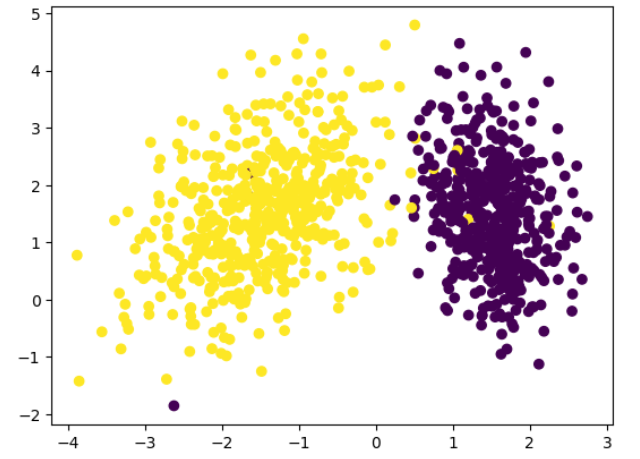
به نام خدا

**گزارش مینی پروژه 1**

امیر ارشام بهشتی طهرانی مبانی سیستم های هوشمند شماره دانشجویی:9820293

**سوال‏1)**

1\_ : ابتدا با استفاده از دستور make\_classification در sklearn.datasets کلاس های خود را با ویژگی x و خروجی (target) y ایجاد می کنیم. در داخل تابع make\_classification ، نیاز داریم تا پارامتر ها را تنظیم کنیم. ابتدا تعداد نمونه ها (n\_samples) را برابر 1000 و تعداد ویژگی‏ها (n\_features) و تعداد کلاس ها را هم (n\_classes) برابر 2 قرار می دهیم. برای تنظیم نمایش و جداسازی بهتر داده ها هم ابتدا نحوه توزیع آن ها (n\_clusters\_per\_class) برابر 1 و سپس n\_redundant را برابر صفر قرار می دهیم. class\_sep ، میزان سختی جداسازی و در هم مخلوط شدن داده ها را نشان می دهد که به طور مثال ابتدا 5/1 در نطر می گیریم. در آخر هم با قرار دادن random\_state برابر با 93، داده به صورت مشخص هر بار تولید می شوند.

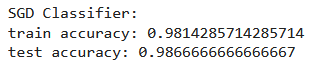


2\_: ابتدا با دستور train\_test\_split داده ها را به دو بخش آموزش و تست تقسیم می کنیم تا دقت مدل را بر روی داده های دیده نشده تست ارزیابی کنیم و دقت مدل را بسنجیم. نسبت تقسیم را 30 درصد قرار می دهیم و در نتیجه 300 داده تست و 700 داده آموزش خواهیم داشت.

سپس مدل اول خود را logistic\_regression از کتابخانه sklearn.linear\_modelانتخاب می کنیم. با دستور model.fit و پارامتر های x و y مربوط به train، مدل خود را بر داده های آموزشی قرار می دهیم تا آن ها را طبقه‏بندی و از هم جدا کند. با دستور model.predict و پارامتر x\_test، خروجی مدل با توجه به ورودی داده های تست را محاسبه می کنیم که در مباحث تئوری همان y\_hat خواهد بود. در نهایت با استفاده از دستور model.score، دقت مدل خود را ارزیابی می کنیم. به این صورت که یک بار دقت داده های آموزشی را با دادن x و y train و بار دیگر دقت داده های تست را به همین صورت می سنجیم. طبیعتا انتظار داریم که دقت داده های train چون مدل با آن ها آموزش داده شده بیشتر باشد.

