

دانشکده مهندسی برق \_گرایش کنترل

مینی پروژه شماره دو

***یادگیری ماشین***

**نگارش**

فاطمه امیری

**40202424**

[لینک گوگل کولب](https://colab.research.google.com/drive/1YsPMz2hQ84kXZEuALBC1jEZGpoZFmZDf?usp=sharing)

[لینک گیت هاب](https://github.com/Fatemeh99Amiri/MachineLearning2024-/tree/main/MiniProject2)

**استاد مربوطه**

جناب آقای دکتر علیاری

**بهار 1403**



فهرست مطالب

[سوال یک 1](#_Toc166771679)

[بخش اول 1](#_Toc166771680)

[بخش دوم 4](#_Toc166771681)

[بخش سوم 6](#_Toc166771682)

[سوال دو 12](#_Toc166771683)

[بخش اول 17](#_Toc166771684)

[بخش دوم 22](#_Toc166771685)

[بخش سوم 23](#_Toc166771686)

[بخش چهارم 34](#_Toc166771687)

[سوال سه 40](#_Toc166771688)

[بخش اول 40](#_Toc166771689)

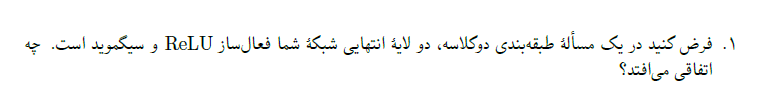
[بخش دوم 49](#_Toc166771690)

[بخش سوم 53](#_Toc166771691)

[سوال چهار 56](#_Toc166771692)

# سوال یک

## بخش اول



در یک مسئله دسته‌بندی دو کلاسه، انتخاب تابع‌های فعال‌سازی در لایه‌های نهایی شبکه عصبی می‌تواند به طور قابل توجهی بر عملکرد و تفسیر خروجی آن تأثیر بگذارد. تابع فعال‌ساز در شبکه عصبی وظیفه‌ی اعمال یک تبدیل غیرخطی بر خروجی هر نورون را دارد. این تبدیل به شبکه کمک می‌کند تا الگوهای پیچیده و غیرخطی را یاد بگیرد. دو تابع فعال‌ساز پرکاربرد عبارتند از:

1. ReLU : که خروجی آن اگر ورودی مثبت باشد، برابر با ورودی است و در غیر این صورت صفر است. این تابع به دلیل سادگی و کارایی در کاهش مشکل ناپدید شدن گرادیان بسیار محبوب است. این تابع اجازه نمیدهد که همه نورون ها در یک زمان فعال باشند، یعنی اگر یک ورودی منفی باشد، این تابع ان را صفر می کند و به نورون اجازه فعال سازی نمی دهد، در واقع با این کار تعداد نورون های فعال کاهش می یابد و محاسبات شبکه نیز کاهش می یابد.

فرمول محاسبه آن به صورت زیر است :

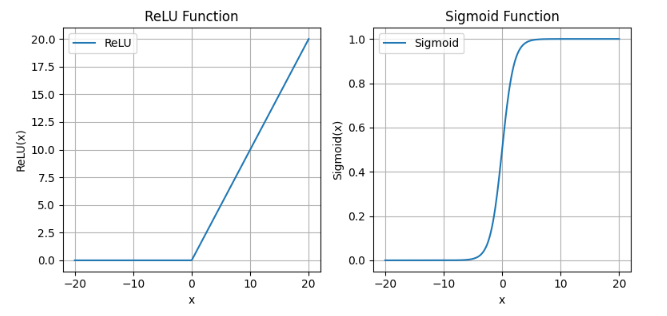


1. سیگموئید : که خروجی آن یک مقدار بین 0 و 1 است و معمولاً برای مسائل دسته‌بندی دودویی استفاده می‌شود، زیرا می‌تواند احتمال تعلق به یک کلاس خاص را مدل‌سازی کند.( یک تابع پیوسته مشتق پذیر غیرخطی است)

فرمول محاسبه آن به صورت زیر است :



با استفاده از کد پایتون ، شکل تابع های فوق به صورت زیر است :



هنگامی که داده‌ها از تابع ReLU عبور می‌کنند، مقادیر مثبت بدون تغییر باقی می‌مانند و مقادیر منفی به صفر تبدیل می‌شوند. بنابراین، خروجی این لایه شامل اعداد غیرمنفی است. پس از آن، خروجی لایه قبلی(a) در وزن‌های لایه نهایی ضرب می‌شود و مقادیر به تابع فعال‌سازی سیگموید وارد می‌شوند. در یک طبقه‌بندی دوکلاسه، لایه نهایی فقط یک نورون دارد و خروجی آن در بازه [0, 1] خواهد بود.

در طبقه‌بندی دوکلاسه، نیازی به استفاده از دو نورون نیست زیرا اگر یک ورودی به یک کلاس تعلق نداشته باشد، به کلاس دیگر تعلق خواهد داشت. به همین دلیل از یک نورون استفاده می‌کنیم تا حجم محاسبات را کاهش دهیم. به عبارت دیگر، درصد تعلق ورودی به یک کلاس را محاسبه می‌کنیم و هر چقدر که به این کلاس تعلق نداشته باشد، به دیگری تعلق دارد.

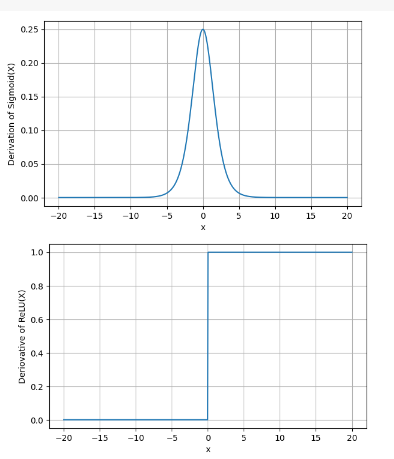
در نهایت، با مقایسه مقدار خروجی با یک آستانه یا threshold، تعیین می‌کنیم که ورودی به کلاس 1 تعلق دارد یا به کلاس 0.

تابع سیگموید زمانی که اشباع می‌شود، در فرآیند Backward Propagation مشکلاتی ایجاد می‌کند زیرا شیب آن در نزدیکی صفر بسیار کوچک است. این باعث می‌شود که گرادیان‌ها تقریباً ناپدید شوند و به شبکه اجازه یادگیری موثر را نمی‌دهد، به ویژه در لایه‌های عمیق‌تر. این مشکل به نام "مشکل ناپدید شدن گرادیان" شناخته می‌شود و منجر به سرعت یادگیری بسیار کند می‌شود، زیرا تغییرات وزن‌ها بسیار کم خواهد بود.

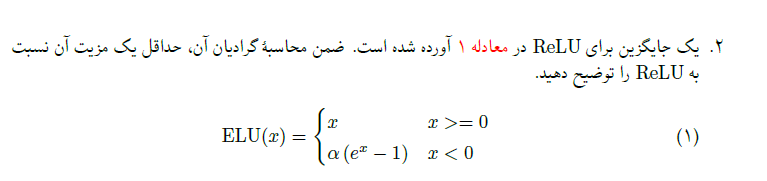
در مقابل، تابع ReLU زمانی که ورودی بزرگ‌تر از صفر باشد، خروجی آن برابر با ورودی است و شیب آن برابر با یک است. این ویژگی باعث می‌شود که گرادیان‌ها در طول Backward Propagation ثابت بمانند و ناپدید نشوند. به همین دلیل، ReLU معمولاً برای شبکه‌های عمیق ترجیح داده می‌شود زیرا باعث می‌شود که مدل به راحتی یاد بگیرد و تغییرات وزن‌ها سریع‌تر انجام شود. با این حال، ReLU نیز مشکلات خاص خود را دارد. یکی از این مشکلات "ReLU مرده" است، یعنی نورون‌هایی که خروجی آن‌ها همیشه صفر است و دیگر در فرآیند یادگیری شرکت نمی‌کنند. این اتفاق زمانی می‌افتد که ورودی به ReLU منفی باشد و باعث شود که نورون به طور دائمی غیر فعال شود.

