
26         print(f'Epoch={epoch}, \t E={e:.4f}, \t w={w.T[0]}')
27
28 plt.plot(error_hist)
```

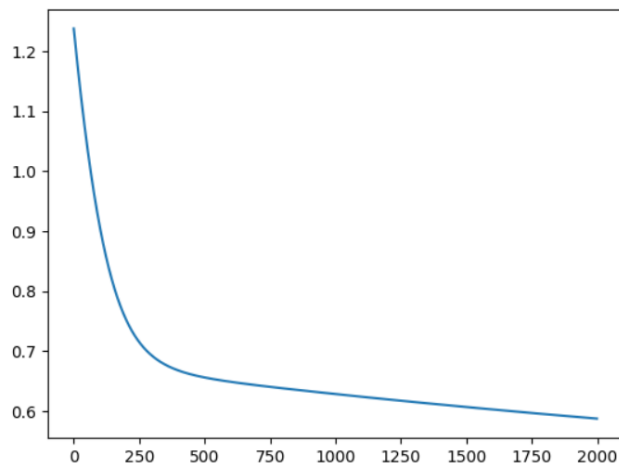
Code 26: (Python)

تابع اتلاف با هر بار دور آموزش کاهش یافته است و شبکه عصبی رو به بهبودی و آموزش بهتر می‌رود اما بعد از نرمال سازی داده‌ها مقدار تابع اتلاف بیشتر شده است و در آخر به مقدار نزدیک ۶.۰ رسیده است که این با نتایج قبل از نرمال سازی داده‌ها متفاوت است. همچنین مقدار ارزیابی دقت داده‌های تست به مقدار ۷۶۷۲.۰ درصد رسیده است که این عدد نیز کمتر از دقت حالت قبل می‌باشد.

```
1 w=[ 0.54361441, -0.92919742, -1.22702091, -0.39217451, 0.69396037]
```



Epoch=1999, E=0.5870, w=[0.54361441 -0.92919742 -1.22702091 -0.39217451 0.69396037]
 [<matplotlib.lines.Line2D at 0x7ec3d20f2290>]



شکل ۱۲: شکل شماره ۱۲

```

2 w = np.array(w)
3 w = w.reshape(-1,1)
4
5 x_test1 = np.asarray(x_test1)
6 x_test1 = np.hstack((np.ones((len(x_test1), 1)), x_test1))
7
8 y_test1 = np.array(y_test1)
9 y_test1 = y_test1.reshape(-1,1)
10 y_hat = np.array(y_hat)
11 y_hat = y_hat.reshape(-1,1)
12
13 y_hat = logistic_regression(x_test1,w)
14 accuracy(y_test1, y_hat)
15
16 #0.7672727272727272

```

Code 27: (Python)

برای نمایش ۵ داده‌ی آن:

```

1 y_hat[:5]
2
3 #array([[0.44783633],

```



```

4      [0.55859103] ,
5      [0.32711303] ,
6      [0.46778033] ,
7      [0.40046827]] )

```

Code 28: (Python)

۶.۲ 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 فهمیدن تعداد داده‌ها برای هر کلاس مختلف، تعداد آن را با روش زیر می‌شماریم و متوجه می‌شویم که تعداد داده‌هایی که کلاس آن‌ها ۱ است با تعداد داده‌هایی که ۹ کلاس آن‌ها ۰ است، یکسان نمی‌باشد.

