

مینی پروژه 1 درس یادیگیری ماشین

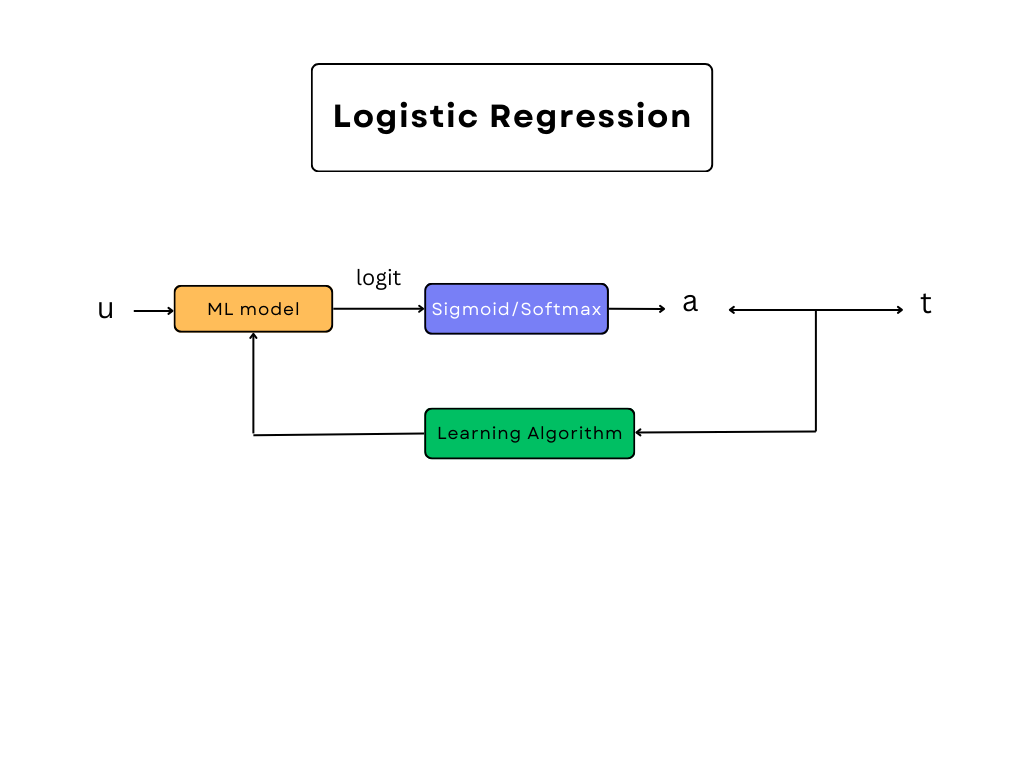
علی باقری‌نژاد

40202854

فهرست

سؤال 1

1-1



شکل 1بلوک دیاگرام مدل طبقه بندی خطی (Logistic Regression)

شکل 1 بلوک دیاگرام مدل طبقه بندی خطی[[1]](#footnote-1) را به نمایش در آورده است. این مدل برای هر دو حالت دسته‌بندی[[2]](#footnote-2) دو کلاسه و چند کلاسه صادق است. در ادامه به توضیح مختصر عملکرد هر یک اجزاء این بلوک دیاگرام می‌پردازیم.

**عملکرد کلی مدل طبقه‌بندی خطی:**

* در اولین مرحله، ورودی به‌صورت یک بردار (**u**) وارد بلوک یادگیری ماشین می‌شود.
* در بلوک نارنجی یک عملیات ضرب داخلی ماتریسی روی ورودی انجام می‌شود. عملیات ذکر شده به‌صورت زیر است:

در این مرحله، یک بردار وزن در u، ضرب می‌شود و یک عدد تولید می‌کند. مقادیر موجود در بردار وزن مشخص می‌کنند که هر کدام از ویژگی‌ها چه مقدار اهمیت دارند. خروجی بلوک یادگیری ماشین، یک عدد است (در صورت حالت چند کلاسه یک بردار به‌اندازه تعداد کلاس‌ها).

* خروجی بلوک یادگیری ماشین (logit) وارد بلوک تابع فعال‌سازی[[3]](#footnote-3) می‌شود. دلیل این که از خروجی بلوک ML به‌صورت خام استفاده نمی‌کنیم این است که خروجی بلوک ذکر شده یک عدد در بازه می باشد ولی ما نیاز به این داریم که خروجی نهایی عدد 0 یا 1 (در صورت دوکلاسه بودن طبقه بندی) باشد. در این مرحله از توابعی مثل sigmoid استفاده می کنیم که خروجی logit را گرفته و به عنوان خروجی نهایی مدل، یک عدد بین 0و1 تحویل می دهد. بر اساس مقدار آستانه‌ای[[4]](#footnote-4) که به دلخواه تعیین می کنیم، بر اساس خروجی a، یک کلاس به ورودی اختصاص می دهیم.
* خروجی a با برچسب[[5]](#footnote-5) t مقایسه می‌شود تا عملکرد مدل ماشین لرنیگ مقایسه شود. با مقایسه a و t و بر اساس روش‌هایی همچون MSE یا MAE یا... یک خطایی برای مدل در نظر می‌گیریم و در ادامه با استفاده از الگوریتم یادگیری[[6]](#footnote-6) (یا الگوریتم بهینه‌سازی[[7]](#footnote-7)) وزن‌های مدل را بهبود می‌دهیم.
* اتفاقی که در بلوک یادگیری می‌افتد این که است که با استفاده از خطا به‌دست‌آمده و روش‌هایی همچون SGD یا Adam یا... که عموماً الگوریتم‌های گرادیان بیس هستند، وزن‌های موجود را به صورتی تغییر می‌دهیم که مقدار تابع هزینه[[8]](#footnote-8) به کمینه خود نزدیک شود. به‌عنوان‌مثال فرمول ریاضی الگوریتم گرادیان نزولی[[9]](#footnote-9) را نمایش می‌دهیم:

**تفاوت طبقه‌بندی دو کلاسه و چند کلاسه:**

هنگامی طبقه‌بندی دو کلاسه، اگر ماشین، خروجی را بیشتر از 0.5(در صورتی که مقدار آستانه 0.5 باشد) قرار داد، ورودی متناظر را به کلاس 1 متعلق می‌دانیم ولی اگر خروجی را کمتر از 0.5 قرار داد، ورودی متناظر را به کلاس 0 اختصاص می‌دهیم. یعنی در حالت طبقه بندی دو کلاسه، صرفاً با داشتن یک عدد به عنوان خروجی می‌توانیم طبقه بندی را انجام دهیم. اما در حالتی که بیشتر از 2 کلاس را طبقه بندی می‌کنیم، دیگر نمی‌توانیم صرفاً با یک عدد طبقه بندی را انجام دهیم. در این حالت باید درصد تعلق ورودی را به هر یک از کلاس‌های خروجی را محاسبه کنیم. مثلاً اگر 5 کلاس به عنوان خروجی داشته باشیم، ماشین باید توانایی این را داشته باشد که 5 عدد به عنوان احتمال تعلق برای هر کلاس تحویل بدهد. یعنی می‌توان گفت که در حالت چند کلاسه، ماشین باید چندین مسئله را حل کند. هر مسئله نیز در واقع احتمال تعلق ورودی به هر یک از کلاس‌ها است. یعنی ماشین باید بررسی کند که احتمال تعلق ورودی به هر کلاس خروجی چقدر است.

در حالت طبقه‌بندی دو کلاسه، خروجی ماشین تنها یک عدد بود؛ ولی در طبقه‌بندی چند کلاسه، ماشین باید به تعداد کلاس‌ها خروجی داشته باشد. یعنی باید بلوک نارنجی (ML model) را به‌گونه‌ای تغییر دهیم که در خروجی تعداد عدد بیشتری را نمایش دهد. البته باید بلوک آبی (Activation function)، برای تمامی خروجی‌ها عمل کند؛ یعنی خروجی این بلوک هم به تعداد خروجی‌های بلوک نارنجی است.

1-2 ایجاد دیتاست

در اولین قدم، با استفاده از دستور make\_classification و ویژگی‌های زیر، دیتاست موردنظر را ایجاد می‌کنیم.