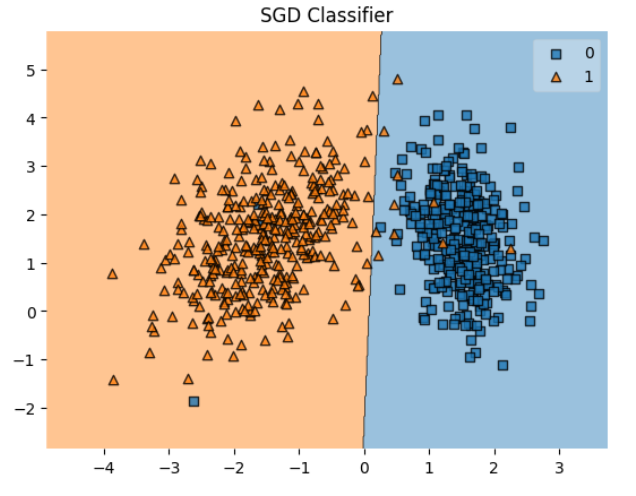
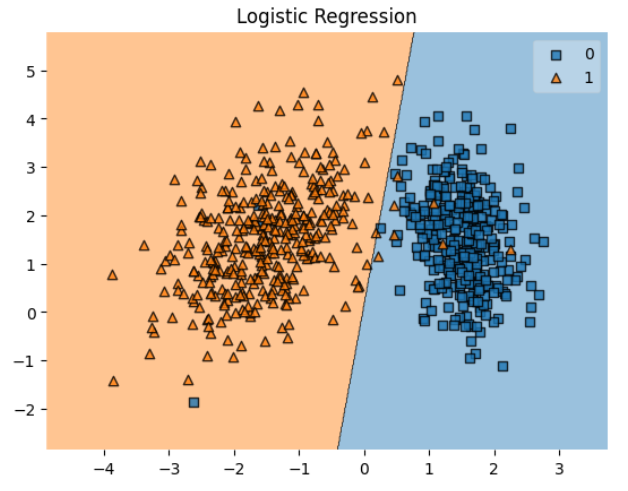
برای مدل دوم از طبقه‏بند، SGDClassifier که بر اساس گرادیان نزولی تصادفی عمل می کند استفاده می کنیم و سایر قسمت ها را مانند مدل قبل انجام می دهیم.

برای بهبود نتایج مدل، در مدل اول با تغییر فراپارامتر max\_iter، که تعداد دور های آموزش مدل توسط داده ها می باشد را تغییر داده و برابر 500 قرار می دهیم. در مدل دوم نیز غلاوه بر max\_iter، learning\_rate که ضریب تغییر و آپدیت وزن ها در گرادیان نزولی بود را می توانیم تغییر دهیم که در اینجا ‘optimal’ قرار می دهیم که طبق توابعی از پیش تعریف شده، نرخ یادگیری را مشخص می کند.

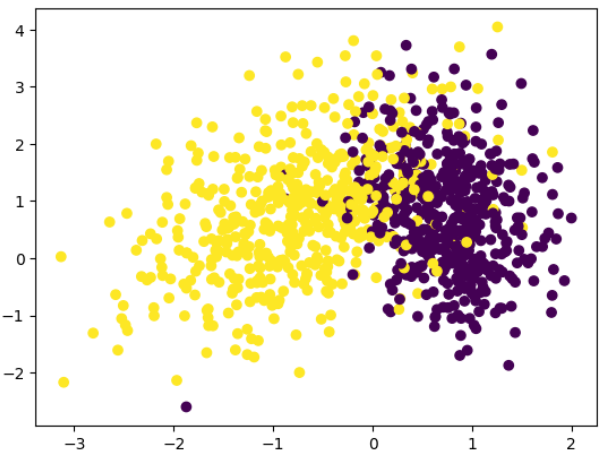


3\_: می توانیم با استفاده از کتابخانه mlxtend.plotting و ابزار plot\_decision\_regions در آن، مرز نواحی تصمیم گیری و کلاس ها را با اشکال متفاوت نمایش دهیم.

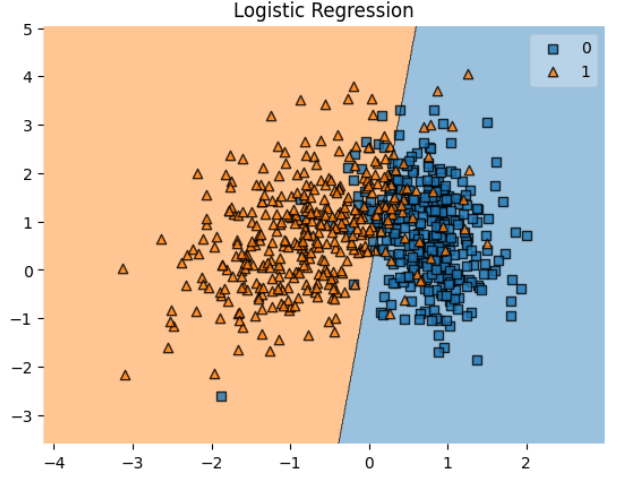
که x و y مربوط به train یا test را به عنوان ورودی به همراه مدل مورد نظر به این تابع می‏دهیم.