استفاده از ترکیب این دو تابع فعال‌سازی سیگموید و ReLU می‌تواند به بهره‌برداری از مزایای هر دو کمک کند. به عنوان مثال، می‌توان از ReLU در لایه‌های مخفی استفاده کرد تا مشکلات ناپدید شدن گرادیان کاهش یابد و از سیگموئید در لایه خروجی استفاده کرد تا خروجی به صورت احتمال بین 0 و 1 تفسیر شود. این ترکیب معمولاً به بهبود عملکرد مدل کمک می‌کند، به ویژه در مسائل پیچیده که نیاز به یادگیری عمیق دارند. با این روش، شبکه می‌تواند از غیرخطی بودن و پراکندگی ReLU بهره‌مند شود و در عین حال یک تفسیر احتمالی از خروجی داشته باشد که توسط تابع سیگموید فراهم می‌شود.

مشتق این دو تابع نیز به صورت زیر است :

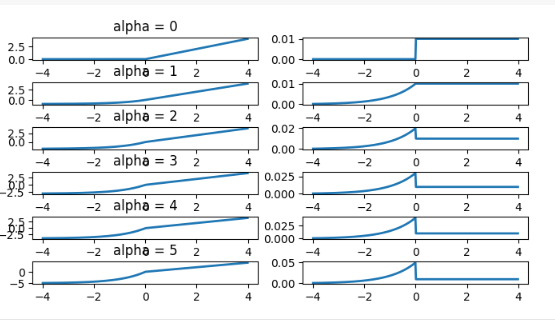


## بخش دوم



در ابتدا به محاسبه گرادیان آن می پردازیم. ( تابع جایگزین را f فرض می کنیم )

برای بررسی دقیق تر مقادیر مختلفی را به ازای الفا قرار می دهیم و خروجی را بدست می آوریم. به ازای آلفا از صفر تا 5 خروجی توسط کد پایتون به صورت زیر است :



در شکل بالا تابع EReLU و گرادیان آن برای الفا های مختلف است، بدین صورت که تصاویر سمت چپ مربوط به خود تابع و تصاویر سمت راست مربوط به مشتق آن می باشند. میبینیم که اگر مقدار باشد انگاه تابع ReLU بدست می آید (مانند قسمت قبل) . این تابع به ازای در نقطه دارای مشتق می باشد. در هر α دیگری، در نقطه مشتق نداریم.

* برتری این تابع نسبت به ReLU عبارت است از:

1. قابلیت خروجی دادن برای مقادیر منفی

یکی از مهم‌ترین مزایای ELU نسبت به ReLU این است که ELU برای ورودی‌های منفی نیز خروجی‌های غیر صفر تولید می‌کند. در حالی که ReLU برای مقادیر منفی ورودی صفر تولید می‌کند، ELU با استفاده از یک تابع نمایی برای ورودی‌های منفی، خروجی‌های منفی تولید می‌کند. این ویژگی به شبکه عصبی کمک می‌کند تا اطلاعات بیشتری از ورودی‌های منفی حفظ کند و از مشکل "ReLU مرده" جلوگیری کند. مشکل "ReLU مرده" زمانی رخ می‌دهد که نورون‌های ReLU به دلیل ورودی‌های منفی یا صفر به طور دائمی غیرفعال می‌شوند.

2. نرمی و مشتق‌پذیری بیشتر در نقطه صفر

تابع ELU در نقطه 0 پیوسته و مشتق‌پذیر است، در حالی که ReLU در نقطه 0 ناپیوستگی دارد و مشتق‌پذیر نیست. این نرمی و مشتق‌پذیری باعث می‌شود که گرادیان‌ها به طور پیوسته و هموار تغییر کنند و فرآیند آموزش شبکه عصبی پایدارتر و مؤثرتر باشد. نرمی بیشتر در نقطه 0 باعث می‌شود که ELU در بهینه‌سازی و همگرایی شبکه عصبی عملکرد بهتری داشته باشد.

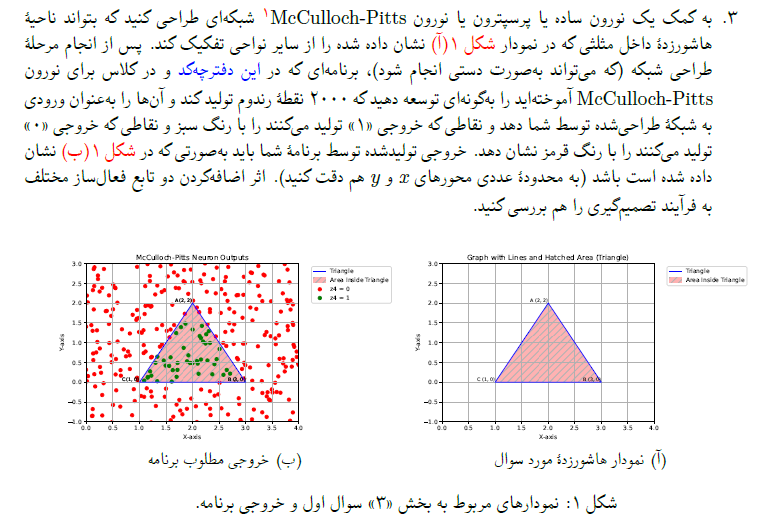
3. کاهش میانگین خروجی و تثبیت گرادیان‌ها

ELU با داشتن مقادیر منفی برای ورودی‌های منفی، میانگین خروجی‌ها را به سمت صفر نزدیک می‌کند. این ویژگی می‌تواند به تثبیت و تسریع فرآیند آموزش کمک کند، زیرا باعث می‌شود که گرادیان‌ها به طور مؤثرتری جریان یابند و ناپدید شدن یا انفجار گرادیان‌ها کاهش یابد. ReLU این خاصیت را ندارد و میانگین خروجی آن مثبت است، که می‌تواند به عدم تعادل در جریان گرادیان‌ها منجر شود.

4. سرعت همگرایی بیشتر

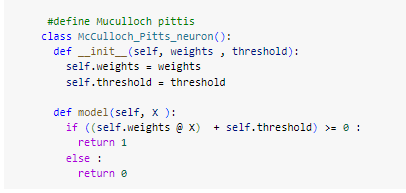
شبکه‌های عصبی با استفاده از ELU می‌توانند سریع‌تر همگرا شوند. به دلیل ویژگی‌های منحصر به فرد ELU مانند تولید مقادیر منفی، نرمی بیشتر، و کاهش میانگین خروجی، فرآیند بهینه‌سازی شبکه سریع‌تر و پایدارتر انجام می‌شود. این ویژگی به خصوص در شبکه‌های عمیق که مشکلات ناپدید شدن گرادیان و کندی آموزش شایع‌تر هستند، بسیار مفید است.

## بخش سوم



برای انجام این بخش، از مدل MCP استفاده می‌کنیم. فرض می‌کنیم که سه خط وجود دارند و اگر نقطه‌ای نسبت به این خطوط شرایط خاصی را برآورده کند؛ مثلاً پایین‌تر از یک خط و بالاتر از دو خط دیگر باشد، در ناحیه هاشورخورده قرار می‌گیرد. چون در این مسئله نیاز است که یک سری شرایط به صورت درست یا نادرست بررسی شوند، از نورون MCP استفاده می‌کنیم.

در اینجا از تابع فعال ساز استفاده نمی کنیم و کلاس McCulloch pittis به صورت زیر تعریف می شود:



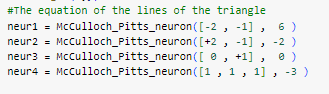
برای تعیین نقاطی که در ناحیه هاشورخورده قرار دارند، باید معادلات خطوطی که این ناحیه را ایجاد می‌کنند به صورت مناسب تعریف شوند. برای درک بهتر این موضوع، تئوری حل مسئله را توضیح می‌دهیم و سه معادله خط به دست می‌آوریم که از تقاطع این خطوط، مثلث مورد نظر ایجاد می‌شود.