n\_samples = 1000

n\_features = 3

n\_classes = 4

n\_informative = 3

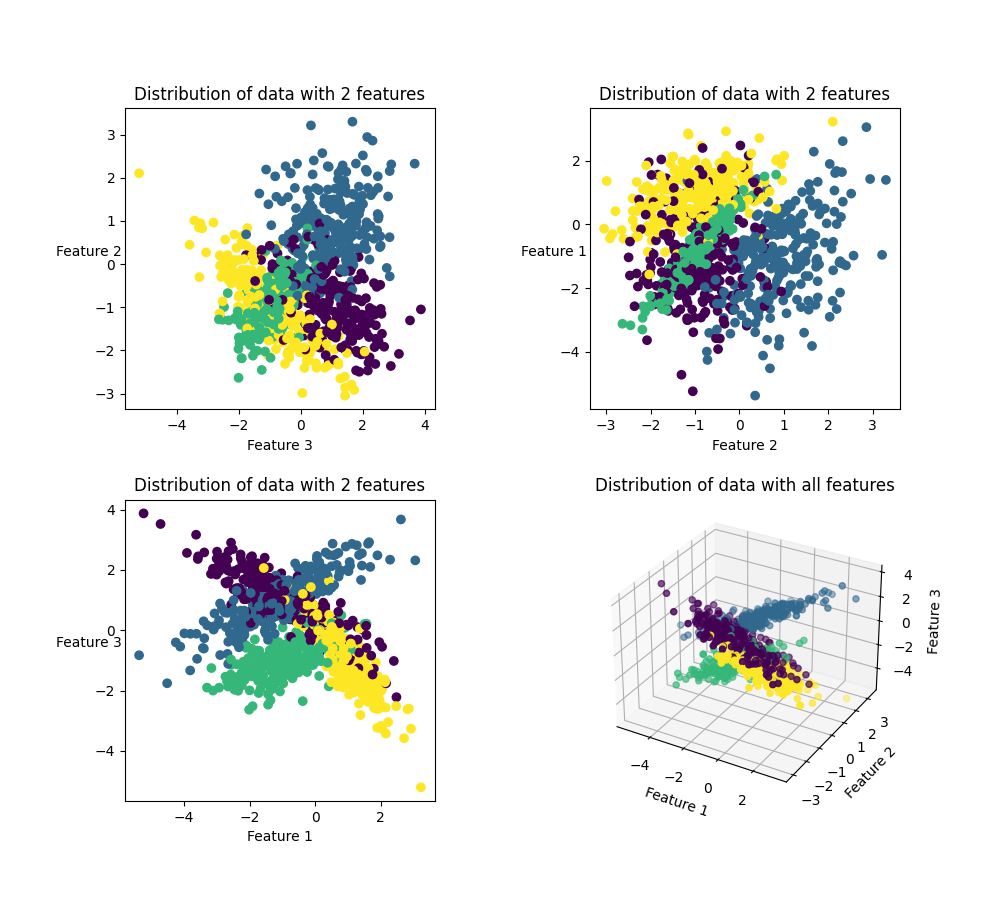
n\_redundant = 0

n\_repeated = 0

n\_clusters\_per\_class = 1

clss\_sep = 3

random\_state = 69

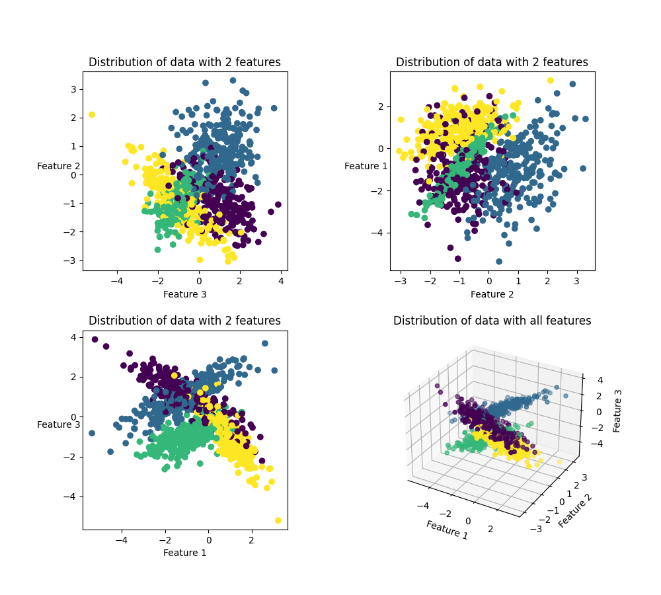
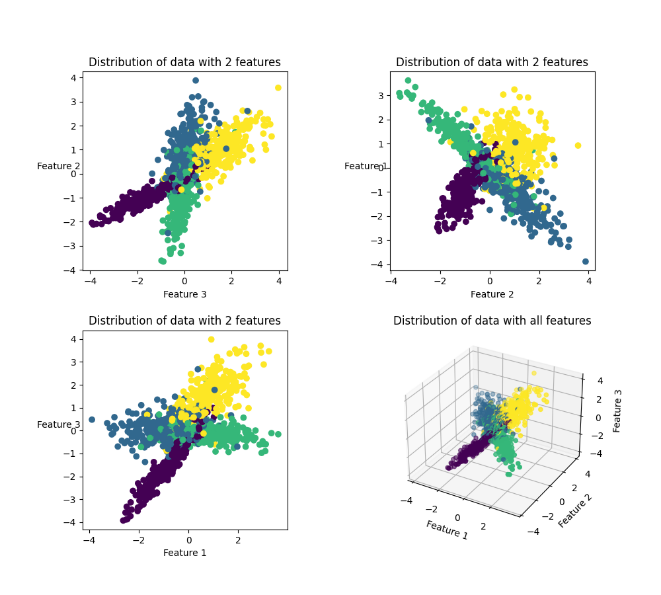


شکل 2 دیتاست ساخته شده

باتوجه‌به شکل 2، می‌توان گفت که دیتاست ساخته شده برای طبقه بندی، مناسب نیست؛ زیرا با توجه به شکل، نقاط به خوبی از هم جدا نشده اند و نقاط داده‌ی[[10]](#footnote-10) کلاس‌های مختلف به صورت خطی قابل تفکیک[[11]](#footnote-11) نیستند.

برای دشواری بیشتر می‌توان از راهکارهای زیر استفاده کرد:

* کاهش ویژگی class\_sep: این ویژگی نماینده میزان جداسازی و فاصله دسته داده‌ها است. هر چه مقدار این ویژگی کمتر باشد، داده‌ها از هم کمتر جدا بوده و جداسازی آن‌ها دشوارتر است.
* ایجاد عدم تعادل[[12]](#footnote-12) در داده: برای انجام این کار می‌توان نسبت تعداد داده‌های هر کلاس به هم را تغییر داد تا تعداد آن‌ها یکسان نباشد. این عمل با استفاده از ویژگی weight انجام می شود. با تغییر تعادل در داده، امکان یادگیری اشتباه برای ماشین وجود خواهد داشت.
* اعمال نویز[[13]](#footnote-13): با اعمال نویز به داده، می‌توان در روند یادگیری ماشین اخلال ایجاد کرد و دقت آن را کاهش داد.
* افزایش استقلال خطی بین ویژگی‌ها: ویژگی n\_informative تعیین می‌کند که چه تعداد از ویژگی‌ها دارای استقلال خطی باشند. در دیتاست ایجاد شده، n\_informative برابر 3 در نظر گرفته شده است. اگر این مقدار را کاهش دهیم و مثلاً n\_redundant را افزایش دهیم، دیتاست راحتر قابل تفکیک خواهد بود.



شکل 3دیتاست با n\_redundant های مختلف

شکل 3، دو تصویر از دیتاست‌هایی با استقلال خطی مختلف بین ویژگی‌هایشان را به نمایش گذاشته است. شکل سمت راست دارای استقلال خطی کمتری نسبت به شکل سمت چپ می باشد. همانطور که مشاهده می‌شود، شکل با استقلال خطی کمتر، قابلیت تفکیک پذیری خطی بهتری دارد.

1-3 ایجاد مدل طبقه بندی خطی

در اولین قدم، داده‌های ایجاد شده در بخش قبل را به دسته‌های آموزش[[14]](#footnote-14) و اعتبارسنجی[[15]](#footnote-15) تقسیم می‌کنیم. این تقسیم با استفاده از دستور train\_test\_split از کتابخانه scikit-learn انجام شد و نسبت تقسیم داده‌ها نیز 80 به 20 بود.

پس از تقسیم‌کردن داده‌ها به دسته‌های آموزش و صحه‌سنجی، با استفاده از تابع StandardScaler از کتابخانه scikit-learn، یک تبدیل‌گر برای داده‌های آموزش ایجاد کرده و بر اساس آن داده‌های آموزش و صحه‌سنجی را استاندار سازی کردیم.

LogisticRegression

اولین مدلی که برای ایجاد ماشین یادگیری از آن استفاده کردیم، مدل رگرسیون لجستیک بود. این مدل با استفاده از تابع LogisticRegression از کتابخانه scikit-learn ایجاد شد که پارامترهای استفاده شده برای آموزش در ادامه آورده شده‌اند:

penalty = 'l2'

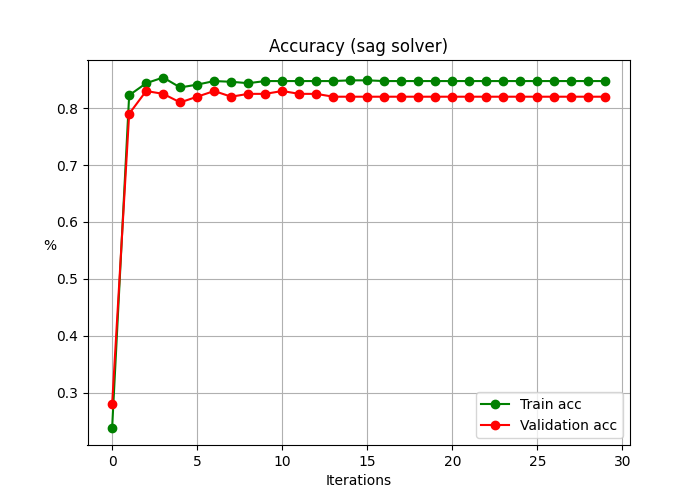
fit\_intercept = True

random\_state = 69

البته این فراپارامترها[[16]](#footnote-16)، فراپارامترهایی بودند که برای مقایسه تمامی مدل‌های رگرسیون لجستیک ثابت در نظر گرفته شدند. دو فراپارامتر تعداد تکرار[[17]](#footnote-17) و حلگر[[18]](#footnote-18) به‌عنوان فراپارامترهایی در نظر گرفته شدند که عملکرد مدل به‌ازای مقادیر مختلف آن‌ها مورد آزمایش قرار گرفت.

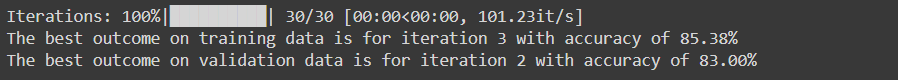
البته لازم به ذکر است که نرخ یادگیری[[19]](#footnote-19) برای تابع رگرسیون خطی کتابخانه scikit learn قابل‌تعریف نبود.

**تعداد تکرار:** تعداد تکرار مدل از 1 بار تکرار تا 30 بار تکرار تست شد.



شکل 4نمودار دقت مدل نسبت به تعداد تکرار

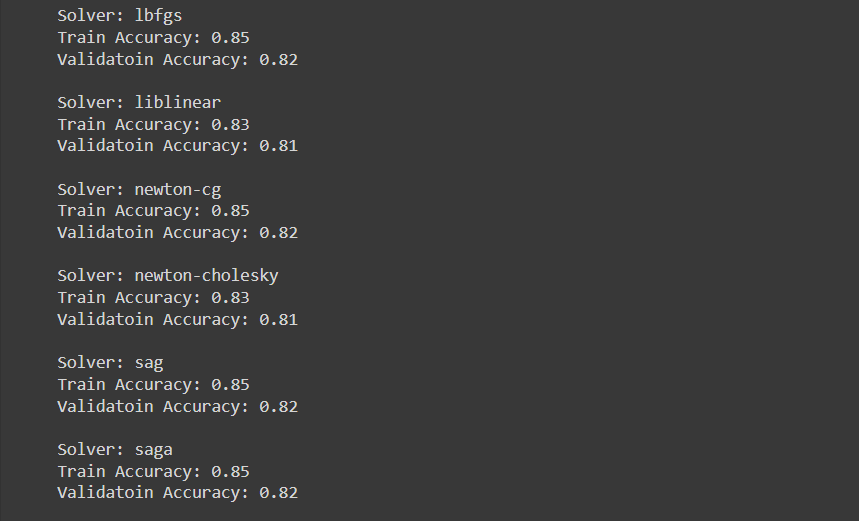
شکل 4 دقت مدل را برای مجموعه داده آموزش و صحت سنجی به نمایش گذاشته است.



شکل 5 دقت و بهترین تکرار برای مجوعه داده ها

شکل 5 مقدار بهترین دقت هر مجموعه داده و تکرار متناظر با آن را به نمایش گذاشته است.

**حلگر:** عملکرد مدل به‌ازای تعداد تکرار 15 برای حلگرهای مختلف به صورت زیر می‌باشد:



شکل 6 عملکرد مدل به ازای حلگر ها

باتوجه‌به شکل 6، حلگرهای lbfgs، newton-cg، sagو saga بهترین عملکرد را داشتند.

باتوجه‌به تابع LogisticRegression ای که در کتابخانه scikit-learn تعریف شده است، نمی‌توانستیم که خلاقیت‌های زیادی برای بهبود عملکرد مدل را اعمال کنیم. مثلاً این که در این تابع امکان تعریف نرخ یادگیری وجود نداشت و یا این که ضریب جریمه[[20]](#footnote-20) را نمی‌توانستیم تغییر دهیم و تعیین کنیم. تمامی کارهایی که می‌توان انجام داد، در تغییر تعداد تکرار و روش حلگر خلاصه می‌شود.