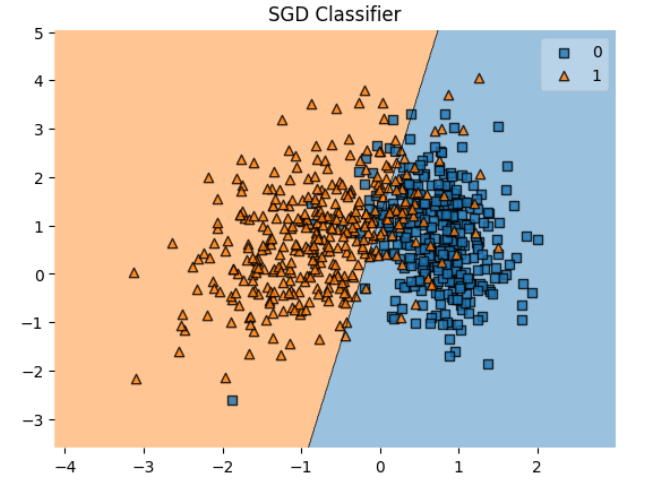


4\_: اگر مقدار پارامتر class\_sep، در make\_classification را کوچک تر کنیم داده ها بیشتر درهم مخلوط می شوند و جداسازی آن ها سخت تر خواهد بود. این پارامتر را برابر با 75/0 قرار داده و مشاهده می کنیم داده ها در هم مخلوط می شوند. سپس با تکرار مراحل بالا آن ها را از هم جدا می کنیم و در نهایت دقت داده های آموزش و تست را محاسبه می کنیم که مشاهده می شود نسبت به قسمت قبلی دقت کاهش یافته است.









سوال 2 )

**1\_ :** ابتدا برای باز کردن دیتاست، پس از اینکه فایل دیتاست را در google drive قرار دادیم، با استفاده از دستور gdown و آی دی مورد نظر فایل، در محیط google colab بارگذاری می‏کنیم.

پس از اینکه با دستور pd.read\_csv، محتویات دیتاست را داخل دیتافریم df قرار می دهیم، مشاهده می شود که یکی از سطر ها به جای عنوان های ستون ها قرار گرفته. درنتیجه ابتدا با دستور df.columns، و تبدیل آن به list، اعداد سطری که به اشتباه در عنوان قرار دارد را داخل r1 دخیره می کنیم. با دستور df.loc[len(df)] و برابر قرار دادن آن با r1، r1 به عنوان ردیف آخر (1372) به دیتافریم اضافه می شود زیرا df.loc ردیف مورد نظر را در آرگومان خود که طول دیتافریم است اضافه می کند.

در انتها هم با توجه به مشخصات دیتاست داخل لینکی که از آن دانلود شده، با دستور df.columns، هر ستون را به درستی نام گذاری می کنیم.

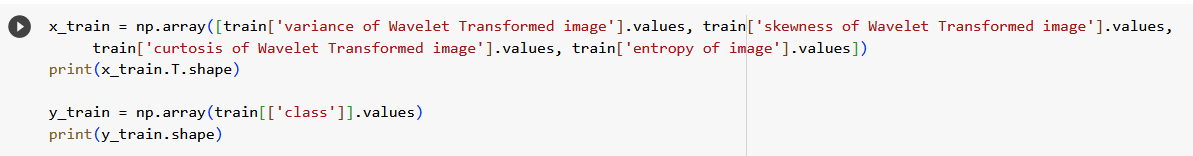
**2\_ :** اصلی ترین دلیل بُر زدن داده ها قبل از تقسیم به دو دسته train و test، جلوگیری از بایاس شدن مدل نسبت به داده هاست. در حقیقت اگر داده ها بُر زده نشوند، همواره اعداد خاصی در train و test قرار می گیرند و مدل با توجه به آنها یادگیری و ارزیابی را انجام می دهد. همچنین چون به ترتیب ایندکس عمل جداسازی انجام می شود، ممکن است همه داده ها با یک کلاس خاص در داده های test قرار بگیرند و یا تعداد کلاس ها در train، نابرابر شود و یادگیری و ارزیابی مدل نادرست شود.