سه معادله خط به صورت زیر هستند :

اکنون برای آن که نقاط بین این خطوط قرار گیرند، باید در شرط های زیر صدق کنند :

با توجه به معادلات حاصل، می‌خواهیم تعداد نورون‌های MCP مورد نیاز را محاسبه کنیم. با توجه به اینکه با استفاده از سه معادله موفق به تشخیص ناحیه مورد نظر شده‌ایم، باید سه نورون MCP را طراحی کنیم. هدف این سه نورون این است که با گرفتن ورودی‌های خود، که مختصات x و y هستند، اعلام کنند که آیا نقطه مورد نظر در ناحیه مورد نظر واقع شده یا خیر. علاوه بر این، یک نورون دیگر نیز باید طراحی شود که با دریافت خروجی سه نورون اول، تصمیم بگیرد که آیا نقطه مورد نظر درون ناحیه تعیین شده قرار دارد یا خیر.

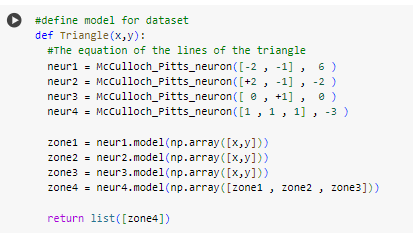
در نتیجه، ما سه نورون داریم که ورودی‌های x و y را دریافت کرده و اگر خروجی تمامی این نورون‌ها True باشد، آنگاه با گذر از نورون نهایی(neur4) که پس از نورون‌های اولیه قرار می‌گیرد، تصمیم گیری می‌شود که آیا نقطه در ناحیه مشخص شده قرار دارد یا خیر. وزن‌ها و آستانه‌های نورون‌ها به شکل زیر محاسبه می‌شوند:



که در واقع اولین عضو وزن ها هستند و عضو دوم threshold ها هستند. لازم به ذکر است که ما محور x و y نداریم و منظور از x و y در معادلات بالا، همان x1 و x2 است و وزن ها همان ضرایب x ها هستند و threshold هم همان بایاس عرض از مبدا می باشد.

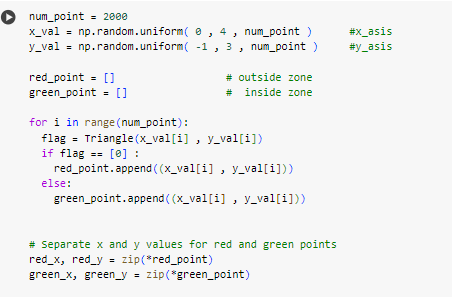
درواقع در کد بالا داده هایی که داخل مثلث باشند مقدار صفر می گیرند یعنی مقدار برای هر سه خط، یک می شود و از مقدار 3 کم میشود و درنهایت صفر میشود. (neur4). وزن های این نورون به این معنی است که نقاطی که توسط نورون‌های قبلی تشخیص داده شده‌اند (نقاطی که داخل مثلث قرار دارند)، همگی به عنوان ورودی به این نورون وارد می‌شوند. زمانی که همه این ورودی‌ها یک باشند، مقدار خطی این نورون (مجموع وزن‌ها ضرب در ورودی‌ها به اضافه آستانه) برابر با (1 \* 1) + (1 \* 1) + (1 \* 1) - 3 = 0 می‌شود.

پس یک تابع Triangle می نویسیم که به صورت زیر است :

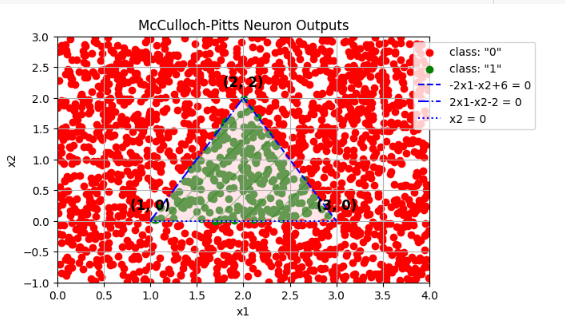


در این شبکه، خروجی‌های سه نورون اول به عنوان ورودی به نورون چهارم می‌رسند. اگر خروجی همه سه نورون اول برابر 1 باشد، به این معنی است که نقطه مورد نظر در ناحیه مورد نظر قرار دارد. اما اگر حداقل یکی از خروجی‌های نورون‌های اولیه صفر باشد، نورون نهایی مقدار 0 را باز می‌گرداند، که نشان می‌دهد که نقطه مورد نظر در ناحیه مورد نظر قرار نمی‌گیرد.

اکنون 2000 نقطه رندوم را به مدل می دهیم تا تشخیص دهد و اگر نقطه داخل ناحیه بود با رنگ سبز و اگر نقطه خارج از ناحیه بود با رنگ قرمز مشخص کند.



حال خروجی به صورت زیر است :

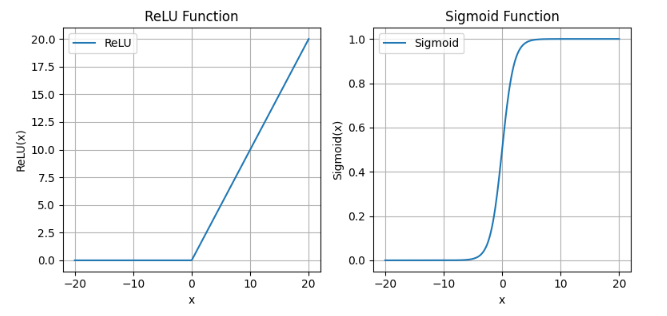


می بینیم که مدل به خوبی عمل کرده است و نقاط را به خوبی تشخیص داده است.

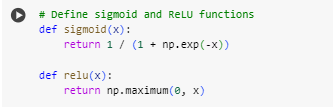
مدلی که در اینجا استفاده کردیم با استفاده از تابع فعال‌ساز hard limit بود. این تابع فعال‌ساز، برای تمامی نورون ها استفاده شد و با استفاده از تابع hard limit، به عنوان خروجی صرفا قرار داشتن یا نداشتن نقطه در ناحیه خواسته شده معلوم می شود.

***در این قسمت اثر اضافه کردن دو تابع فعال ساز به فرایند تصمیم گیری را بررسی می کنیم :***

از توابع فعال ساز سیگموید و ReLU استفاده می کنیم. نمودار این ها به صورت زیر است.

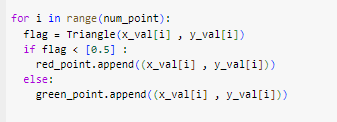


و کد این توابع به صورت تابع های زیر است :

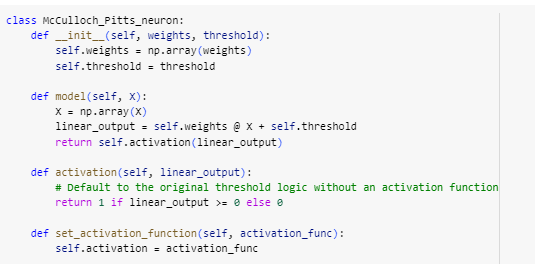


**در ابتدا از سیگموید استفاده می کنیم :**

خروجی این تابع بین ۰ و ۱ قرار می‌گیرد. وزن‌های نورون‌ها طوری تنظیم می‌شوند که اگر خروجی به ۱ نزدیک باشد، اشاره به حضور در ناحیه مورد نظر دارد و اگر به ۰ نزدیک باشد، به خروج از ناحیه مورد نظر اشاره می‌دهد. به دلیل پیوستگی خروجی تابع، برای تصمیم‌گیری در مورد تعلق به یک کلاس، یک آستانه یا threshold تعیین می‌شود. برای استفاده از تابع sigmoid، آستانه را معمولاً در مقدار ۰.۵ تعیین می‌کنیم، به این معنی که اگر خروجی از این تابع بیشتر از ۰.۵ باشد، به یک کلاس نسبت داده می‌شود و اگر کمتر باشد، به کلاس دیگری نسبت داده می‌شود. پس به ازای تمام نقاط رندوم، بررسی می شود که کمتر از 0.5 است یا بیشتر، تا متوجه شویم مربوط به کلاس قرمز است یا سبز !