SGD(Stochastic Gradiant Descent)

دومین الگوریتم استفاده شده، SGD بود. برای انجام طبقه‌بندی با استفاده از الگوریتم SGD، از تابع SGDClassifier از کتابخانه scikit-learn استفاده کردیم. موارد زیر برای تمامی مدل‌های SGD یکسان بودند:

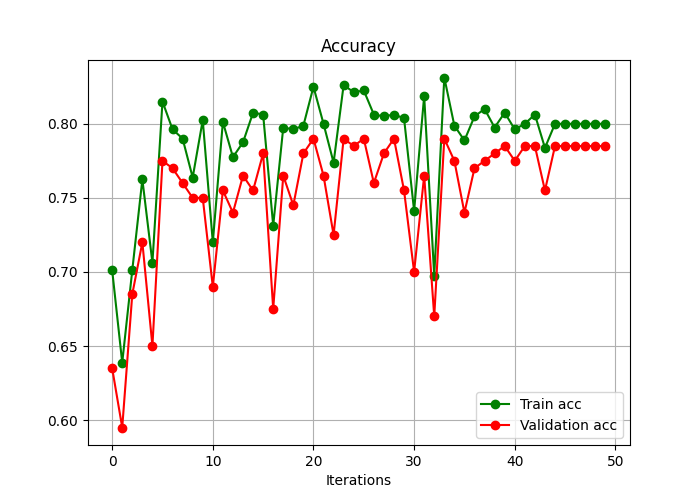
penalty = 'l2'

fit\_intercept = True

random\_state = 69

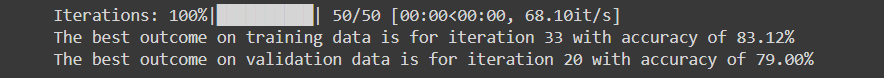
فراپارامترهایی که برای این الگوریتم بهینه‌سازی شدند به‌صورت مفصل در ادامه توضیح داده خواهند شد.

**تعداد تکرار:** برای آموزش این مدل، از 1 تا 50 بار تکرار استفاده کردیم که روند آن در شکل 7 نشان داده شده است.



شکل 7نمودار دقت مدل نسبت به تعداد تکرار ها

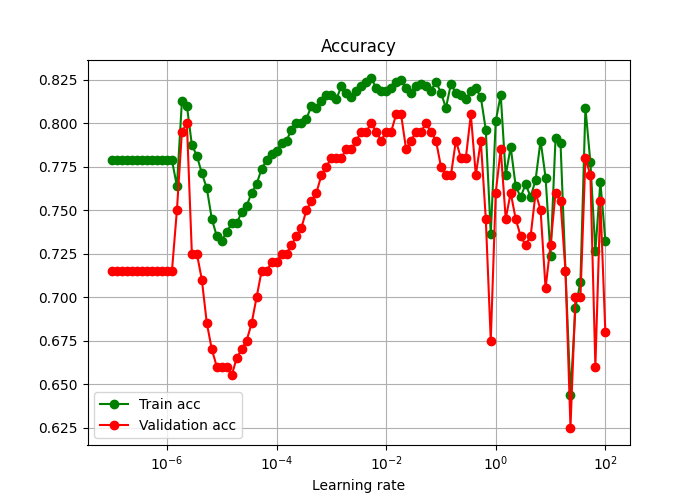
در شکل 7 نمودار دقت بر حسب تعداد تکرار، برای مجموعه داده آموزش و صحه‌سنجی به نمایش درآمده است.



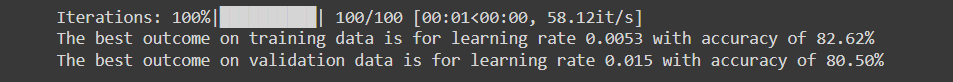
شکل 8 بهترین دقت و تکرار متناظر با آن

در شکل 8 نیز بهترین دقت بدست آمده و شماره تکرار متناظر با آن، برای هر دو مجموعه داده آموزش و صحه‌سنجی به نمایش در آمده است.

**نرخ یادگیری:** در این حالت تعداد تکرار را ثابت در نظر گرفتیم و برای نرخ‌های یادگیری مختلف، عملکرد مدل را بررسی نمودیم.



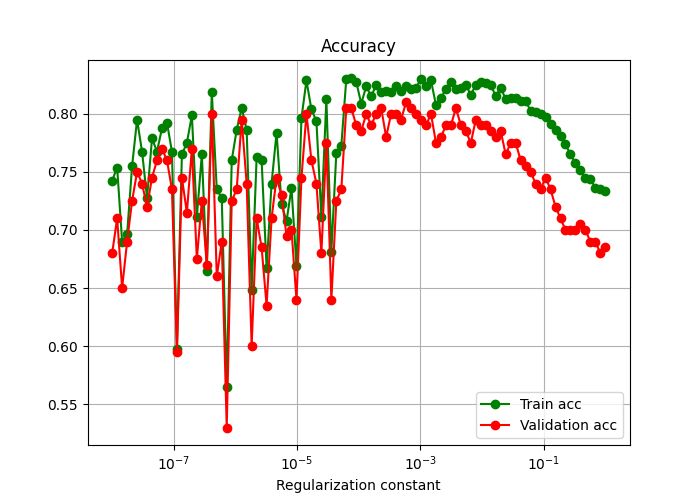
شکل 9نمودار دقت نسبت به نرخ یادگیری



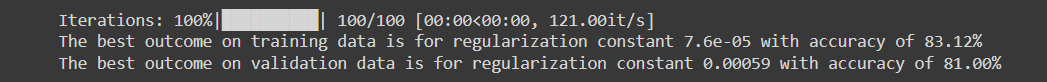
شکل 10 بهترین دقت بدست آمده و نرخ یادگیری متناظر با آن

شکل 10 بهترین نرخ یادگیری برای مجموعه‌های داده آموزش و صحه‌سنجی و دقت متناظر با آن را به نمایش گذاشته است.

**ضریب تنظیم‌سازی:** این ضریبی است که مقدار penalty را کنترل می‌کند. حال برای ضرایب از تا 1 ، عملکرد سیستم را بررسی می کنیم.



شکل 11نمودار تغییر دقت بر حسب ضریب regularization



شکل 12بهترین دقت بدست آمده و ضریب regularization متناسب با آن

Perceptron

سومین الگوریتم استفاده شده، الگوریتم perceptron بود که با استفاده از تابع Perceptron از کتابخانه scikit-learn به دست آمد. پارامترهای ثابت برای تمامی مدل‌ها عبارت‌اند از:

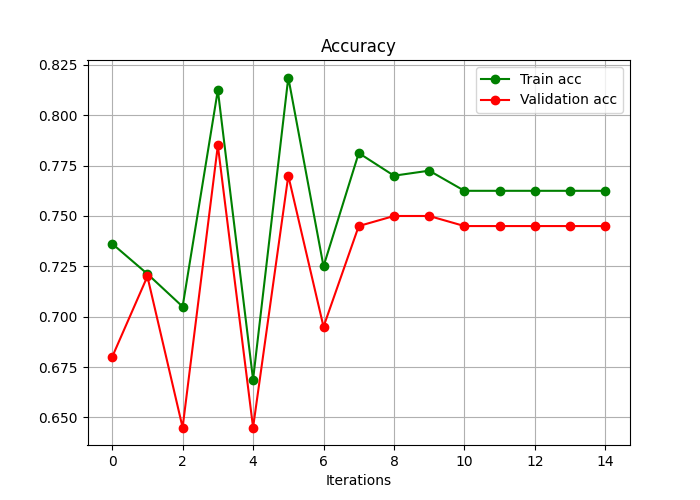
penalty = 'l2'

fit\_intercept = True

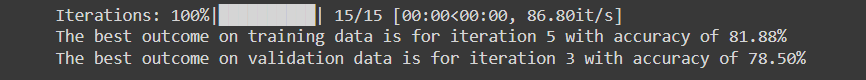
random\_state = 69

فراپارامترهایی که برای این الگوریتم بهینه‌سازی شدند به‌صورت مفصل در ادامه توضیح داده خواهند شد.

**تعداد تکرار:** برای آموزش این مدل، از 1 15تا بار تکرار استفاده کردیم که روند آن در شکل 13 نشان داده شده است.

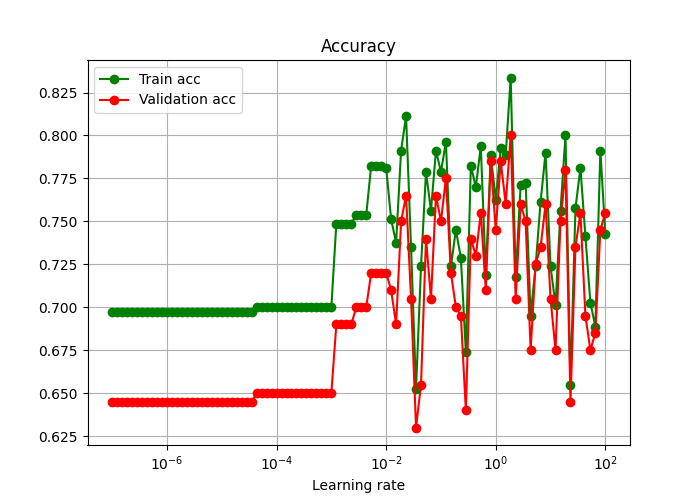


شکل 13نمودار دقت بر حسب تعداد تکرار

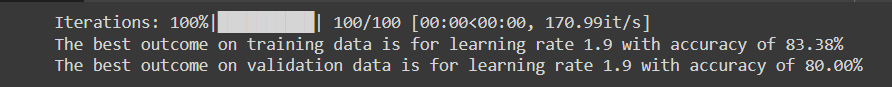


شکل 14بهترین دقت بدست آمده و تعداد تکرار متناظر با آن

**نرخ یادگیری:** در این حالت تعداد تکرار را ثابت در نظر گرفتیم و برای نرخ‌های یادگیری مختلف، عملکرد مدل را بررسی نمودیم.

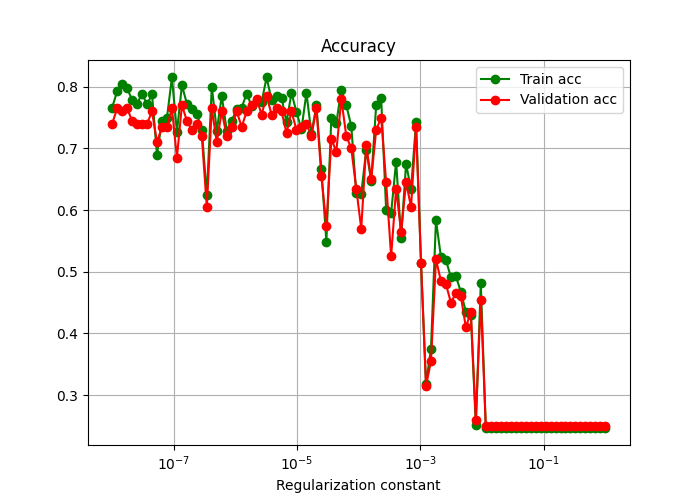


شکل 15نمودار دقت بر حسب نرخ یادگیری

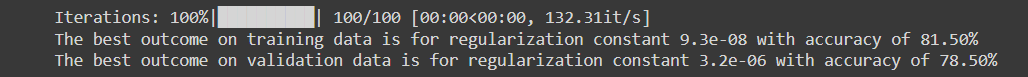


شکل 16بهترین دقت و نرخ یادگیری متناظر با آن

**ضریب تنظیم‌سازی:** این ضریبی است که مقدار penalty را کنترل می‌کند. حال برای ضرایب از تا 1 ، عملکرد سیستم را بررسی می کنیم.