```

1 a = df[df['Units'] == 1]
2 b = df[df['Units'] == 0]
3
4 print(f'len a:{len(a)}')
5 print(f'len b:{len(b)}')
6
7 #len a:610
8 #len b:762

```

Code 29: (Python)

وجود تعداد ناصحیح نمونه‌ها در هر کلاس می‌تواند منجر به تأثیرات منفی بر عملکرد مدل‌های یادگیری ماشین شود. مدل‌هایی که با داده‌های نامتوازن آموزش داده شده‌اند، ممکن است تمایل به پیش‌بینی کلاس اکثریت داشته باشند و در تشخیص کلاس‌های کمتری دچار مشکل شوند. معیارهای ارزیابی مانند دقت (Accuracy) در مواجهه با دیتاست‌های نامتوازن ممکن است تا حدودی مطلوبیت خود را از دست بدهند. به عنوان مثال، اگر یک کلاس دارای تعداد نمونه کمی باشد و سایر کلاس‌ها دارای تعداد بیشتری نمونه داشته باشند، مدلی که تمام نمونه‌ها را به عنوان عضو اکثریت تشخیص دهد، با دقت بالایی عمل می‌کند که این مورد معمولاً نمایانگر یک عملکرد نامطلوب است. وجود تعداد نامتوازن نمونه‌ها می‌تواند باعث شود که الگوهای کمتر مشاهده شوند و در نتیجه توانایی مدل در تشخیص و یادگیری این الگوها کاهش یابد. این موضوع ممکن است در مسائلی که تشخیص کلاس‌های کمتر مهم است (مانند تشخیص بیماری‌های نادر)، اثر مخربی داشته باشد. همچنین ممکن است تعمیم‌پذیری مدل را کاهش دهند. در صورتی که مدل تنها با داده‌های کلاس اکثریت آموزش ببیند، احتمال بروز overfitting به داده‌های این کلاس بیشتر است و توانایی عمومی‌سازی مدل کاهش می‌یابد. در برخی موارد، اگر داده‌ها نامتوازن باشند، اعتبارپذیری نتایج و استنتاج‌ها ممکن است کاهش یابد. این مسئله می‌تواند وجود داشتن تعداد کمی از یک



کلاس را نادیده گرفته و تحلیل‌های نادرستی را به دنبال داشته باشد. برای حل این موضوع اگر تعداد داده‌های ما زیاد بود، می‌توانیم تعداد داده‌های کلاس بیشتر را کم کنیم تا تعداد یکسانی داشته باشند که البته این روش روش خوبی نیست. روش دیگری برای درست کردن این موضوع، ایجاد داده‌ی فیک است. این کار را در این پروژه با میانگین‌گیری از دو سطر و ایجاد سطر جدید انجام دادیم. دیتا فریم جدید و خالی‌ای به اسم new-row ایجاد می‌کنیم و به تعداد اختلاف a و b، با استفاده از میانگین سطرهای بالایی و پایینی در a داده‌ی جدید ایجاد کرده و به دلیل آن که کلاس a، ۱ است Unit تمام داده‌های تولید شده را برابر ۱ قرار می‌دهیم و در نهایت تمامی داده‌های تولید شده‌ی جدید را در new-row قرار داده و آن را با a ابتدایی، مخلوط می‌کنیم.

```

1 new_rows = pd.DataFrame()
2
3 for i in range(len(b)-len(a)):
4     v= a.iloc[i:i+2, :-1].mean()
5
6     new_row = v.append(pd.Series({'Units': 1}))
7     new_rows = new_rows.append(new_row, ignore_index=True)
8
9
10 a.reset_index(drop=True, inplace=True)
11 new_rows.reset_index(drop=True, inplace=True)
12
13 updated_df = pd.concat([a.reset_index(drop=True), new_rows], ignore_index=True
14                          )
15 updated_df

```

Code 30: (Python)

اکنون تعداد داده‌های کلاس ۱ و ۰ با هم برابر و مقدار ۷۶۲ سطر را دارند. و در نهایت updated-df را نیز با b مخلوط می‌کنیم و در دیتا فریم combined-df می‌ریزیم.

```

1 combined_df = updated_df.append(b)
2
3 combined_df
4 #1524 rows × 5 columns

```

Code 31: (Python)



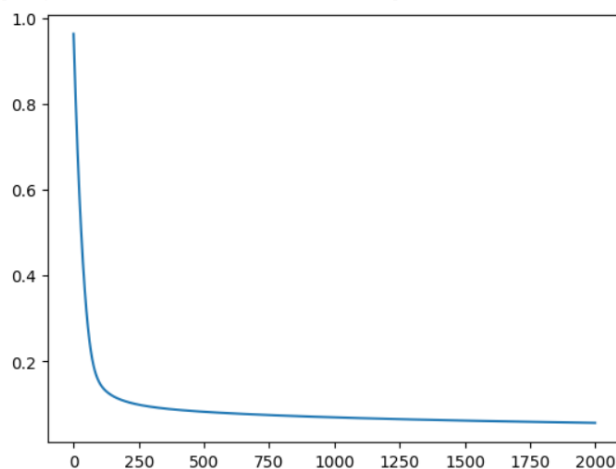
```
1 X2 = combined_df.iloc[:,0:4]
2 y2 = combined_df.iloc[:, -1]
3
4 x_train2, x_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.2)
5
6 x_train2 = np.asarray(x_train2)
7 x_train2 = np.hstack((np.ones((len(x_train2), 1)), x_train2))
8
9 y_train2 = np.array(y_train2)
10 y_train2 = y_train2.reshape(-1,1)
11
12 error_hist = []
13
14 for epoch in range(n_epochs):
15     y_hat = logistic_regression(x_train2, w)
16
17     e = bce(y_train2, y_hat)
18     error_hist.append(e)
19
20     grads = gradient(x_train2, y_train2, y_hat)
21
22     w = gradient_descent(w, eta, grads)
23
24     if (epoch+1) % 100 == 0:
25         print(f'Epoch={epoch}, \t E={e:.4f}, \t w={w.T[0]}')
26
27 plt.plot(error_hist)
```

Code 32: (Python)