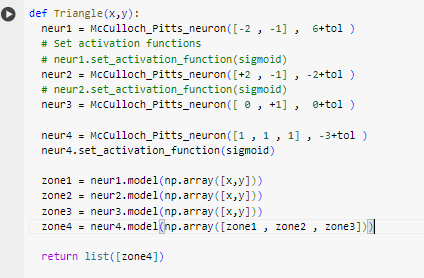
یک کلاس به صورت زیر تعریف می کنیم که از تابع فعال ساز سیگموید استفاده می کنیم:



متغیر tol را در اینجا در نظر می گیریم. این متغیر برای تنظیم حساسیت یا دقت مدل در تصمیم‌گیری‌هایی که بر اساس آستانه انجام می‌شود، استفاده شود. به هر یک از آستانه‌های نورون‌ها یک مقدار tol اضافه می شود به عبارت دیگر، مقادیر آستانه نورون‌ها از آستانه‌های اصلی کمی بیشتر هستند. این عمل باعث می‌شود که نورون‌ها به عنوان ناحیه تشخیص‌دهنده، به نقاطی که در مرزها قرار دارند، حساس‌تر باشند و دقت بیشتری در تصمیم‌گیری در مورد تعلق یا عدم تعلق نقاط به داخل مثلث داشته باشند.

در کل، با افزایش مقدار tol، حساسیت مدل در تصمیم‌گیری‌هایی که بر اساس آستانه انجام می‌شود، کمتر می‌شود، و با کاهش آن، حساسیت بیشتری خواهد داشت. این به معنی این است که با افزایش tol، مدل به تصمیم‌های مبتنی بر آستانه‌ها کمتر اعتماد می‌کند و با کاهش آن، اعتماد بیشتری دارد. هر چقدر مقدار tol به صفر نزدیک باشد، مدل به hard limit نزدیک تر می شود و هر چقدر بزرگتر در نظر گرفته شود، تعداد داده هایی هایی که اشتباه طبقه بندی شده اند بیشتر است.

پس تغییر آستانه‌ها با اضافه کردن tol به مدل کمک کند تا نقاطی که در مرز قرار دارند، دقیق‌تر طبقه‌بندی شوند. این کار به خصوص در نورون‌هایی که مسئول تشخیص نقاط داخل یا خارج از یک ناحیه خاص هستند، اهمیت دارد. پس تابع تشکیل مثلث را این بار با در نظر گرفتن tol به صورت زیر تعریف می کنیم :

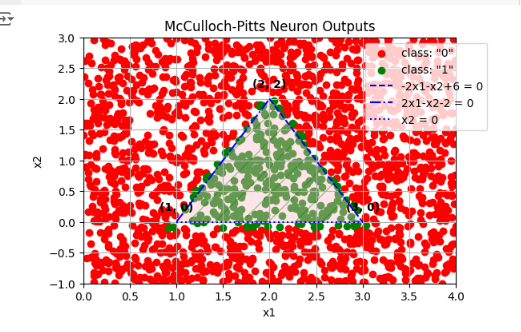


با تنظیم تابع فعال‌سازی به سیگموید، خروجی نورون چهارم به یک مقدار پیوسته بین ۰ و ۱ تبدیل می‌شود. این تغییر باعث می‌شود که نورون چهارم بتواند دقت بیشتری در تصمیم‌گیری داشته باشد. خروجی نورون چهارم اکنون یک مقدار بین ۰ و ۱ است که نشان‌دهنده احتمال تعلق نقطه ورودی به کلاس مورد نظر است.

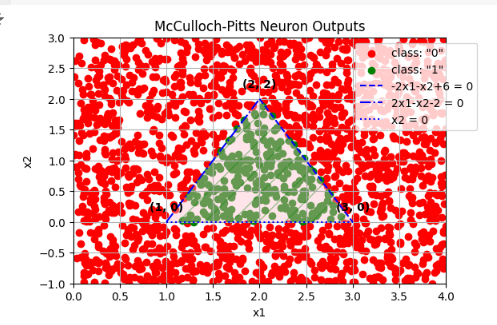
اگر خروجی نورون چهارم نزدیک به ۱ باشد، به معنای حضور نقطه در مثلث است و اگر خروجی نورون چهارم نزدیک به ۰ باشد، به معنای خارج بودن نقطه از ناحیه مورد نظر است.

برای تصمیم‌گیری نهایی، معمولاً یک آستانه (مثلاً ۰.۵) در نظر گرفته می‌شود. اگر خروجی بیشتر از ۰.۵ باشد، نقطه به یک کلاس نسبت داده می‌شود و اگر کمتر باشد، به کلاس دیگر نسبت داده می‌شود.

مشابه حالت قبل خروجی را برای 2000 داده رندوم رسم می کنیم و tol را برابر 0.1 در نظر می گیریم :



می بینیم که تعدادی از داده ها اشتباه طبقه بندی شده اند، حال مقدار tol را برابر 0.0001 در نظر می گیریم و داریم :

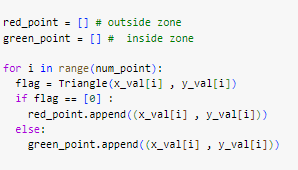


مشاهده می شود که در این حالت طبقه بندی بهتر صورت گرفته است و نتیجه می شود که کم بودن tol عملکرد مدل را مانند حالت بدون تابع فعال ساز قسمت قبل، می کند.

**اکنون از تابع ReLU استفاده می کنیم :**

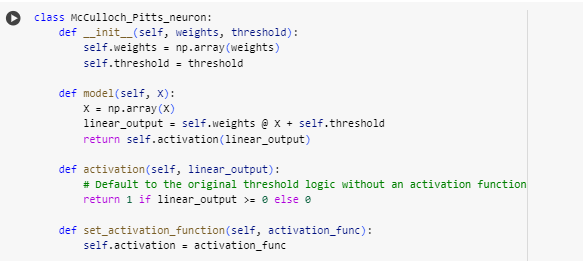
تابع ReLU یک تابع فعال‌سازی است که برای استفاده در شبکه‌های عصبی مصنوعی استفاده می‌شود و در بخش های قبل به توضیح بیشتر درباره آن پرداختیم . محدوده خروجی به این صورت است که تابع ReLU خروجی صفر را برای همه مقادیر منفی و مقدار ورودی را برای مقادیر مثبت برمی‌گرداند. به عبارت دیگر، اگر ورودی مثبت باشد، خروجی برابر با خود ورودی است و اگر منفی باشد، خروجی صفر است.

پس درواقع یک شرط می نویسیم که ببینیم هر نقطه رندوم مربوط به کدام کلاس است. ( اگر مقدار صفر شد خارج مثلث است در غیر این صورت اگر مقدار برابر خود ورودی شد، داخل مثلث است.)

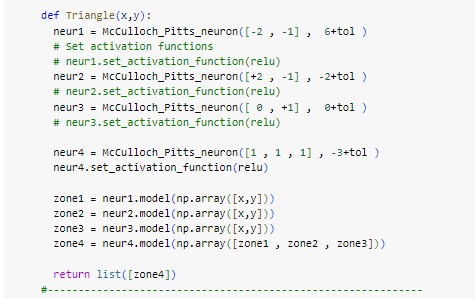


در تفسیر ساده‌تر، ReLU مانند یک سوئیچ عمل می‌کند. اگر مقدار ورودی مثبت باشد، خروجی همان مقدار ورودی است و اگر مقدار ورودی منفی باشد، خروجی صفر است.

مانند قسمت قبل یک کلاس برای تعریف MCP داریم : که از تابع فعال ساز هم استفاده می شود.