شکل 17نمودار دقت برحسب ضریب regularization

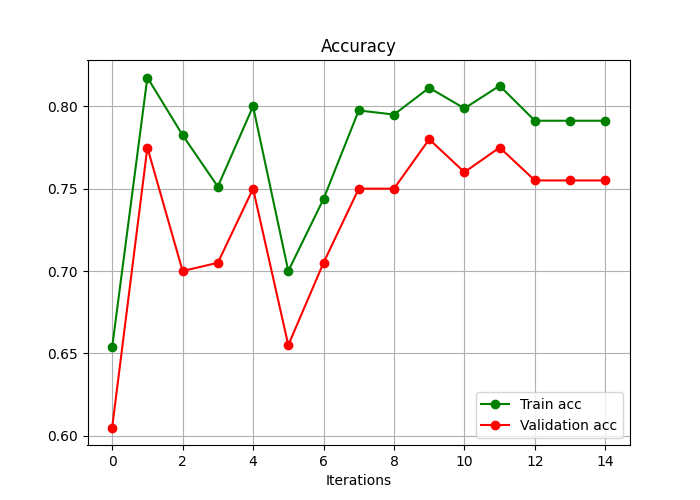


شکل 18بیشترین دقت بدست آمده و ضریب regularization متناظر با آن

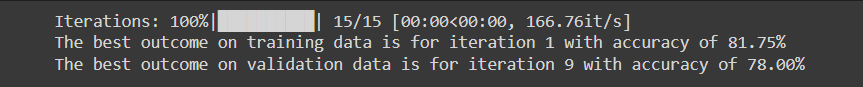
Passive Aggressive Classifier

برای استفاده از این الگوریتم از تابع PassiveAgressiveClassifier موجود در کتابخانه scikit-learn استفاده کردیم.

**تعداد تکرار:** برای بررسی این فراپارامتر از 1 تا 15 تکرار را بررسی کردیم.



شکل 19نمودار دقت بر حسب تعداد تکرار



شکل 20بیشترین دقت بدست آمده و تعداد تکرار متناظر با آن

تابع استفاده شده برای این الگوریتم از هیچ نرخ یادگیری‌ای استفاده نمی‌کرد و نمی‌توانستیم که عملکرد سیستم را برای نرخ‌های یادگیری متفاوت بررسی کنیم.

تکنیک‌های بهبود مدل

تمرکز بخش‌های قبلی برو روی این بود که بهترین تعداد تکرار برای الگوریتم یا بهترین نرخ یادگیری را بیابند. اما در علم یادگیری ماشین، کارهای بسیار مختلف دیگری برای بهبود عملکرد مدل پیشنهاد می‌شود که بسیاری از آن‌ها صرفاً به تنظیم فراپارامترهای خود مدل نیز محدود نمی‌شوند.

**تکنیک‌هایی که به‌صورت کلی می‌توان از آن‌ها برای بهبود عملکرد مدل استفاده کرد:**

* اولین و مهم‌ترین تکنیکی که برای بهبود عملکرد مدل پیشنهاد می‌شود، نرمال‌سازی داده ورودی به ماشین است. این بدان معنا است که بازه تغییرات داده را به صفر تا 1 یا -1 تا 1 انتقال دهیم. این کار برای تمامی ویژگی‌ها انجام می‌شود. این کار به این علت انجام می‌شود که تاثیر تغییرات تمامی ویژگی‌ها به صورت یکسانی باشد.
* در بسیاری از موارد، نسبت تغییرات هر ویژگی را به تغییرات خروجی نهایی را می‌سنجند. با این کار می‌توان متوجه شد که کدام ویژگی‌ها روی خروجی نهایی تأثیر دارند و استفاده از چه ویژگی‌هایی به ایجاد یک مدل بهتر کمک می‌کند. مثلاً اگر با تغییرات متغیر x، خروجی تغییر نکرد، می‌توان آن ویژگی را حذف کرد تا از داده کمتری استفاده شود و سرعت محاسبات افزایش یابد.
* از جمله مهم‌ترین تکنیکی که بر روی داده ورودی انجام می‌شود، اصلاح عدم تعادل درداده است. این تکنیک برای مسائل طبقه‌بندی انجام می‌شود. هنگامی که از کلاس‌های مختلف خروجی، تعداد یکسانی داده در اختیار نداریم، ایجاد یک مدل یادگیری ماشین در این حالت باعث می‌شود که طبقه‌بندی‌ای که مدل انجام می‌دهد، دارای یک سوگیری[[21]](#footnote-21) در طبقه‌بندی به نفع داده آموزش با تعداد بیشتر خواهد بود. برای جلوگیری از این مشکل، از روش‌های اصلاح عدم تعادل استفاده می‌شود.
* استفاده از توابع هزینه مناسب می‌توان بسیار به رسیدن به نتیجه مطلوب کمک کند. اگر تابع هزینه به صورتی تعریف شود که کمینه آن به معنای نزدیک‌تر بودن پاسخ مدل به نتیجه مطلوب باشد، قطعاً عملکرد مدل افزایش پیدا خواهد کرد.
* می‌توان گفت که مهم‌ترین بخش یادگیری ماشین، الگوریتمی است که برای بهینه‌سازی وزن‌های مدل استفاده می‌کنیم. پس با انتخاب الگوریتم بهینه‌سازی مناسب، هم به مدل بهتری دست پیدا خواهیم کرد و زمان کمتری برای آموزش مدل می‌گذاریم.
* ما باید باتوجه‌به تعداد داده‌های آموزش، و تعداد ویژگی‌های موجود، یک پیچیدگی مناسب برای مدل انتخاب کنیم. هر چه ویژگی‌های موجود درداده بیشتر باشد، باید مدل بزرگ‌تری انتخاب شود تا قادر باشد که همه آن‌ها را بررسی کرده و نتیجه مناسب را تحویل دهد. اگر تعداد داده آموزش بیشتری در اختیار داشته باشیم، مدل‌های کوچک قابلیت این را نخواهند داشت که از ویژگی‌های موجود در تمامی داده‌ها را برای ساخت ماشین استفاده کنند و هر مدلی می‌تواند تعداد محدودی داده را، متناسب با پیچیدگی خود مدل، برای ساخت مدل استفاده کند. داده کم باعث می‌شود که مدل به‌صورت ناقص آموزش داده شود و داده زیاد هم بیشتر از ظرفیت مدل بوده و تمامی ویژگی‌های موجود درداده استخراج نمی‌شود
* Regularization یک از مهم‌ترین مباحثی است که برای بهبود عملکرد مدل یادگیری ماشین استفاده می‌شود. این روش عموماً برای کاهش واریانس[[22]](#footnote-22) اعمال می‌شوند. Regularization تکنیک‌های بسیار متنوعی را شامل می‌شود. مثل اعمال یک مقدار خطا اضافه به تابع هزینه برای کاهش وزن‌ها.

**تکنیک‌های استفاده شده در این مسئله:**

1. نرمالایز کردن داده ورودی
2. آموزش داده روی مدل‌های مختلف
3. اعمال عبارت خطا اضافی (regularization) به تابع هزینه (در مدل‌هایی که قابلیت را داشتند)
4. استفاده از حلگرهای مختلف و مقایسه نتایج

به دلیل کم‌بودن داده یا متعادل بودن کلاس‌ها و... نمی‌توانستیم از دیگر تکنیک‌های بهبود مدل استفاده کنیم.

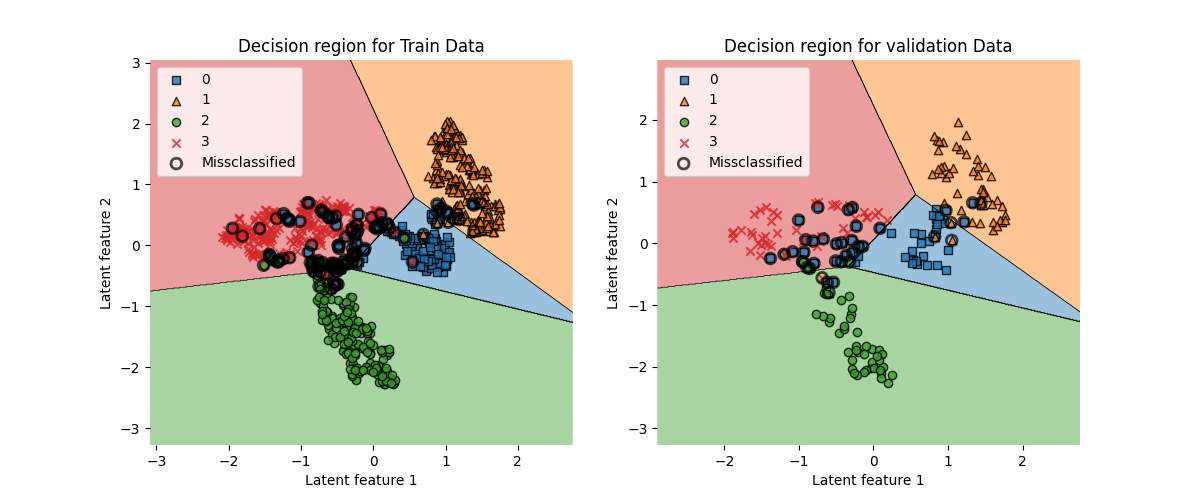
1-4 نمایش مرز و نواحی تصمیم گیری

داده‌های تولید شده دارای 3 ویژگی هستند. برای نمایش مرزها و نواحی تصمیم‌گیری می‌بایست که تعداد ویژگی داده‌ها را به 2 بعد کاهش دهیم. برای کاهش ابعاد داده‌ها از الگوریتم t-SNE استفاده کردیم. کاهش ابعاد با استفاده از تابع TSNE از کتابخانه scikit-learn انجام شد.

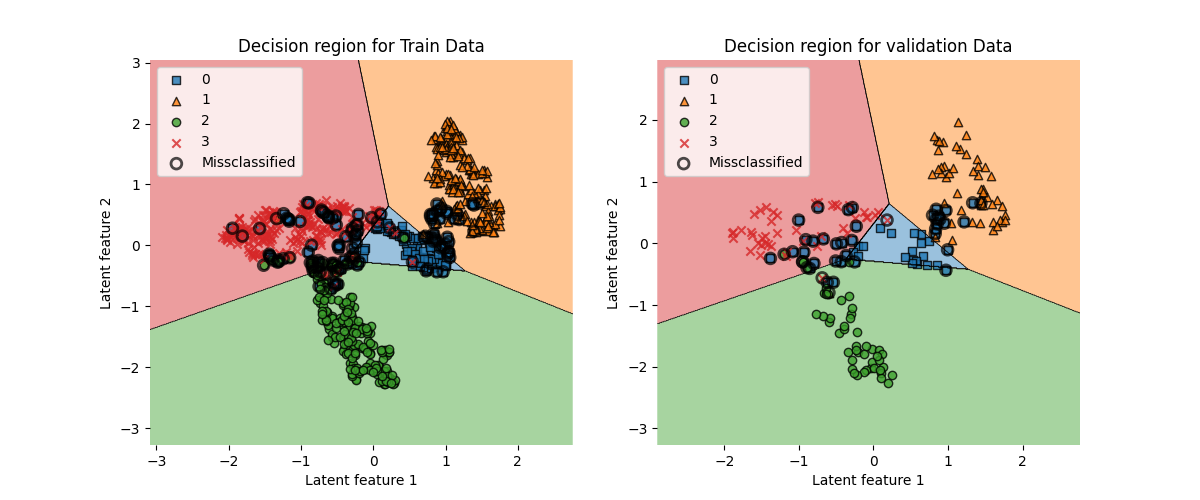
پس از کاهش ابعاد داده‌ها، مدل‌های یادگیری ماشین بار دیگر روی داده‌های جدید که دارای 2 ویژگی بودند، آموزش داده شدند. لازم به ذکر است که پس از کاهش ابعاد داده، دوباره با استفاده از دستور train\_test\_split، بار دیگر داده‌ها را به دسته‌های آموزش و صحه‌سنجی تقسیم کردیم.