```
1 #test
2 x_test2 = np.asarray(x_test2)
3 x_test2 = np.hstack((np.ones((len(x_test2), 1)), x_test2))
4
```



```
Epoch=1999,      E=0.0562,      w=[ 1.3163901 -1.62990052 -0.88214326 -0.99184632 -0.19526835]  
[<matplotlib.lines.Line2D at 0x7ec3cdc44df0>]
```



شکل ۱۳: شکل شماره ۱۳

```
5 y_test2 = np.array(y_test2)
6 y_test2 = y_test2.reshape(-1,1)
7 y_hat = np.array(y_hat)
8 y_hat = y_hat.reshape(-1,1)
9
10 w=[1.3163901, -1.62990052, -0.88214326, -0.99184632, -0.19526835]
11 w = np.array(w)
12 w = w.reshape(-1,1)
13
14 y_hat = logistic_regression(x_test2,w)
15 accuracy(y_test2, y_hat)
16
17 #0.9868852459016394
```

Code 33: (Python)

بعد از برابر کردن تعداد داده‌ها، می‌بینیم که عملکرد شبکه ما تا حد زیادی بهبود یافته است و همچنین مقدار خطای ما از عدد کمتری شروع می‌شود و به عدد کمتری از حالت قبل نیز می‌رسد. در نتیجه مشاهده می‌کنیم که عملکرد شبکه عصبی با یکسان بودن تعداد داده‌های تست و آموزش بهبود می‌یابد.



۷.۲ فرآیند آموزش و ارزیابی مدل را با استفاده از یک طبقه بند آماده پایتونی انجام داده و این بار در این حالت چالش عدم تعادل داده‌های کلاس‌ها را حل کنید.

برای انجام فرآیند آموزش و ارزیابی از طبقه بند آماده‌ی LogisticRegression استفاده می‌کنیم. قبل از متعادل کردن تعداد داده‌ها، دقت ارزیابی شبکه ما ۹۸۱۸ درصد است.

```
1 from sklearn.linear_model import LogisticRegression, SGDClassifier
2
3 x_train3, x_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.2)
4
5 model = LogisticRegression(solver='sag', max_iter=200, random_state=83)
6 model.fit(x_train3, y_train3)
7
8 model.score(x_test3, y_test3)
9
10 #0.9818181818181818
```

Code 34: (Python)

برای متعادل کردن تعداد داده‌ها از کتابخانه sklearn.utils.resample را import می‌کنیم. مانند قسمت بالا، کلاس‌های ۰ و ۱ را از هم جدا می‌کنیم و مینیمم تعداد آن‌ها را در min-class-size می‌ریزیم. بعد از آن کلاسی که تعداد داده‌ی کمتری دارد را افزایش می‌دهیم تا به تعداد کلاس بالاتر برسد. و سپس هر دو کلاس را در balanced-df که یک دیتا فریم جدید است، می‌ریزیم. بعد از آن ستون ویژگی‌ها و تارگت را از هم جدا می‌کنیم.