سپس تابعی برای محاسبه نقاط داخل مثلث می نویسیم :



در کد بالا نورون چهارم برای تعیین نهایی اینکه نقطه در داخل یا خارج از مثلث قرار دارد استفاده می‌شود. با استفاده از دستور neur4.set\_activation\_function(relu) تابع فعال‌سازی این نورون به تابع ReLU تغییر داده می‌شود.

به طور پیش‌فرض، نورون‌های McCulloch\_Pitts از یک تابع آستانه ساده (خطی) استفاده می‌کنند که خروجی را به 0 یا 1 محدود می‌کند.

با تنظیم تابع فعال‌سازی به ReLU، نورون چهارم دیگر از این تابع پیش‌فرض استفاده نمی‌کند. در عوض، تابع ReLU اعمال می‌شود که خروجی آن برای مقادیر منفی صفر و برای مقادیر مثبت همان مقدار ورودی خواهد بود.

وقتی ورودی‌های zone1, zone2, و zone3 به neur4 داده می‌شوند، نورون چهارم مقدار خطی خود را محاسبه می‌کند (مجموع وزن‌ها ضربدر ورودی‌ها به علاوه آستانه). سپس، تابع ReLU این مقدار خطی را تبدیل می‌کند: اگر این مقدار خطی منفی باشد، خروجی نورون چهارم صفر خواهد بود؛ اگر مثبت باشد، خروجی همان مقدار خطی خواهد بود.

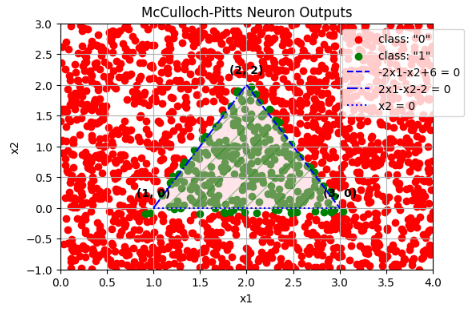
پس با تنظیم تابع فعال‌سازی neur4 به ReLU، عملکرد نورون چهارم تغییر می‌کند تا مقادیر منفی را به صفر تبدیل کند و مقادیر مثبت را به همان شکل حفظ کند. این تغییر می‌تواند تأثیر مهمی در تصمیم‌گیری نهایی داشته باشد، به خصوص زمانی که ترکیب ورودی‌ها از نورون‌های قبلی (که تعیین می‌کنند نقطه در داخل یا خارج مثلث است) مورد نظر است.

دقیقا مانند حالت قبل در اینجا هم باید از یک tol یا تولرانس استفاده کنیم که تنظیم مقدار آن دست ماست و هر چه به صفر نزدیک تر باشد طبقه بندی بهتر صورت می گیرد و مدل به hard limit شبیه تر می شود.

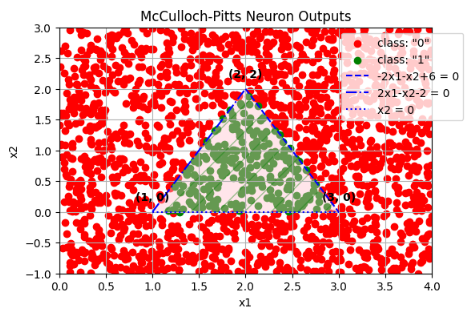
به هر آستانه‌ای که برای نورون‌ها تعیین شده است، مقدار tol اضافه می کنیم. به عنوان مثال، آستانه نورون اول به جای 6 برابر با 6.0001 شده است ( به ازای tol=0.0001) . این کار به منظور تنظیم دقیق‌تر مرزهای تصمیم‌گیری و بهبود دقت مدل انجام شده باشد. با اضافه کردن tol به آستانه، مقدار خطی خروجی نورون‌ها به مقدار بسیار کمی تغییر می‌کند. این تغییر می‌تواند به عنوان یک تلورانس کوچک در تصمیم‌گیری‌ها عمل کند.

به عنوان مثال، اگر مقدار خطی خروجی یک نورون بسیار نزدیک به آستانه باشد، اضافه کردن tol می‌تواند تعیین کند که خروجی نهایی نورون 0 یا 1 باشد.

برای دو تلورانس مختلف چک می کنیم. ابتدا tol را برابر 0.1 در نظر می گیریم :



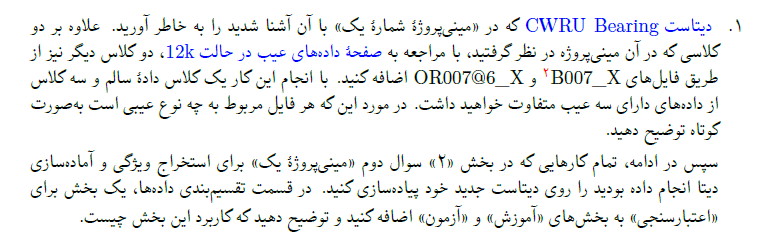
می بینیم که تعدادی از داده ها به خوبی طبقه بندی نشده اند، پس مقداری نزدیک تر به صفر در نظر می گیریم، اکنون tol را برابر 0.0001 در نظر می گیریم :



می بینیم که در این حالت طبقه بندی مدل خیلی بهتر شده است و عملکردی مشابه حالت hard limit دارد.

# سوال دو

## بخش اول

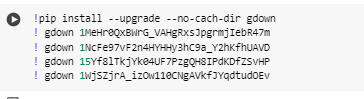


داده‌های جمع‌آوری‌شده برای یاتاقان‌های سالم و معیوب در سمت رانش (drive end) و سمت فن (fan end) با سرعت نمونه‌برداری 12,000 و 48,000 نمونه در ثانیه (برای آزمایش‌های سمت رانش) و 12,000 نمونه در ثانیه (برای آزمایش‌های سمت فن) بوده است. فایل‌های داده در فرمت Matlab شامل داده‌های ارتعاشات یاتاقان‌های سمت رانش و سمت فن و همچنین سرعت چرخش موتور هستند. متغیرهای داخل فایل‌ها شامل داده‌های شتاب‌سنج سمت رانش (DE)، داده‌های شتاب‌سنج سمت فن (FE)، داده‌های شتاب‌سنج پایه (BA)، داده‌های سری زمانی (time) و دور در دقیقه (RPM) می‌باشند.

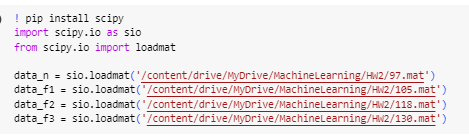
داده‌های معیوب به سه دسته‌بندی اصلی تقسیم می‌شوند: عیب شیار داخلی (Inner race)، عیب ساچمه (Ball) و عیب شیار خارجی (Outer race) نام‌گذاری داده‌ها برای هر نوع عیب به ترتیب به صورت IR007\_0 برای عیب شیار داخلی، B007\_0 برای عیب ساچمه، و OR007@6\_0 برای عیب شیار خارجی است. عیب‌های مربوط به شیار خارجی دارای سه موقعیت مختلف (مرکز، عمود، و مقابل) هستند که در این تمرین، حالت مرکز انتخاب شده است.

پس در کل 4 حالت داریم یک حالت دیتاهای سالم و سه حالت عیب به صورت فوق. داده ها را دانلود می کنیم و در ابتدا مانند تمرین قبل، پیش می رویم.

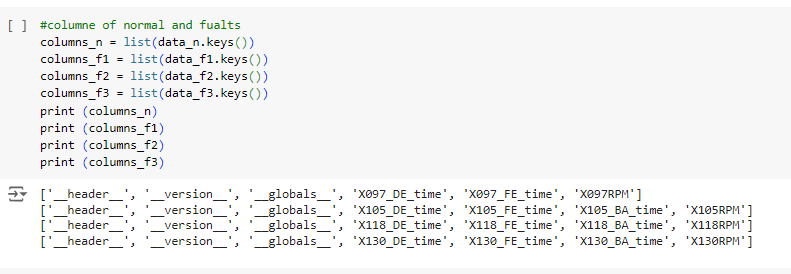
برای دانلود دیتاست‌های مربوط به هر کلاس، از دستور زیر استفاده می شود و 4 دیتاست را دانلود می کنیم.