پس از آموزش دوباره مدل‌ها، با استفاده از دستور plot\_decision\_regions از کتابخانه mlxtend، مرزها و نواحی تصمیم‌گیری به نمایش درآمدند.

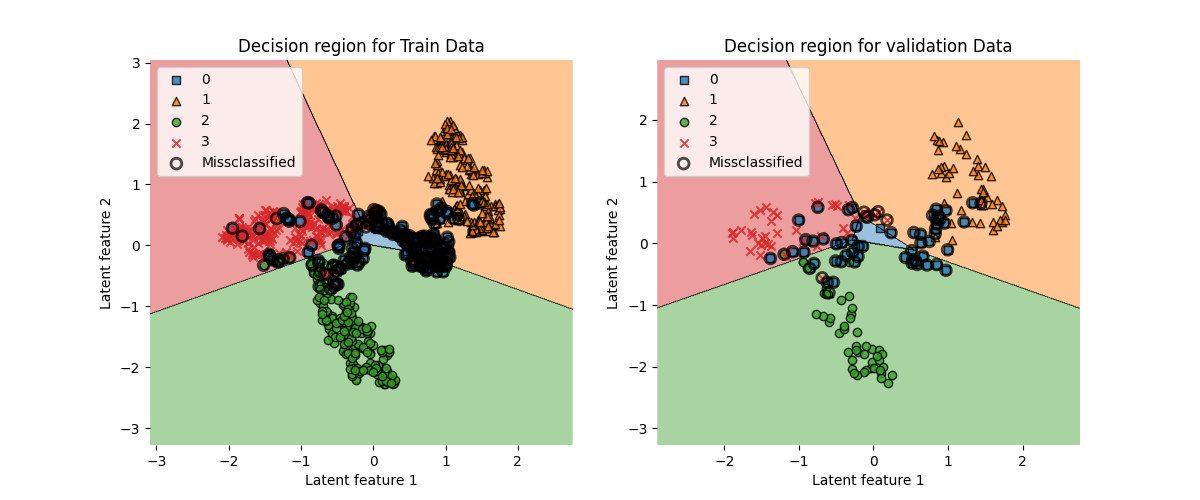
در ادامه نمودارهای به وجود آمده را برای هر مدل به نمایش می‌گذاریم.



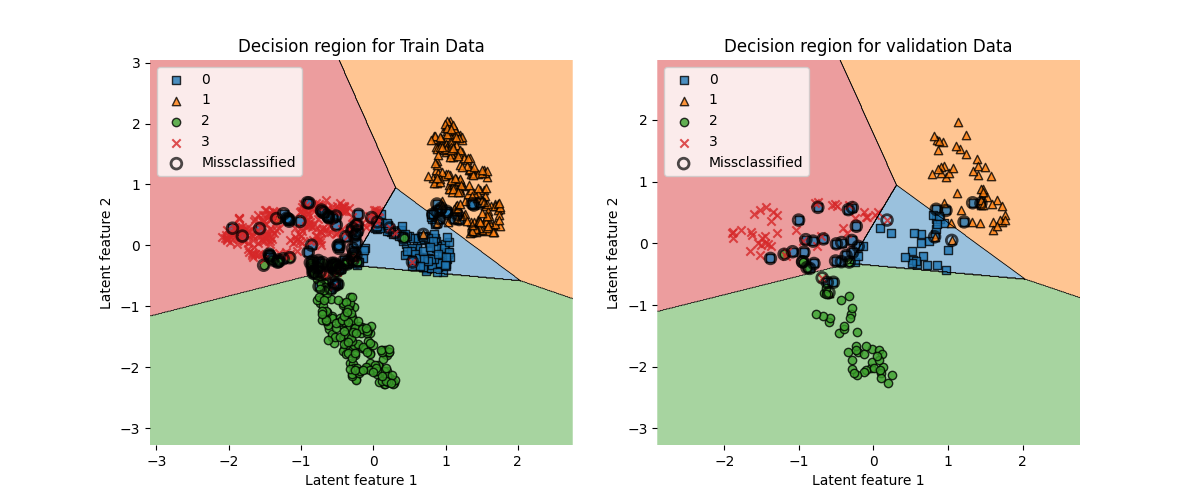
شکل 21نواحی تصمیم گیری برای مدل Logisic Regression



شکل 22نواحی تصمیم‌گیری برای مدل SDG



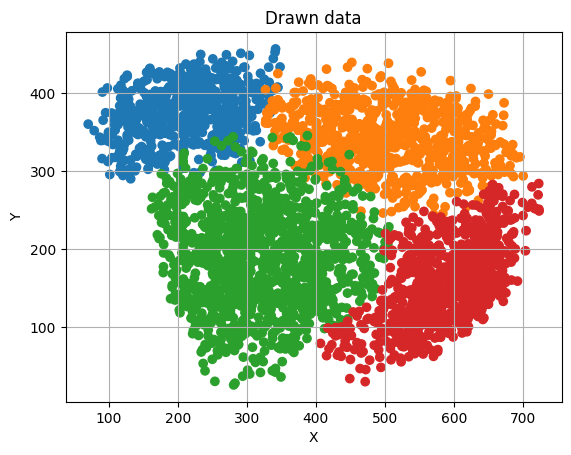
شکل 23 نواحی تصمیم‌گیری برای مدل Perceptron



شکل 24 نواحی تصمیم‌گیری برای مدل Passive Aggressive Classifier

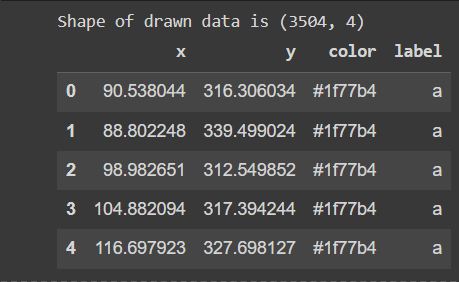
1-4 استفاده از ابزار draw data

در اولین قدم با استفاده از کتابخانه rawdata و با استفاده از دستور ScatterWidget، یک مجموعه داده دلخواه که شامل 4 کلاس می باشند را ایجاد می کنیم.

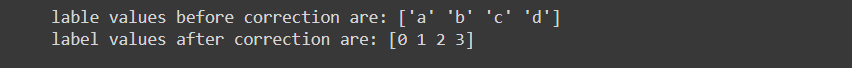


شکل 25پخش نقاط داده ایجاد شده

شکل 25 پخش داده ایجاد شده را بر روی صفحه دوبعدی به نمایش گذاشته است. داده ایجاد شده را در یک دیتافریم پاندا[[23]](#footnote-23) ذخیره می کنیم. شکل 26 دیتافریم ایجاد شده را به نمایش گذاشته است. در اولین قدم باید مقادیر رشته[[24]](#footnote-24) موجود در ستون برچسب[[25]](#footnote-25) را به مقادیر عدد تبدیل کنیم.



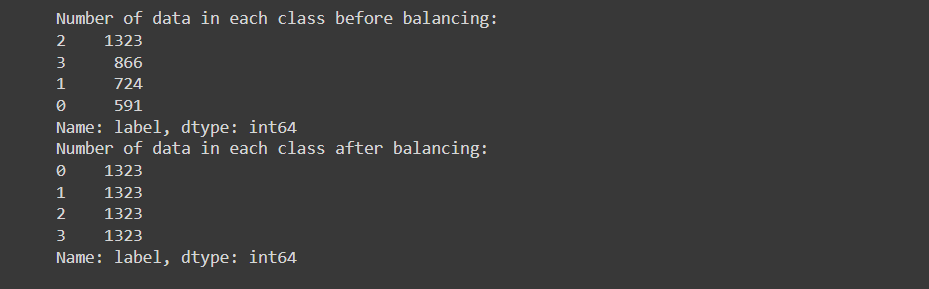
شکل 26 دیتافریم ایجاد شده از rawdata



شکل 27تبدیل مقادیر رشته‌ای به مقادیر عددی

شکل 27 نمایش می دهد که مقادیر رشته‌ای را به مقادیر عددی تبدیل کردیم و می توانیم محاسبات را ادامه دهیم.

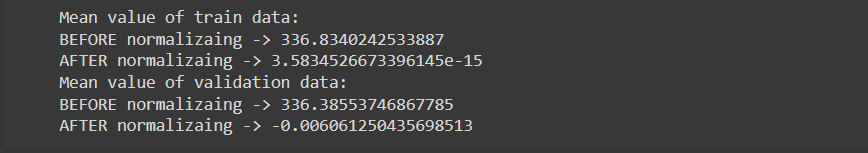
همانطور که از شکل 25 و 28 مشخص می باشد، داده ها دارای عدم تعادل می‌باشند. پیشتر درباره ضرر عدم تعادل در داده صحبت شد. برای رفع این عیب در دیتافریم، از روش SMOTE[[26]](#footnote-26) استفاده می کنیم.



شکل 28تعداد داده ها قبل و بعد از متعادل کردن مجموعه داده

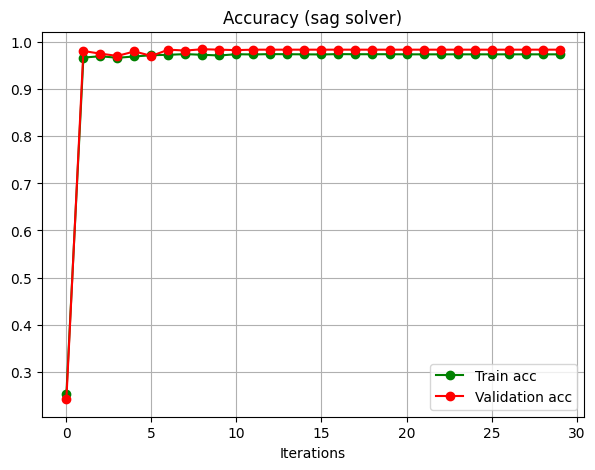
شکل 28 نمایش می دهد که تعداد داده ها در همه کلاس ها برابر شده است.

بعد از نرمالایز کردن داده و تقسیم آن به مجموعه های آموزش و ارزیابی[[27]](#footnote-27)، مدل هایی بخش 3 را دوباره اجرا می کنیم و نتابج بدست آمده را نمایش می دهیم.