```
1 class_0 = df[df['Units'] == 0]
2 class_1 = df[df['Units'] == 1]
3
4 min_class_size = min(len(class_0), len(class_1))
5
6 balanced_class_0 = resample(class_0, replace=True, n_samples=min_class_size,
7                             random_state=83)
8 balanced_class_1 = resample(class_1, replace=True, n_samples=min_class_size,
9                             random_state=83)
10
11 balanced_df = pd.concat([balanced_class_0, balanced_class_1])
12
13 features = balanced_df.drop('Units', axis=1)
```




```

12 target = balanced_df['Units']
13
14 X_train4, X_test4, y_train4, y_test4 = train_test_split(features, target,
15                                                         test_size=0.2, random_state=83)
16
17 model = LogisticRegression(solver='sag', max_iter=200, random_state=83)
18
19 model.fit(X_train4, y_train4)
20
21 accuracy = model.score(X_test4, y_test4)
22 print(f"Accuracy: {accuracy}")
23
24 #0.9918032786885246

```

Code 35: (Python)

پس از متعادل سازی داده‌ها مشاهده می‌کنیم دقت شبکه عصبی افزایش یافته است. تکالیف درس تشخیص و شناسایی عیب می‌تواند در قالب \LaTeX (LaTeX) تحویل داده شوند.

۳ سوال سوم

۱.۳ 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 دانلودشده دیتاست را روی گوگل درایو خود قرار داده و با استفاده از دستور gdown آن را در محیط گوگل کولب بارگذاری کنید.

این مجموعه داده شامل اطلاعات مختلف در مورد سلامت نمونه‌ای از افراد هست. این مجموعه داده ۲۲ ستون برای شاخص‌های مختلف سلامت، شامل وضعیت فشار خون، وضعیت کلسترول، شاخص BMI، سیگاری بودن یا نبودن افراد، داشتن دیابت یا نداشتن آن، وضعیت فعالیت جسمی، میزان مصرف میوه، میزان مصرف سبزیجات، جنسیت، سن و دیگر اطلاعات می‌شود و در ستون اول وضعیت سگته قلبی را نشان می‌دهد.

```

1 !pip install --upgrade --no-cache-dir gdown
2 !gdown 1w1xGfzLg0n-WElysHsmnRSUC9-KXFKLF
3
4 import numpy as np
5 import pandas as pd
6 import matplotlib.pyplot as plt
7 from sklearn.model_selection import train_test_split

```



```
8
9 df = pd.read_csv('heart_disease_health_indicators.csv')
10 df.head()
```

Code 36: (Python)

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

برای تبدیل آن به دیتا فریم از دستور `pd.dataframe` استفاده می‌کنیم. با توجه به آنکه ستون `target` ما که همان ابتدا به سکت قلبی است، در ستون اول قرار دارد، باید آن را به ستون آخر منتقل کنیم که این کار با دستور `pop` و `insert` صورت می‌گیرد و مشخص می‌کنیم که در ستون آخر قرار گیرد.

```
1 df = pd.DataFrame(df)
2
3 first_column = df.pop('HeartDiseaseorAttack')
4 df.insert(len(df.columns), 'HeartDiseaseorAttack', first_column)
```

Code 37: (Python)

برای انتخاب ۱۰۰ داده از کلاس ۰ و ۱، دو کلاس را از هم جدا می‌کنیم و سپس ۱۰۰ تای اول برای هر کلاس را در یک دیتافریم جدید می‌ریزیم و در نهایت هر دو دیتا فریم جدید را با یکدیگر مخلوط می‌کنیم:

```
1 df_output_1 = df[df['HeartDiseaseorAttack'] == 1].head(100)
2 df_output_0 = df[df['HeartDiseaseorAttack'] == 0].head(100)
3
4 combined_df = df_output_1.append(df_output_0)
```

Code 38: (Python)



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

ستون‌های اول تا یکی مانده به آخر را به عنوان feature در نظر می‌گیریم و ستون آخر نیز target ما می‌باشد. و سپس با دستور train-test-split داده‌ها را به نسبت ۸۰ به ۲۰، به داده‌های آموزش و تست دسته‌بندی می‌کنیم.

```
1 X = combined_df.iloc[:,0:-1]
2 y = combined_df.iloc[:,-1]
3
4 x_train,x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
5 x_train.shape, y_train.shape, x_test.shape, y_test.shape
6
7 #((160, 20), (160,), (40, 20), (40,))
```

Code 39: (Python)

سپس کتابخانه‌های مورد نیاز خود برای آموزش و ارزیابی شبکه، شامل LogisticRegression, SGDClassifier، را import می‌کنیم. برای طبقه‌بند، LogisticRegression max-iter را برابر ۳۰۰ و random-state را برابر ۸۳ قرار می‌دهیم و شبکه خور را اندازه می‌گیریم.