با استفاده از train\_test\_split ، داده ها را به دو دسته آموزش (train) و ارزیابی (test) تقسیم می کنیم. در آرگومان های آن دیتافریم df را قرار داده و %30 درصد داده ها را برای test انتخاب می کنیم. با قرار دادن shuffle = True، داده ها را بُر میزنیم و سپس تقسیم می‏کنیم. با قرار دادن random\_state = 93، همواره داده های مشخصی در هر دور تقسیم می‏شود و مدل را با آن ها تنظیم می کنیم.

در نتیجه داده ها در train و test به صورت دیتافریم قرار می گیرند.

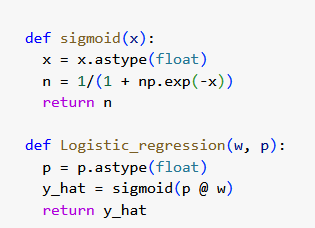
**3\_ :** ابتدا نیاز داریم تا ورودی ها و خروجی (label) را در داده های train و test را با x و y، مشخص کنیم. چون train و test دیتافریم هستند، در نتیجه با دستور .values برای هر ستون آن ها که مربوط به feature های نمونه ها هستند (همه ستون ها به جز ‘class’) ، مقدار هر ستون را به ترتیب در یک آرایه قرار می دهیم تا x\_train حاصل شود اما چون آرایه 4\*960 خواهد بود، در ادامه با ترانهاده آن (x\_train.T )کار خواهیم کرد.

y\_train هم به صورت بالا خواهد بود اما تنها به ستون class که مربوط به خروجی داده هاست نیاز خواهیم داشت.

****

به همین صورت با توجه به داده های test، x\_test و y\_test را هم تعریف می کنیم.

با توجه به مدل classifer، باید قسمت های مختلف آن را به صورت توابع کدنویسی کنیم. ابتدا نیاز داریم تا تابع داخل مدل classifier یعنی تابع sigmoid را پیاده سازی کنیم. سپس مدل classifier که Logistic Regression می باشد را کدنویسی می کنیم. این تابع دو پارامتر ورودی های آموزشی و وزن های خط جداکننده را می پذیرد. با دستور .astype(float) ورودی ها را در حالت float قرار داده و ضرب ماتریسی آن با وزن ها را به تابع sigmoid می دهیم و در نهایت خروجی مدل Logistic Regression ، y\_hat خواهد بود که ابعاد آن (960\*1) خواهد بود.



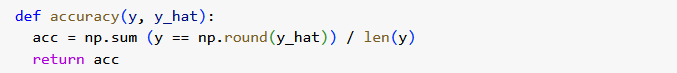
برای تابع اتلاف، BCE (Binary Cross Entropy) را تعریف می کنیم که پارامتر های آن yهای آموزشی و y\_hat های خروجی مدل است و در نهایت میانگینی از مجموع خطا ها در خروجی نمایش داده که در ادامه سعی می کنیم این خطا را کاهش دهیم.

برای بخش بهینه ساز (Optimizer) که وزن ها را update می کند تا خطا کاهش یابد، از روش گرادیان نزولی استفاده می کنیم که طبق رابطه آن ابتدا تابع گرادیان (GD) که مشتق اول تابع اتلاف و همان BCE در w است را تعریف کرده و با استفاده از آن در تابع optimizer بهینه سازی را انجام می دهیم که خروجی آن w (وزن های جدید) خواهد بود. در این تابع eta، learning rate در تابع گرادیان نزولی بوده که مقدار آن را 0.05 انتخاب می کنیم.

برای کدنویسی روند آموزش باید این روند در تعداد بار های مشخصی انجام شود (epoch). ابتدا با استفاده از تابع random.randn(4\*1) ، w ها را به صورت رندوم و در حالت 4\*1 تولید می‏کنیم.