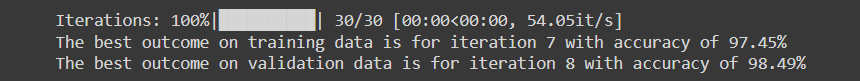


شکل 29 میانگین مجموعه داده های آموزش و ارزیابی قبل و بعد از نرمال کردن

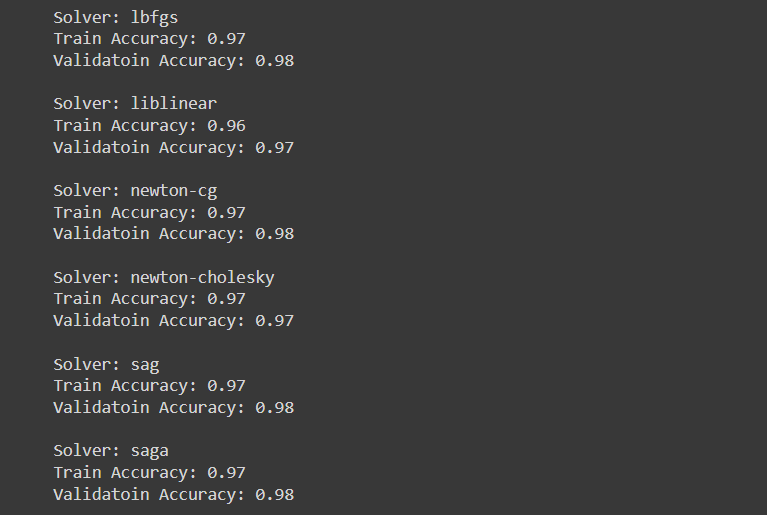
Logistic Regression



شکل 30 نمودار دقت بر حسب تکرار

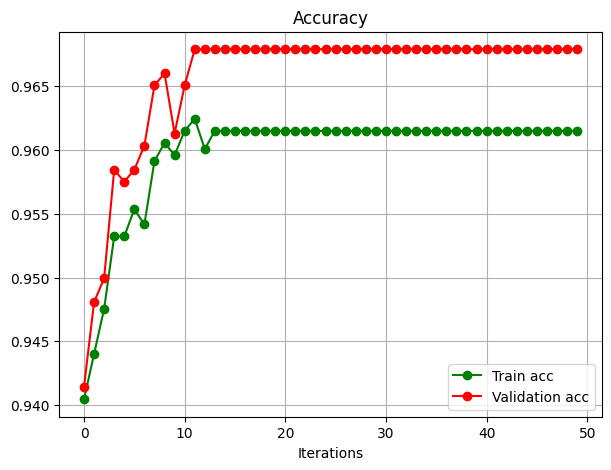


شکل 31 بهترین دقت و تکرار متناظر با آن

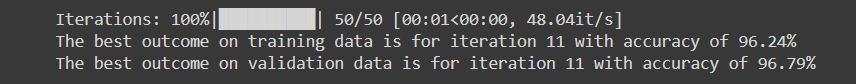


شکل 32 دقت های بدست آمده برای حلگر های مختلف

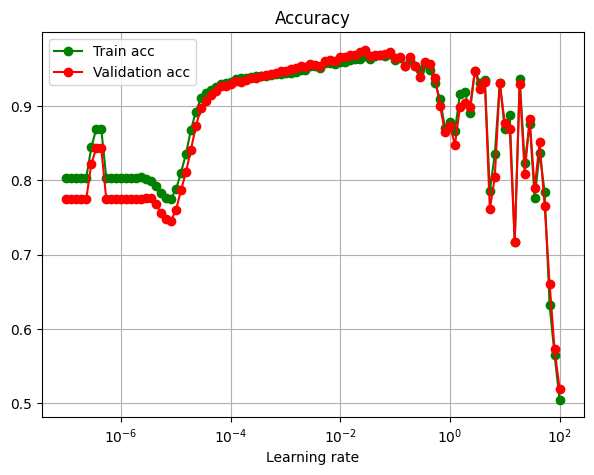
SGD



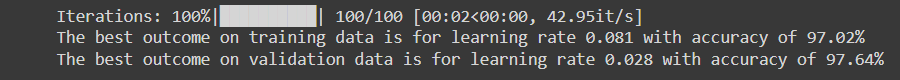
شکل 33نمودار دقت بر حسب تکرار



شکل 34 بهترین دقت بدست آمده و تکرار متناظر با آن

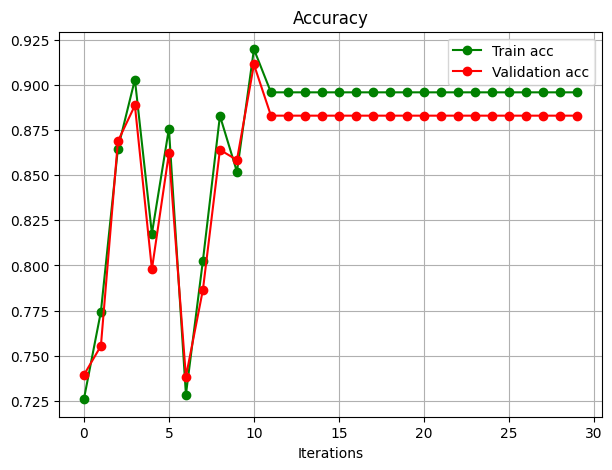


شکل 35 نمودار دقت برحسب نرخ یادگیری

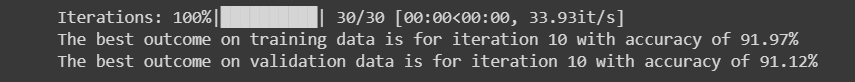


شکل 36 بهترین دقت بدست آمده و نرخ یادگیری متناظر با آن

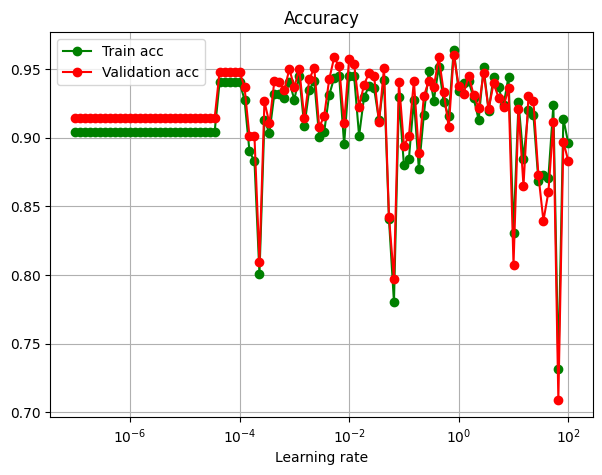
Perceptron



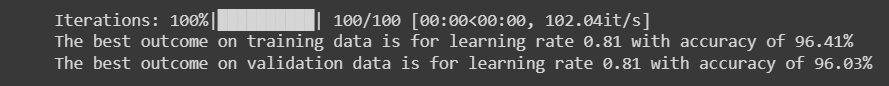
شکل 37نمودار دقت بدست آمده برحسب تکرار



شکل 38 بهترین دقت و تکرار متناظر با آن

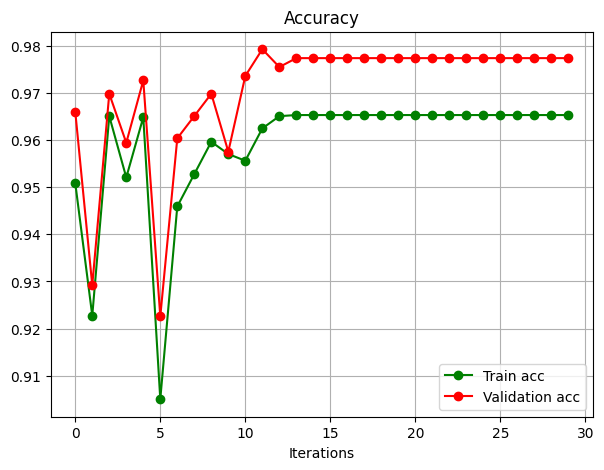


شکل 39نمودار دقت برحسب نرخ یادگیری

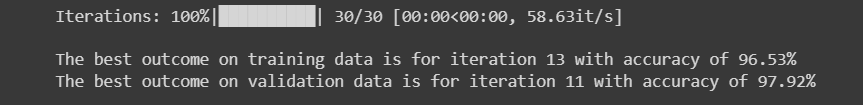


شکل 40 بهترین دقت بدست آمده و نرخ یادگیری متناظر با آن

Passive Aggressive Classifier

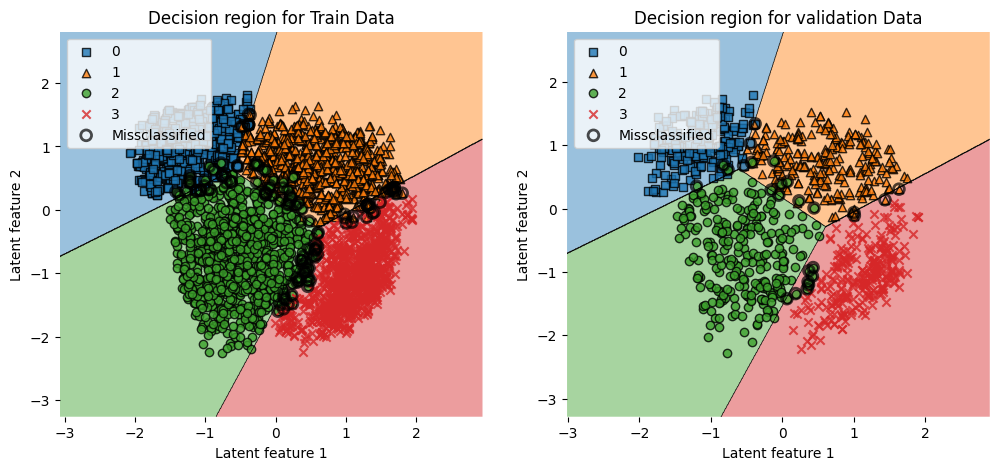


شکل 41 نمودار دقت برحسب تکرار

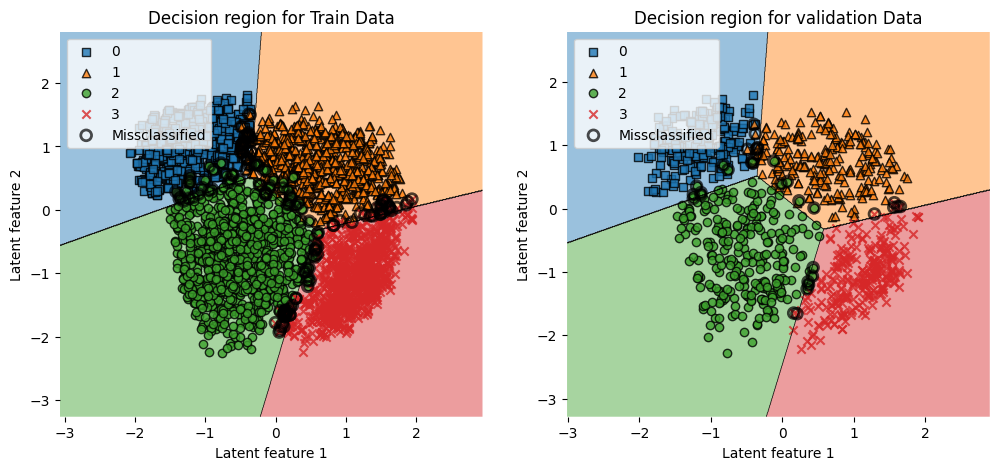


شکل 42بهترین دقت بدست آمده و تکرار متناظر با آن

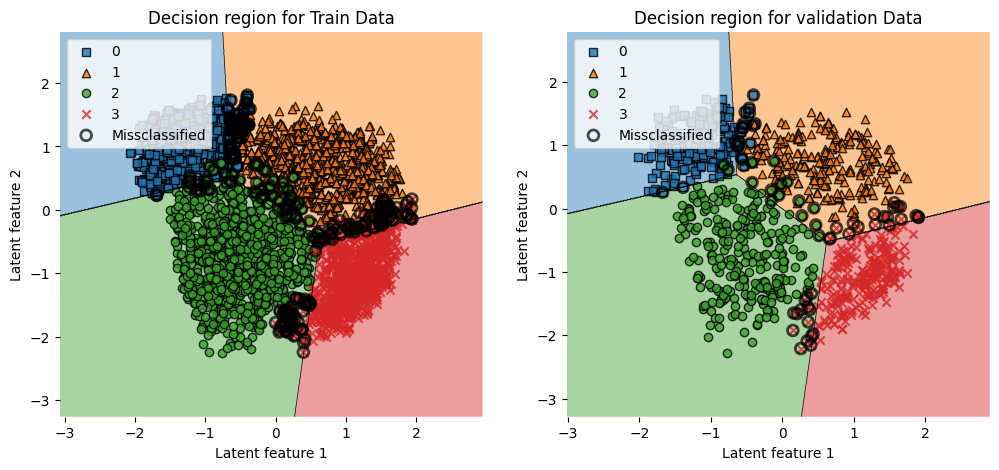
نمایش مرز و نواحی تصمیم‌گیری



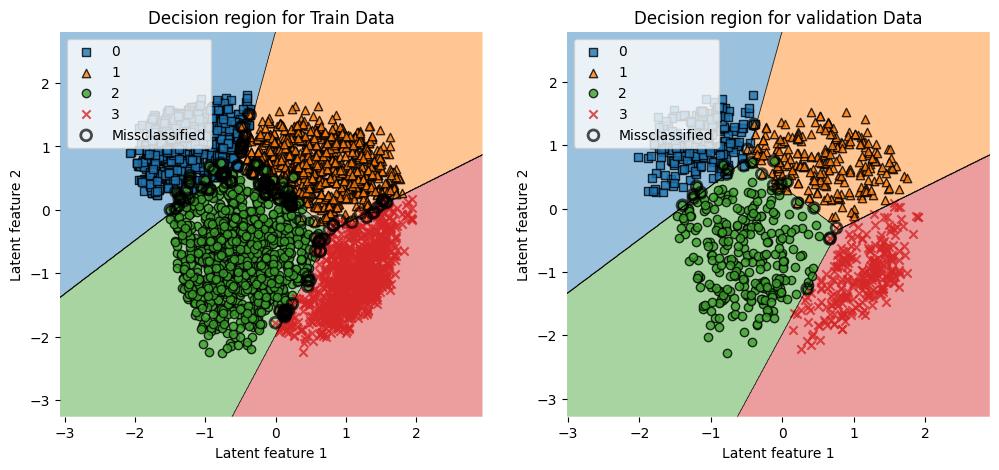
شکل 43نواحی تصمیم‌گیری برای مدل Linear Rgression



شکل 44 نواحی تصمیم‌گیری برای مدل SGD



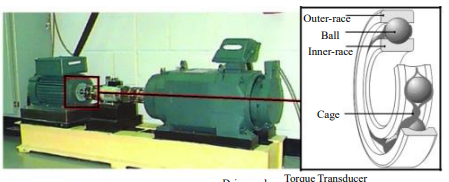
شکل 45 نواحی تصمیم‌گیری برای مدل Perceptropn



شکل 46 نواحی تصمیم‌گیری برای مدل Passive Agressive Classifier

سوال 2

2-1 CWRT dataset



شکل 47پلتفرم آزمایش یاتاقان ها برای استخراج داده[1]

اهداف

این مجموعه داده توسط دانشگاه Case Western Reserve ایجاد شده است. این مجموعه داده، یک دیتاست متن-باز است که به عنوان اساس و مرجع اصلی بسیاری از مدل های یادگیری ماشین، که در حوزه تشخیص عیوب یاتاقان کار می کنند، است[1]. هدف از ایجاد این مجموعه داده، کمک به ایجاد مدل های تشخیص و پیشبینی عیب در یاتاقان ها می باشد.

ویژگی ها و حالات مختلف

این مجموعه متشکل از 161 رکورد است که در چهار کلاس طبقه‌بندی شده اند:

1. Normal-baseline با نرخ داده برداری 48000 نمونه در ثانیه
2. عیب drive-end با نرخ داده برداری 48000 نمونه بر ثانیه
3. عیب drive-end با نرخ نمونه برداری 12000 نمونه بر ثانیه
4. عیب fan-end با نرخ نمونه برداری 12000 نمونه بر ثانیه

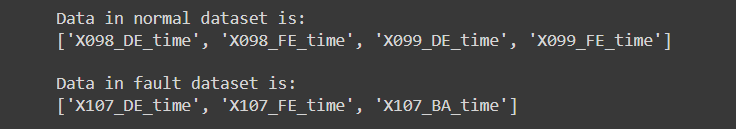
هر یک از کلاس های بیان شده در ادامه به دیتاست هایی با عیوب مختلف تقسیم می شود. این عیب ها می توانند شامل عیوب زیر باشند:

* عیب ساچمه
* عیب حلقه داخلی
* عیب حلقه خارجی؛ تقسیم‌بندی براساس موقعیت بار وارد شده:
  + متمرکز[[28]](#footnote-28)
  + قائم
  + متقابل[[29]](#footnote-29) [1]

دو رقم آخر شماره دانشجویی 54 می باشد که باقی مانده آن بر 4 برابر 2 است. بنابراین فایل های Normal-2 و IR007-2 را دریافت کردیم.

2-2 کار با دیتاست

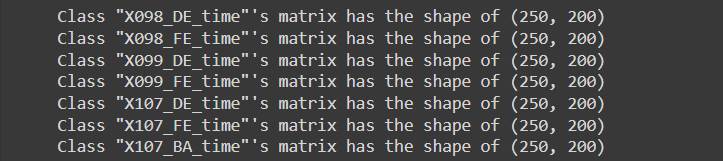
فایل های دانلود شده به فرمت .mat هستند. برای این که این فایل ها توسط پایتون خوانده و ذخیره شوند از دستور loadmat موجود در کتابخانه scipy استفاده کردیم.



شکل 48داده موجود در دیتاست ها

2-2-1 استخراج داده