پس از دانلود داده‌ها، فایل‌ها با استفاده از دستور loadmat از کتابخانه scipy بارگذاری می‌شوند

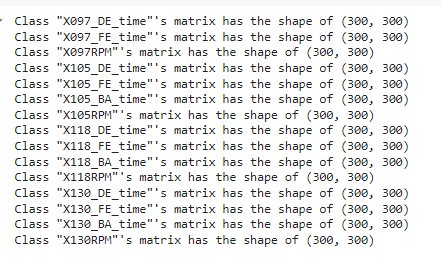


. مشابه تمرین قبلی، داده‌ها از بخش‌های مختلفی تشکیل شده‌اند که در زیر قابل مشاهده است :

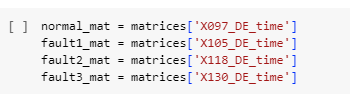


برای ادامه حل مسئله، مدل DE-time که در تمامی فایل‌ها مشترک است، انتخاب می کنیم و سپس همانند تمرین سری اول، 300 نمونه به اندازه 300 از داده های اصلی جدا میکنیم.

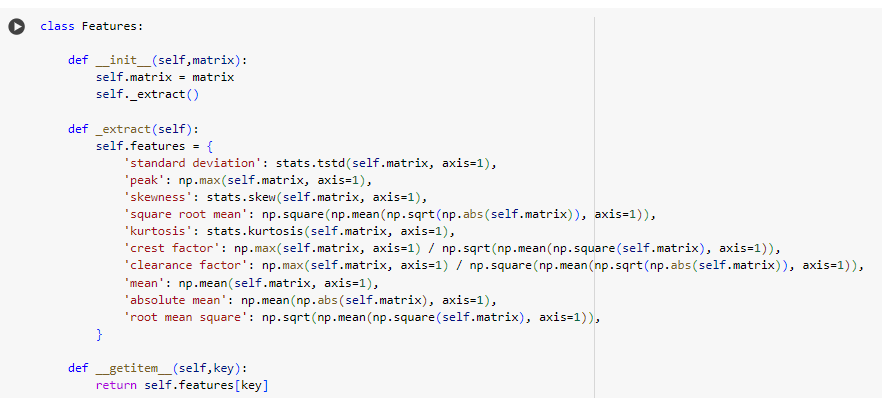
من در ابتدا برای تمام بخش های 4 دیتاست ماتریس های 300\*300 جدا کردم و سپس دیتای DE-time را برای ادامه کار انتخاب کردم. ( درواقع 4 کلاس از 15 کلاس زیر را انتخاب کردم.)

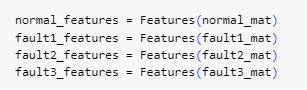


پس 4 دسته داده دارم که به صورت زیر هستند:



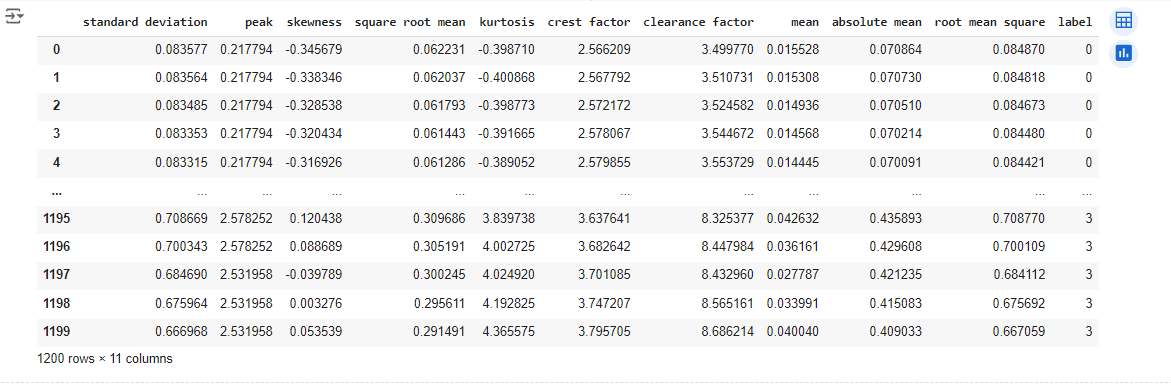
اکنون 10 ویژگی مشابه تمرین قبل از این دیتاست استخراج می کنم:





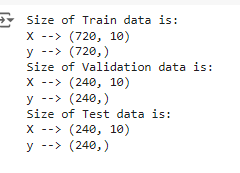
اکنون به داده ها لیبل میزنیم، داده های سالم با لیبل صفر و داده های معیوب؛ با توجه به نوع عیب لیبل های 1و 2و 3 دارند. این لیبل ها برابر مقدار y می شود و فیچر هایی که استخراج شد همان x ها میشود.

داده های هر 4 کلاس را به صورت سطری زیر هم مینویسیم پس 1200 نمونه با 11 ستون داریم که 10 ستون فیچر ها هستند و یک ستون لیبل ها ، مانند زیر :



اما مشکل این است که به ترتیب قرار می گیرند یعنی 300 ستون اول برای داده های نرمال است و .. ، برای حل این مشکل داده ها را shuffle می کنیم.

برای تقسیم بندی داده ها، داده ها را به تست و اموزش و اعتبار سنجی تقسیم می کنیم و نسبت این ها به صورت زیر است :



* کاربرد بخش اعتبار سنجی

مجموعه‌داده اعتبارسنجی (Validation Set) در یادگیری ماشین نقش بسیار مهمی ایفا می‌کند و برای چندین هدف مهم استفاده می‌شود. این اهداف عبارتند از:

1. انتخاب مدل (Model Selection)

انتخاب مدل به معنای انتخاب بهترین مدل از بین چندین مدل مختلف است. با استفاده از مجموعه‌داده اعتبارسنجی، می‌توان عملکرد هر مدل را بررسی کرد و مدلی را که بهترین عملکرد را دارد، انتخاب کرد. این فرآیند به اطمینان از انتخاب مدلی که به خوبی به داده‌های جدید تعمیم می‌یابد، کمک می‌کند.

2. تنظیم‌کردن Hyperparameters (Hyperparameter Tuning)

Hyperparameters پارامترهایی هستند که در فرآیند آموزش مدل تنظیم می‌شوند و تاثیر زیادی بر عملکرد نهایی مدل دارند. این پارامترها شامل عواملی مانند نرخ یادگیری (learning rate)، تعداد لایه‌ها در یک شبکه عصبی و تعداد نورونها در هر لایه هستند. با استفاده از مجموعه‌داده اعتبارسنجی، می‌توان این پارامترها را تنظیم کرد تا مدل بهترین عملکرد ممکن را داشته باشد.

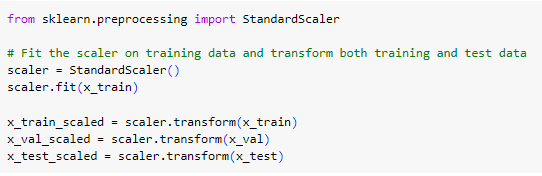
3. جلوگیری از Overfitting

Overfitting زمانی رخ می‌دهد که مدل به خوبی بر روی داده‌های آموزشی عملکرد دارد، اما بر روی داده‌های جدید (که قبلاً ندیده) عملکرد خوبی ندارد. مجموعه‌داده اعتبارسنجی می‌تواند به شناسایی زمانی که مدل شروع به overfit شدن می‌کند کمک کند. این کار با ارزیابی مداوم عملکرد مدل بر روی داده‌های اعتبارسنجی و تطبیق مدل به گونه‌ای که بتواند به خوبی تعمیم دهد، انجام می‌شود.