سپس با یک حلقه for در هر epoch فرایند آموزش را انجام می دهیم. w و x\_train.T را به تابع مدل Logistic Regression داده و سایر قسمت ها را همان زور که گفته شد کامل می‏کنیم.

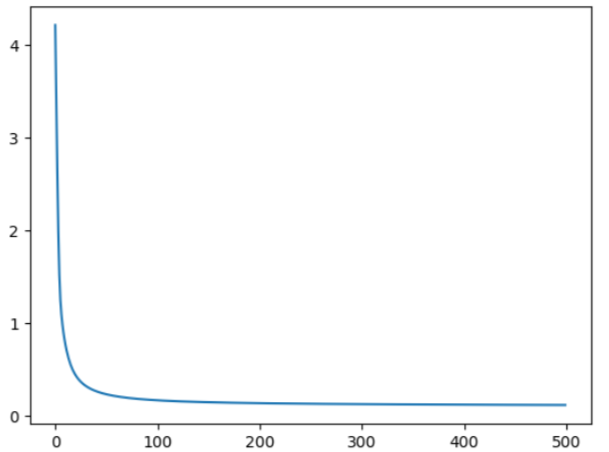
در نهایت برای ارزیابی مدل، نیاز به تعریف یک accuracy و دقت مدل خواهیم داشت. با تعریف به صورت زیر، تعداد y\_hat هایی که به درستی انتخاب شده اند و مانند خروجی y ها هستند را بر تعداد y ها تقسیم می کنیم.



و سپس x و y های train را به عنوان ورودی به آن می دهیم تا دقت را بر روی داده های آموزشی بسنجیم. ولی برای عملکرد صحیح مدل باید دقت مدل برای داده های ارزیابی (test) را مبنا قرار دهیم. برای این کار این بار فقط ورودی داده های test (x\_test) را به ورودی مدل می‏دهیم و فرایند آموزشی رخ نمی دهد. سپس خروجی هایی که مدل تولید می کند را با خروجی های متناظر داده های test طبق معیار accuracy تعریف شده می سنجیم و دقت مدل بر روی داده های test هم مشخص می شود.



برای رسم نمودار تابع اتلاف نیاز داریم تا خطا ها (خروجی بخش تابع اتلاف BCE) را در یک لیست دخیره کنیم و با تابع plt.plot() آن را نمایش دهیم. با تعریف لیست خالی error و .append() کردن خروجی BCE در هر مرحله خطا ها در error قرار گرفته و آن ها را در نمودار نمایش می دهیم.



طبق این نمودار مشخص می شود که داده های آموزشی به سرعت تفکیک شده اند زیرا طبق جدول حدودا در epoch=20 خطا کاهش زیادی داشته و در ادامه هم به تقریبا 0.1 همگرا می‏شود.

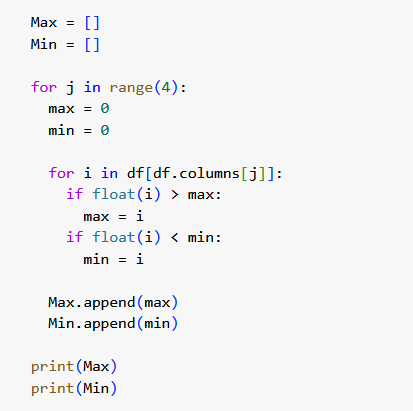
**پاسخ سوال:**

خیر نمی توان با قطعیت نظر داد زیرا تابع اتلاف برای ورودی های آموزشی که با آن ها مدل را آموزش داده ایم تا خروجی ها را مشخص کند، خطا را مشخص می کند ولی ما نیاز داریم تا خطا برای داده های دیده نشده که مدل طبق آن ها تنظیم نشده است را مشاهده کنیم. برای رفع این مشکل با استفاده از تابع accuracy برای داده های test دقت مدل بر روی آن ها را می سنجیم.