```
1 from sklearn.linear_model import LogisticRegression, SGDClassifier
2
3 model = LogisticRegression(solver='sag', max_iter=300, random_state=83)
4 model.fit(x_train, y_train)
5
6 print(model.score(x_train, y_train))
7 print(model.score(x_test, y_test))
8 #0.75
9 #0.525
```

Code 40: (Python)

مشاهده می‌کنیم دقت شبکه ما برای داده‌های آموزش ۷۵ درصد و برای داده‌های تست ۵۲ درصد است که مقدار کمی دارد و دقت شبکه ما کم است.

برای کلاسیفایر SGD نیز کار را تکرار می‌کنیم:

```
1 model1 = SGDClassifier(loss= 'log_loss', random_state=83)
2 model1.fit(x_train, y_train)
3
```



```
4 print(model1.score(x_train, y_train))
5 print(model1.score(x_test, y_test))
6 #0.6875
7 #0.6
```

Code 41: (Python)

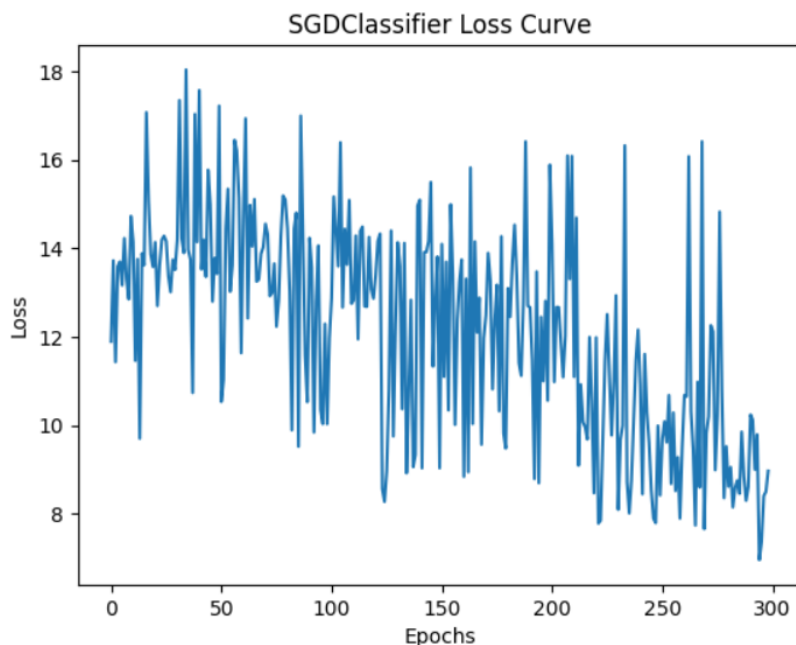
برای این روش، دقت شبکه ما تا حد کمی افزایش پیدا کرده است.

۴.۳ در حالت استفاده از دستورات آمادهٔ سایکیت لرن، آیا راهی برای نمایش نمودار تابع اتلاف وجود دارد؟ پیاده سازی کنید.

به طور مستقیم تابعی در sickitlearn برای نمایش تابع اتلاف وجود ندارد اما می‌توانیم از predict_proba که نشان دهنده‌ی میزان تفاوت تارگت ما با پیش‌بینی شبکه عصبی است این کار را انجام دهیم. برای اینکار ابتدا میزان پیش‌بینی شبکه را در حالت کلاسیفایر SGD محاسبه می‌کنیم. و برای تعداد آموزش‌ها شبکه اتلاف را محاسبه کرده و نمودار شکل آن را رسم می‌کنیم.