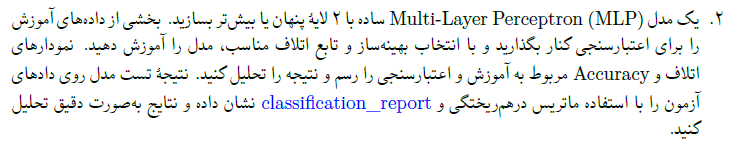
4. Early Stopping در شبکه‌های عصبی و یادگیری عمیق

در یادگیری عمیق و شبکه‌های عصبی، یکی از روش‌های جلوگیری از overfitting، استفاده از تکنیکی به نام Early Stopping است. در این روش، فرآیند آموزش مدل زمانی متوقف می‌شود که عملکرد مدل بر روی مجموعه‌داده اعتبارسنجی شروع به بدتر شدن می‌کند. این نشان می‌دهد که مدل در حال overfit شدن بر روی داده‌های آموزشی است و ادامه آموزش ممکن است به تعمیم‌پذیری مدل آسیب برساند.

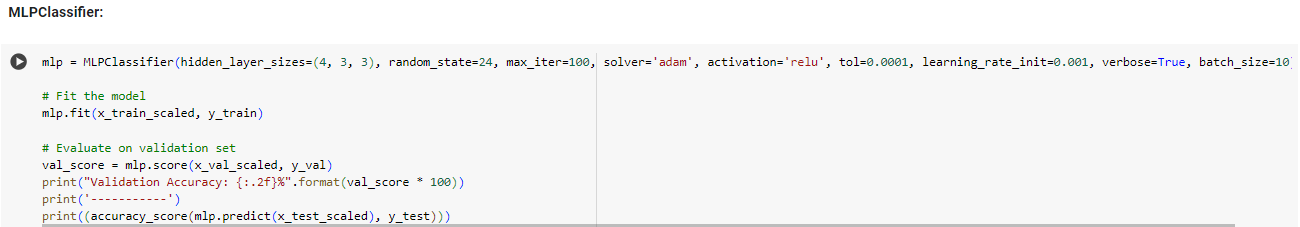
بعد از جداسازی داده‌ها، مرحله بعدی تبدیل آنها به فرم معمولی است، که از طریق دستور StandardScaler انجام می‌شود. در این مرحله، ابتدا با توجه به داده‌های آموزش، پارامترهای scaler را تنظیم می‌کنیم و سپس این تبدیل را بر روی تمامی داده‌ها اعمال می‌کنیم.



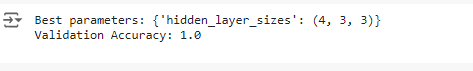
## بخش دوم



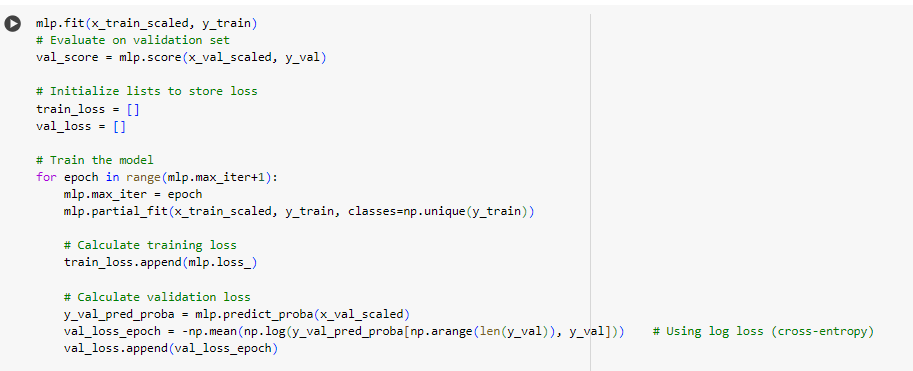
**حالت اول )**

در این جا، برای تشکیل مدل MLP از کتابخانه SKlearn استفاده می کنیم. یک مدل سه لایه با هایپرپارامترهایی به صورت زیر تنظیم می کنیم

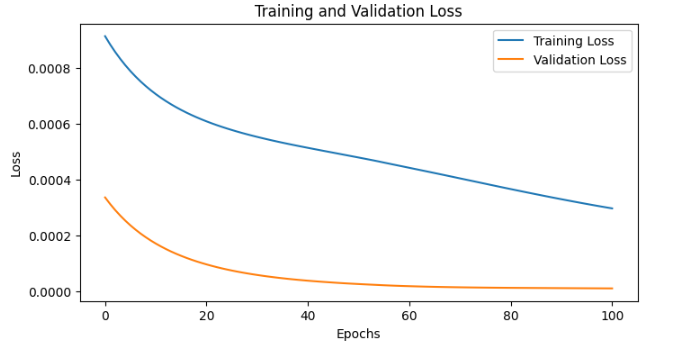
با توجه به یک حلقه for که برای هر لایه مقادیر یک تا 5 نورون قرار می داد و در نتیجه بهترین دقت را میداد، بهترین تعداد نورون برای هر لایه به ترتیب به صورت زیر است که در مدل بالا لحاظ شده است.



با استفاده از کد زیر نمودار loss را برای داده های آموزش و اعتبار سنجی رسم می کنیم:



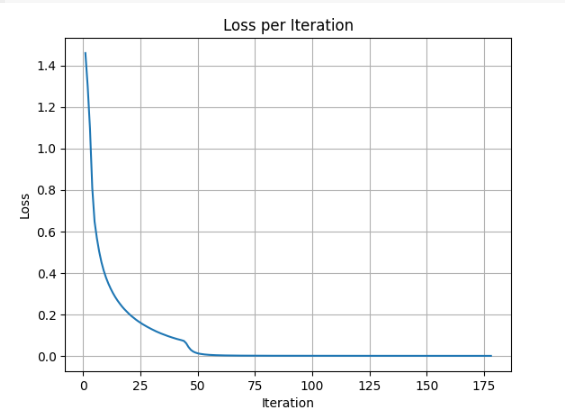
برای هر ایپاک از یک تا 100 رسم میشود. داخل خود تابع mlpclassifier توسط mlp.loss\_ مقدار loss برای داده های آموزش محاسبه میشود اما این مقدار loss برای هر دوره، اخرین مقدار loss آن است که طبیعتا بهترین مقدار آن است.



اگر بخواهیم مقدار loss را برای داده های آموزش در هر ایتریشن را محاسبه کنیم( همان مقادیری که با ران کردن مدل mlp دیده میشوند)، نمودار زیر بدست می اید:

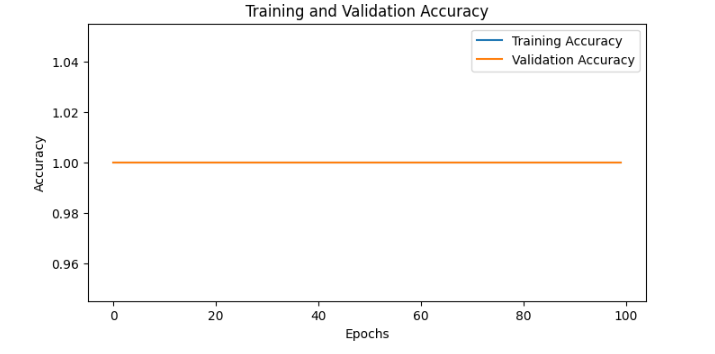
از دستور روبه رو استفاده میشود.

losses = mlp.loss\_curve\_



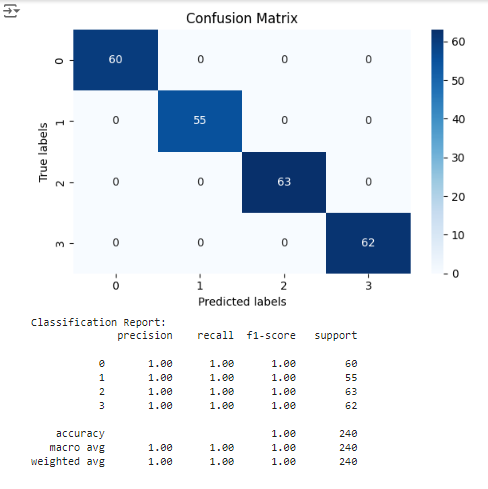
پس علت تفاوت این نمودار با حالت قبل در این است که حالت قبلی، loss های اخرین را در نظر می گیرد.