شکل 48 تمامی حالت‌های موجود در دیتاست را به نمایش گذاشته است. در کل 7 حالت داریم که می بایست برای هر کدام یک ماتریس MxN تشکیل دهیم. برای این کار از دو حلقهfor تودرتو استفاده کردیم و تمامی ماتریس ها را در یک دیکشنری به نام matrices ذخیره نمودیم. M و N به صورت زیر در نظر گرفته شدند.



شکل 49 شکل ماتریس های ایجاد شده برای هر کلاس

اما از آنجایی که بای برای هر یک از دیتاست های نرمال و معیوب، یک کلاس داشته باشیم، یکی از کلاس های موجود را انتخاب کرده و محاسبات را ادامه می دهیم.(X\*\*-DE-time را انتخاب می کنیم)

2-2-2 استخراج ویژگی

تشخیص عیب یا هر کار دیگر مدل های یادگیری ماشین، با استفاده از داده انجام می شود. اما داده خام که مثلا توسط سنسور یا دوربین یا ... دارای اطلاعات اضافی و به‌درد نخور زیادی است. در واقع داده خام دارای نویز می باشد. نویز داده باعث بوجود آمدن اختلال در تصمیم‌گیری ماشین می شود؛ زیرا نمی تواند با کنار هم گذاشتن تمام اطلاعات موجود تصمیم درستی بگیری. برای این که بهبود عملکرد مدل، سعی می کنیم که اطلاعات اضافی و نویز را از داده ورودی حذف کنیم. یعنی از میان تمامی اطلاعات موجود، باید یک سری ویژگی استخراج کنیم که ماشین با استفاده از آن ها عملیات تصمیم گیری را انجام دهد.

تمامی ویژگی های موجود را محاسبه کردیم. برای استخراج ویژگی ها، بعضی از ویژگی ها را که در کتابخانه های numpy یا scipy، توابع از پیش تعریف شده داشتند، از آن توابع استفاده کردیم و برای محاسبه دیگر ویژگی ها، از فرمول های آنان استفاده کردیم. سپس برای هر کدام از ماتریس های نرمال و عیب، یک دیتاست جداگانه ایجاد کردیم. برای هر یک از دیتاست ها یک ویژگی جدید به نام "label"[[30]](#footnote-30) ایجاد کردیم و برای داده های نرمال عدد 0 و برای داده های عیب عدد 1 بود. سپس دو دیتاست را در راستای محور 0 یا x به هم اضافه کردیم (یعنی سطر ها را به هم اضافه کردیم).

2-2-3 برزدن

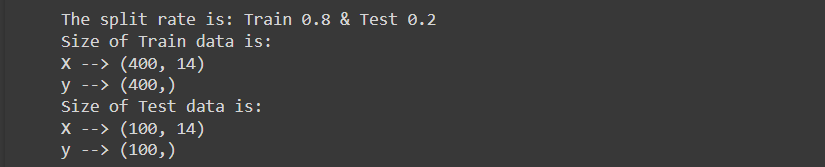
وظیفه یک مدل یادگیری ماشین در بحث طبقه بندی این است که یک کلاس مناسب برای ورودی خود اختصاص دهد. با استفاده از داده های آموزش، مدل رابطه ای بین ورودی و خروجی پیدا میکند. اما تمامی این یادگیری بر مبنای داده های آمورش است. یعنی توانایی طبقه‌بندی برای داده هایی است که مثل داده های آموزش باشند. به عبارت دیگر برای انجام آموزش و ارزیابی مدل ایجاد شده، نیاز داریم که داده ها پخش یکسانی داشته باشند.

تشکیل دیتاست جدید بدین صورت بود که ابتدا داده های نرمال قرار داده شدند و سپس داده های عیب. اگر تقسیم بندی آموزش و ارزیابی را در این حالت انجام دهیم، مثلا 80 درصد ابتدایی را به آموزش و بقیه داده ها را به ارزیابی اختصاص دهیم، داده های ازریابی فقط دارای کلاس عیب خواهند بود. از طرفی داده های آموزش با این که هم داده های نرمال و هم داده های عیب را دارند ولی تعداد داده های هر کلاس یکسان نیست و عدم تعادل در داده آموزش وجود خواهد داشت. یعنی اگر برزدن را انجام ندهیم و داده های آموزش و ارزیابی را از هم جدا کنیم، پخش داده ها یکسان نبوده و نتایج مدل غیرقابل اعتماد خواهند بود.

برای برزدن و تقسیم کردن داده می‌توان دو روش را در پیش گرفت:

1. ابتدا دیتاست را بر بزنیم و سپس داده را بدون برزدن جدا کنیم
2. دیتاست برنخورد ولی هنگام تقسیم به دیتاست های آموزش و ارزیابی، shuffle=True قرار دهیم.

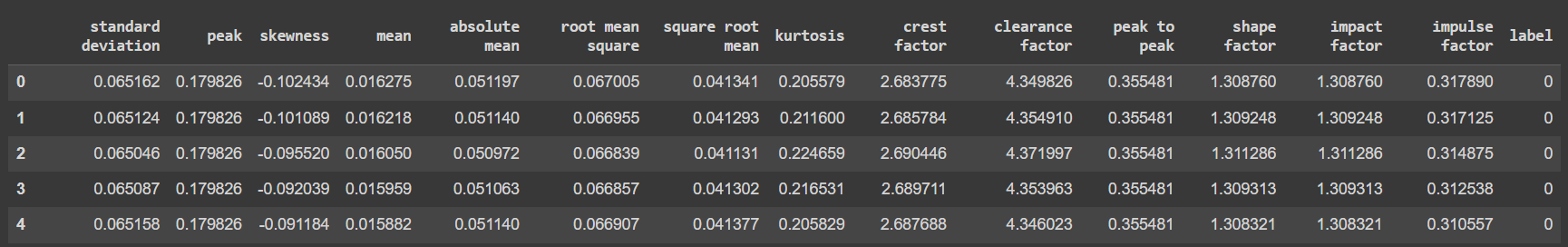
برای برزدن از روش دوم استفاده می کنیم. برای برزدن از تابع train\_test\_split از کتابخانه scikit-learn استفاده می کنیم.



شکل 50 تقسیم بندی داده ها

همانطور که در شکل 50 مشاهده می شود، داده ها با نسبت 0.8 و 0.2 تقسیم شده اند و اندازه تمامی متغییر های ایجاد شده به نمایش درآمده است.