4\_ : **نرمالیزاسیون،** یکی از روش ها در مرحله پیش‏پردازش داده‏ها است که برای جلوگیری از بایاس مدل نسبت به feature هایی که مقادیر زیادی نسبت به بقیه دارند استفاده می شود؛ زیرا ممکن است این مقادیر زیاد بر مدل غالب شوند که با نرمالیزاسیون همه feature ها به نسبت یکسانی بر مدل اثر می‏گذارند و رنج مقدار های آن ها بین 0 و 1 می شود.

دو روش نرمالیزاسیون، و هستند که از روش اول که رنج داده ها را بین بازه [0 , 1] قرار می دهد، استفاده می کنیم.

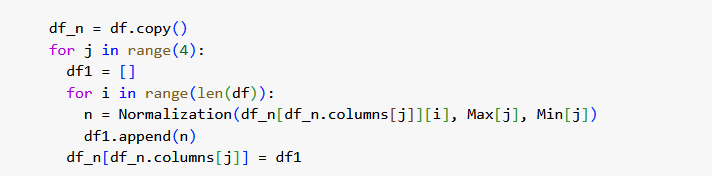
ابتدا مقدار max و min را در هر ستون دیتافریم به جز کلاس ها می یابیم و آن را داخل لیست max و min قرار می دهیم.



که df[df.columns[j]] مقدار اعداد هر ستون تا ستون سوم دیتافریم df است.

حال ابتدا تابع نرمالیزاسیون را مطابق رابطه نرمالیزاسیون اول در بالا تعریف می کنیم.سپس دیتافریم df\_n را مانند df تعریف کرده و برای قسمت نرمالیزاسیون با آن کار می کنیم و در هر دور حلقه ستون های آن را با تابع نرمالیزاسیون، نرمالیزه می کنیم.

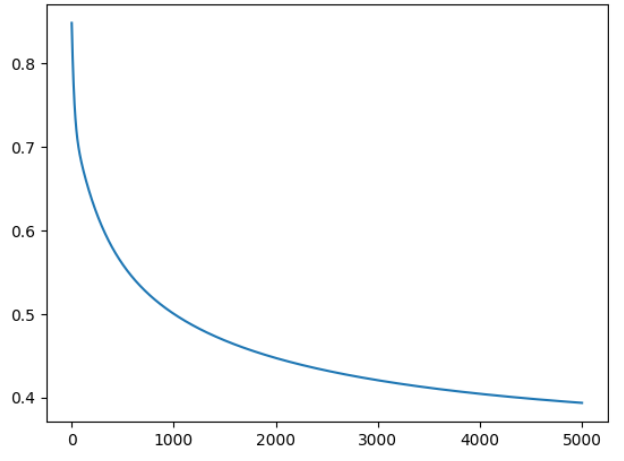
به این صورت که برای هر عدد در ستون مورد نظر، آن عدد (df\_n[df\_n.columns[ j ]][i]) و max در آن ستون (max[j]) و min در آن ستون (min[j]) را به تابع normalization داده و خروجی را در df1 که لیست خالی است قرار داده و سپس آن ستون دیتافریم را (df\_n[df\_n.columns[ j ]) با df1 جایگزین می کنیم.



**پاسخ سوال:** بله زیرا مدل با داده های آموزشی با ابعاد و scale متفاوتی آموزش دیده در نتیجه داده های تست هم باید در همین ابعاد و رنج ، مدل را ارزیابی کنند.

5\_ : مانند قسمت های قبل ولی این بار df\_n را به قسمت های train و test تقسیم کرده و مانند قسمت های قبل مراحل را تکرار می کنیم. اما مشاهده می شود که این بار مدل با epoch های خیلی بیشتری باید آموزش ببیند تا خطا کاهش یابد و همچنین میزان دقت مدل برای داده های train و test ، کمتر شده است که می تواند به دلیل نزدیک شدن feature های مختلف به هم و متمایز نبودن feature های مختلف از هم نسبت به قبل باشد که باعث شده تفکیک کلاس ها از هم مشکل شود.





6\_ : در این قسمت با دیتافریم df\_b که مانند df است پیش می رویم.

تعداد نمونه ها با کلاس 1 را با استفاده از تابع len() بر df [ df[‘class’] ==1] پیدا می کنیم و همین طور برای کلاس صفر. که مشخص می شود کلاس صفر، 761 نمونه و کلاس1، 610 نمونه دارد.