```
1 model1_proba= model1.predict_proba(x_train)
2
3 model1 = SGDClassifier(loss='log_loss', random_state=83)
4 model1.fit(x_train, y_train)
5
6 model1_proba = model1.predict_proba(x_train)
7
8 losses = []
9 for epoch in range(1, 300):
10     model1.partial_fit(x_train, y_train, classes=np.unique(y_train))
11     epoch_loss = log_loss(y_train, model1.predict_proba(x_train))
12     losses.append(epoch_loss)
13
14 plt.plot(losses)
15 plt.xlabel('Epochs')
16 plt.ylabel('Loss')
17 plt.title('SGDClassifier Loss Curve')
18 plt.show()
```

Code 42: (Python)



شکل ۱۴: شکل شماره ۱۴

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

یکی از شاخص‌های ارزیابی مهم برای مدل‌های طبقه‌بندی، ماتریس درهم‌ریختگی یا matrix confusion است که اطلاعاتی راجع به عملکرد مدل بر روی داده‌های تست ارائه می‌دهد. این ماتریس بر اساس پیش‌بینی‌های مدل و برچسب‌های واقعی داده‌های تست ساخته می‌شود. این ماتریس به صورت یک جدول دوبعدی نشان می‌دهد که مدل ما چه تعداد نمونه‌ها را به درستی دسته‌بندی کرده است و چه تعداد نمونه را اشتباه تشخیص داده است.

برای محاسبه‌ی کد آن، ماتریس درهم‌ریختگی را برای برچسب‌های واقعی و پیش‌بینی شده محاسبه می‌کنیم. plt.imshow از این دستور برای نمایش ماتریس درهم‌ریختگی استفاده می‌کنیم و رنگ معیار آن را آبی قرار می‌دهیم. سپس کلاس را مشخص می‌کنیم و از آن برای مشخص نمودن جایگاه اندیس‌ها استفاده می‌کنیم. برای مشخص کردن شدت تیره و روشن بودن رنگ آبی، مقدار ترش‌هولدی برای آن در نظر می‌گیریم. و محاسبات را درون ماتریس می‌نویسیم.

```
1 from sklearn.metrics import confusion_matrix
```

```
2
```

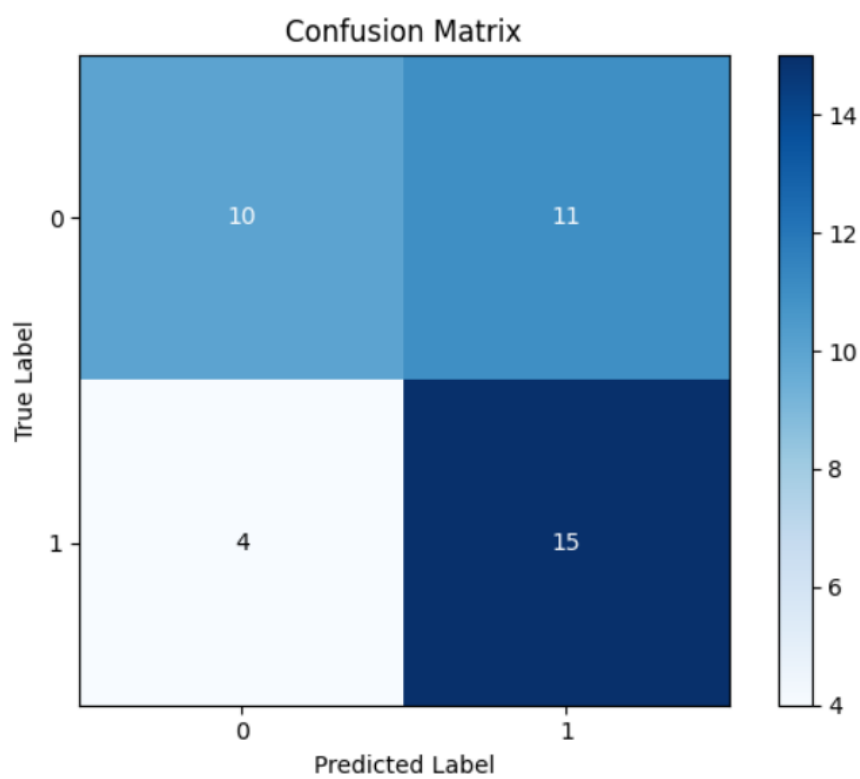
```
3 y_pred = model.predict(x_test)
```

```
4
```



```
5 conf_matrix = confusion_matrix(y_test, y_pred)
6
7 plt.imshow(conf_matrix, cmap='Blues', interpolation='nearest')
8 plt.title('Confusion Matrix')
9 plt.colorbar()
10
11 classes = [0, 1]
12 tick_marks = np.arange(len(classes))
13 plt.xticks(tick_marks, classes)
14 plt.yticks(tick_marks, classes)
15
16 thresh = conf_matrix.max() / 2.
17 for i in range(conf_matrix.shape[0]):
18     for j in range(conf_matrix.shape[1]):
19         plt.text(j, i, format(conf_matrix[i, j], 'd'),
20                 ha="center", va="center",
21                 color="white" if conf_matrix[i, j] > thresh else "black")
22
23 plt.xlabel('Predicted Label')
24 plt.ylabel('True Label')
25 plt.tight_layout()
26 plt.show()
```

Code 43: (Python)



شکل ۱۵: شکل شماره ۱۵