2-2-4 نرمال سازی داده ها



شکل 51 پنج سطر اول دیتاست

شکل 51 پنج سطر اول دیتاست را به نمایش درآورده است. اگر به مقادیر موجود در ویژگی های Root Mean Square و Clearance Factor توجه کنیم، متوجه می شویم که این مقادیر، هم از نظر بازه خود مقادیر و از نظر بازه تغییراتشان با هم متفاوت هستند. در نتیجه ممکن است که تغییرات یکی، از تغییرات دیگری تاثیر بیشتری داشته باشد(اگر وزن ها برای هر دو ویژگی یکسان باشند). برای این که تأثیر همه ویژگی ها برای محاسبات ماشین یکسان باشد، بازه تغییرات تمامی ویژگی ها را یکسان می کنند؛ به طور معمول بازه همه متغییر ها به [0,1] یا [-1,1] انتقال داده می شود.

دو روش استفاده شده برای نرمال سازی داده ورودی:

* Standardization:   
  در روش بالا، مقدار میانگین از تمامی مقادیر کم شده و سپس بر انحراف معیار تقسیم می شوند. با این روش اعداد بین -1 و 1 قرار می گیرند.
* MinMaxScaler:   
  در این روش با یک تبدیل، کمترین مقدار 0 میشود و با تقسیم تمامی اعداد بر بازه اعداد، همه مقادیر بین 0 و 1 قرار می گیرند.

در این بخش از روش Standardization استفاده شد. با استفاده از scaler.fit(x\_train)، پارمترهای مقیاسگر[[31]](#footnote-31) با توجه به داده های آموزش تنظیم می شوند و سپس از آن مقادیر استفاده شده تا داده های آموزش و ارزیابی نرمالسازی شوند. نرمال سازی کردن داده های ارزیابی به این دلیل است که پخش داده های ارزیابی هم مثل داده های آموزش شود. همانند برزدن داده ها، هدف این بود که پخش داده های ارزیابی و آموزش یکسان باشد، در نرمال سازی هم برای یکسان کردن پخش داده ها استفاده می شود.

2-3 ایجاد مدل طبقه‌بندی از صفر

از آنجایی که باید تمامی محاسبات را به صورت از 0 تا صد توسط کد انجام دهیم و ریاضیات را پیاده‌سازی کنیم، از کتابخانه numpy استفاده می کنیم. مدل های استفاده شده به صورت زیر می باشند:

* **مدل یادگیری ماشین:** Logistic Regression
* **تابع هزینه:**
* **تابع یادگیری:**

در ادامه به توضیح نحوه پیاده‌سازی مدل یادگیری ماشین می پردازیم.

به صورت خلاصه، مدل خام به صورت یک کلاس در پایتون تعریف شد و با استفاده از آن آموزش صورت گرفت.

تعریف کلاس و مقادیر اولیه

class LogisticRegression:

    def \_\_init\_\_(self, n\_iter=10, learning\_rate=0.01, random\_state=None):

        self.n\_iter = n\_iter

        self.eta = learning\_rate

        np.random.seed(random\_state)

متد برای مقداردهی اولیه وزن ها

    def \_weight\_init(self):

        self.w = np.random.rand(14)\*0

        self.b = np.random.rand()

متد پیاده‎سازی الگوریتم یادگیری (بروزرسانی وزن ها)

    def \_update(self,x,E):

        dj = np.dot(x.T,E)

        self.w += self.eta\*dj

        self.b += self.eta\*np.sum(E)

متد برای تبدیل ورودی مدل به مقدار پیش‌بینی (اختصاص برچسب به ورودی)

    def forward(self,x):

        z = np.dot(x,self.w) + self.b

        a = 1/(1+np.e\*\*(-z))

        y\_hat = np.array([1 if hat>.5 else 0 for hat in a])

        return y\_hat

متد محاسبه خطا و هزینه

    def error(self,predict,true):

        E = true - predict

        e = 1/len(true) \* np.dot(E,E)

        return E,e

متد آموزش، در این بخش با بهره گیری از متدهای قبل، عمل آموزش انجام می شود و خطا هر مرحله نیز ذخیره می شود

    def train(self,x,y):

        self.\_weight\_init()

        self.loss = []

        for iter in range(self.n\_iter):

            y\_hat = self.forward(x)

            E,e = self.error(y\_hat,y)

            self.loss.append(e)

            self.\_update(x,E)

ایجاد مدل و اعمال داده های آموزش به آن

# Train model with data

model = LogisticRegression(

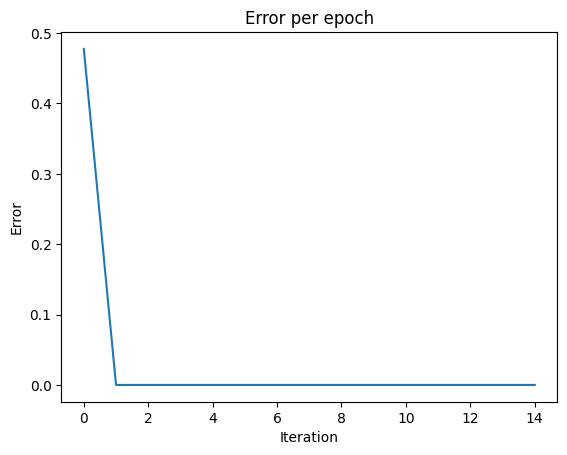
    n\_iter = 15,

    learning\_rate =0.0001,

    random\_state = 25

)

model.train(x\_train\_scaled,y\_train)



شکل 52 نمودار خطا

شکل 52 نمودار خطا مدل به هنگام آموزش را به نمایش می گذارد. همانطور که مشاهده می شود بعد از اولین تکرار مدل، خطا به صفر رسید و مدل توانست بهترین وزن های ممکن را محاسبه کند.

برای ارزیابی عملکرد مدل روی داده تست، از دو معیار دقت[[32]](#footnote-32) و f1-score استفاده کردیم.



شکل 53 معیار های محاسبه شده روی داده تست

شکل 53 عمکلرد مدل را روی داده ارزیابی به نمایش گذاشته است.

برای نظر دادن درباره عملکرد مدل نمی توان تنها با اتکا به نمودار خطا آموزش عمل کرد. باید حتما از یک دسته دیگر داده برای ارزیابی عملکرد مدل استفاده کنیم. مدل با هر بار تکرار و بروزرسانی وزن ها، ماشینی ایجاد می کند که بهتر بتواند داده های آموزش را طبقه‌بندی کند. پس همیشه و با هر بار تکرار، مدل تنیجه بهتری را روی داده آموزش خواهد داشت. اما هدف یادگیری ماشین این نیست که مدلی ایجاد کنیم تا داده های آموزش را به خوبی تشخیص دهد. هدف این است که هر داده ای را به خوبی طبقه‌بندی کند. یعنی به دنبال خاصیب عمومیت‌بخشی[[33]](#footnote-33) هستیم. آموزش بیش‌از‌حد مدل باعث می شود که مدل نویز های موجود در داده آموزش را یاد بگیرد. مدل با آموزش زیاد، داده آموزش را حفظ می کند درحالی که باید رابطه مناسبی بین ورودی و خروجی آموزش پیدا کند. در کل باید علاوه بر کمینه کردن خطا آموزش، خطا ارزیابی را هم کمینه کنیم تا از پدیده over-fitting جلوگیری نماییم.

برای جلوگیری از پدیده over-fitting می توانیم از روش های زیر استفاده کنیم:

* آموزش مدل با تکرار مناسب (تکرار زیاد باعث ایجاد over-fitting و تکرار کم باعث بروز under-fitting می شود.)
* اعمال روش های regularization

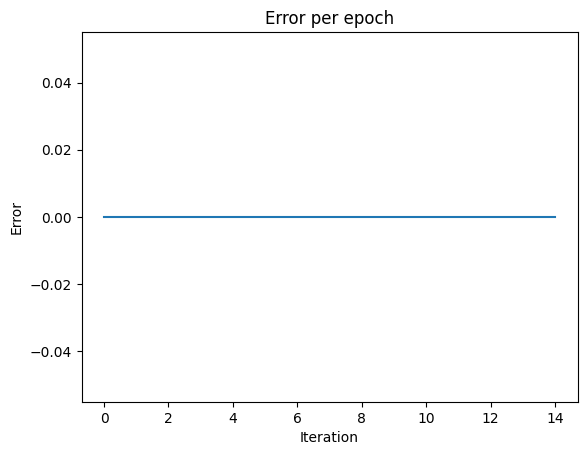
2-4 استفاده از مدل آماده scikit-learn

برای این بخش، از الگوریتم SGD استفاده می کنیم.



شکل 54 نتایج ارزیابی مدل SGD Classifier

همانطور که از شکل 54 مشخص است، تنایج ارزیابی مدل SGD هم همانند مدل خودمان بود.



شکل 55 نمودار خطا بر حسب تعداد تکرار

با توجه به شکل 55، می بینم که خطا در تمامی تکرار ها برابر 0 بوده است.

کلاس SGDClassifier به صورت خودبه‌خود، آرایه ای برای ذخیره سازی خطا در تکرار ها به ما نمی داد. ولی با استفاده از متد partial\_fit()، در هر تکرار خطا را محاسبه کرده و در یک آرایه ذخیره نمودیم. برای پیاده‌‌سازی این ایده، یک زیرکلاس[[34]](#footnote-34) از زیرمجموعه کلاس sklearn.linear\_model.SGDClassifier ایجاد کرده و ویژگی loss\_history را در آن تعریف کردیم. این ویژگی با استفاده از متد partial\_fit() به همراه محاسبات خطا، خطا را در تکرار در خود ذخیره می کرد.

1. Logistic Regression [↑](#footnote-ref-1)
2. Classification [↑](#footnote-ref-2)
3. Activation Function [↑](#footnote-ref-3)
4. Threshold [↑](#footnote-ref-4)
5. Label [↑](#footnote-ref-5)
6. Learning Algorithm [↑](#footnote-ref-6)
7. Optimization Algorithm [↑](#footnote-ref-7)
8. Cost function [↑](#footnote-ref-8)
9. Gradient Descend [↑](#footnote-ref-9)
10. Data points [↑](#footnote-ref-10)
11. Linearly separable [↑](#footnote-ref-11)
12. Imbalance [↑](#footnote-ref-12)
13. Noise [↑](#footnote-ref-13)
14. Train [↑](#footnote-ref-14)
15. Validation [↑](#footnote-ref-15)
16. Hyperparameters [↑](#footnote-ref-16)
17. Iteration [↑](#footnote-ref-17)
18. Solver [↑](#footnote-ref-18)
19. Learnin rate [↑](#footnote-ref-19)
20. Penalty (Regularization) [↑](#footnote-ref-20)
21. Bias [↑](#footnote-ref-21)
22. Variance [↑](#footnote-ref-22)
23. Pandas dataframe [↑](#footnote-ref-23)
24. String [↑](#footnote-ref-24)
25. Label [↑](#footnote-ref-25)
26. Synthetic Minority Over-sampling Technique [↑](#footnote-ref-26)
27. Test [↑](#footnote-ref-27)
28. Centered [↑](#footnote-ref-28)
29. Opposite [↑](#footnote-ref-29)
30. برچسب [↑](#footnote-ref-30)
31. Scaler [↑](#footnote-ref-31)
32. Accuracy [↑](#footnote-ref-32)
33. Generalization [↑](#footnote-ref-33)
34. Subclass [↑](#footnote-ref-34)