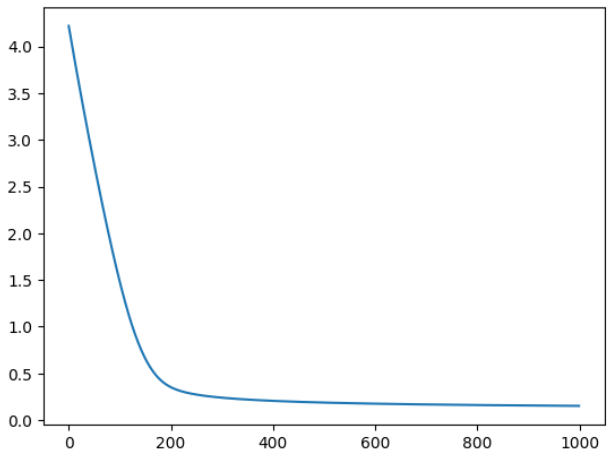
عدم تعادل در دیتاست و کلاس ها می تواند به بایاس شدن مدل نسبت به یکی از کلاس ها شود زیرا مدل با داده های کلاس بیشتر اموزش می بیند.

برای حل این موضوع می توان کلاس بیشتر را undersample کرد و تعداد آن را با تعداد کلاس کمتر برابر کرد. یا اینکه کلاس کمتر را oversample کرد و تعداد آن را با کلاس بیشتر برابر قرار داد که در اینجا از روش undersample استفاده می کنیم.

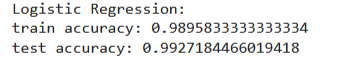
به این صورت که با دستور .sample( n = 610) که طول کلاس 1 است به صورت رندوم (ولی با random state =93) 610 نمونه از این کلاس را انتخاب کرده و در کلاس صفر دیتافریم جدید قرار می دهیم تا تعداد کلاس ها برابر شود.

در ادامه مانند قبل مراحل را تکرار کرده و مشاهده می کنیم که مدل دقت بالاتری دارد و تابع اتلاف به نسبت حالت عادی در epoch بالاتری حدود 180 کاهش زیادی یافته است.





7\_: مانند مراحل قبل ابتدا x , y train و test را آماده کرده و سپس با مدل Logistic Regression مانند سوال یک و با دستور های .fit ، .predict ، و .score مدل را آماده می کنیم که مشاهده می شود دقت بسیار بالایی نسبت به حالت کدنویسی از ابتدا دارد.



**سوال 3)**

1\_ : فایل دیتاست شامل شاخص هایی مرتبط با وضعیت سلامتی، سبک زندگی و ... برای نمونه هایی از افراد است که می توان ریسک های احتمالی و تشابهات برای وضعیت سلامتی و بیماری قلبی را بررسی کرد. این دیتاست شامل 22 ستون مانند حمله قلبی (کلاس) ، فشار خون، BMI، دیابت، جنسیت افراد، سن، سلامت روان و ... را دارد و 251661 نمونه دارد.

2\_ : ابتدا با دستور pd.read\_csv محتویات فایل را داخل دیتافریم قرار داده و سپس مانند قبل با دستور .sample(n = 100 , random\_state =93) ، 100 نمونه از کلاس صفر و 100 نمونه از کلاس 1 را انتخاب کرده و داخل دو دیتافریم df\_0 و df\_1 قرار می دهیم و با دستور df.concat ، این دو دیتافریم را به هم متصل کرده و داخل دیتافریم df جدید قرار می دهیم.

3\_ : مانند مراحل بالا ابتدا داده های test و train را با نسبت 2/0 جدا کرده و برای y هم تنها داده ها در ستون مربوط به حمله قلبی (که خروجی دیتاست است) را با نسبت 2/0 جدا می کنیم.

چون برای جداسازی x\_train و x\_test ، df را به عنوان ورودی به train\_test\_split دادیم، نیاز داریم تا با دستور .pop() ستون مربوط به کلاس ها (ستون حمله قلبی) را حذف کنیم.

سپس مانند قبل مدل را با دو حالت Logistic Regression و SGD Classifier پیاده سازی می کنیم و در هر حالت دقت مدل را بر داده های train و test می سنجیم.