نمودار های دقت هم به صورت زیر بدست می ایند:



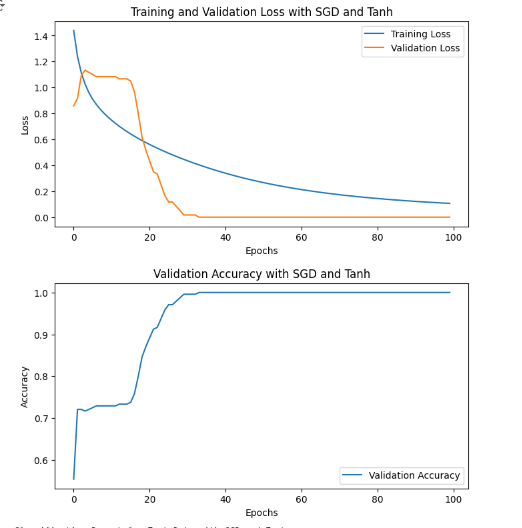
که دقت یک را از همان ابتدا نشان می دهد.

ماتریس درهم ریختگی و classification report هم به صورت زیر هستند که نشان می دهند دقت مدل 100 درصد است و تمام داده ها به خوبی طبقه بندی شده اند.

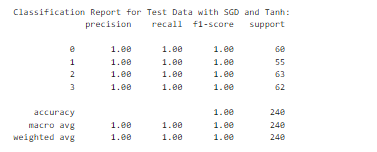


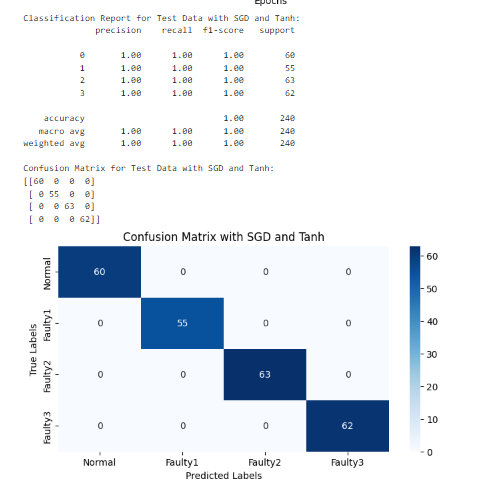
هایپرپارامتر هارا تغییر می دهیم و داریم :

در حالت اول از activation='tanh', solver='sgd' استفاده می کنیم و نمودار ها به صورت زیر بدست می ایند:



و عملکرد مدل هم به صورت زیر است:

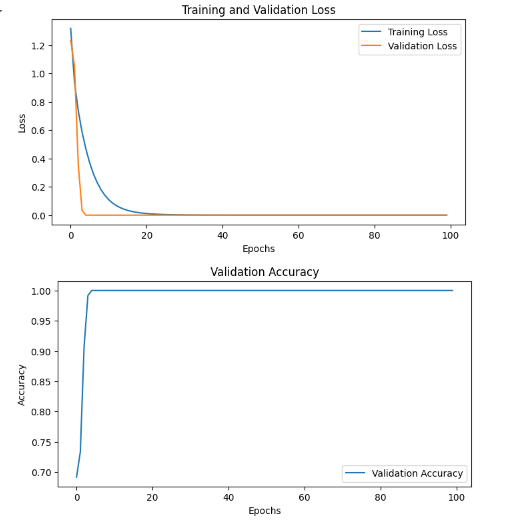




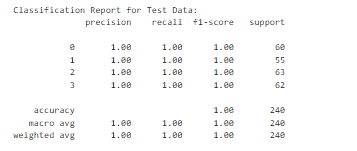
که میبینیم دقت در این حالت برابر 100 درصد شده است.

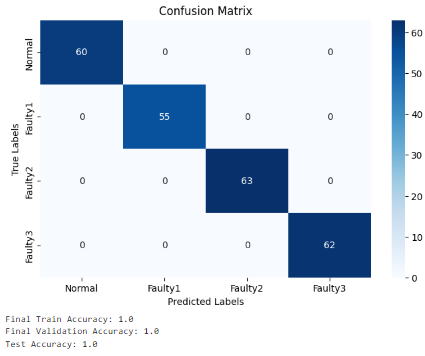
در حالت دوم از activation='tanh', solver='adam' استفاده می کنیم:

نمودار ها به صورت زیر هستند :



و عملکرد مدل به صورت زیر است که میبینیم دقتی برابر با 100 درصد بدست آمده است.



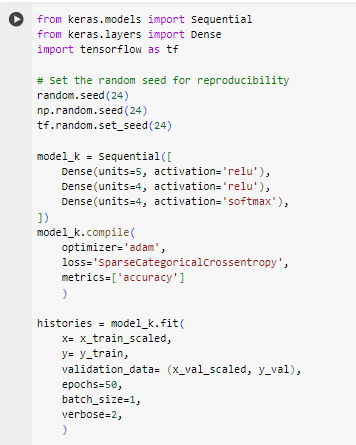


در این حالت هم رسم نمودار ها سخت تر بود و باید از چند حلقه استفاده می کردیم و هم امکان این که برای هر لایه تابع فعال ساز جدا تعریف کنیم و... وجود نداشت. پس به سراغ بررسی حالت دوم رفتیم.

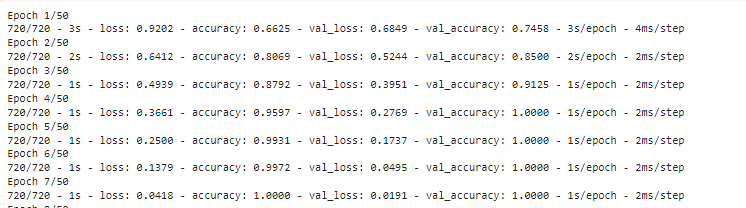
**حالت دوم )**

از Keras Sequential API استفاده کردیم.

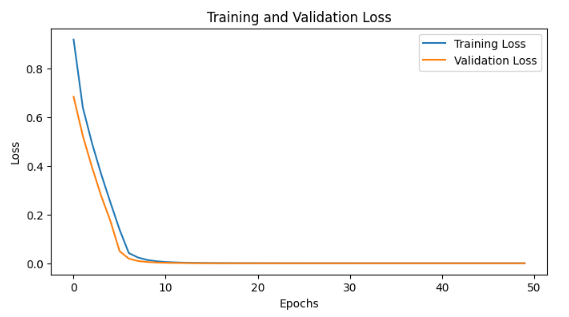
یک شبکه عصبی 3 لایه با تعداد لایه های 5 و 4 و 4 انتخاب کردیم البته با توجه به اینکه 4 کلاس داریم، لزومی ندارد حتما لایه آخر 4 نرون داشته باشد و با یک عدد دو بیتی هم می توان 4 حالت ساخت. در لایه های hidden از فعال ساز relu و لایه آخر از softmax استفاده شد ( به دلیل چند کلاسه بودن داده ها)



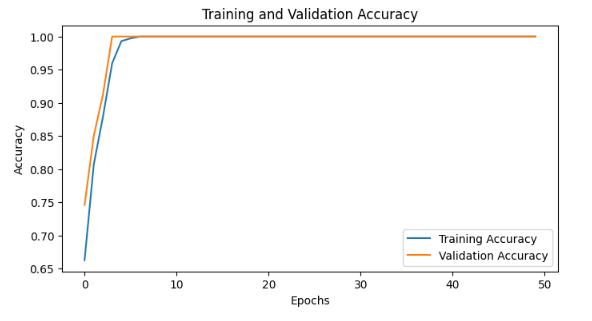
خروجی ها به عنوان مثال برای 7 ایپاک اول به صورت زیر هستند :



حال با توجه به شکل، نمودار های loss و دقت را رسم می کنیم :

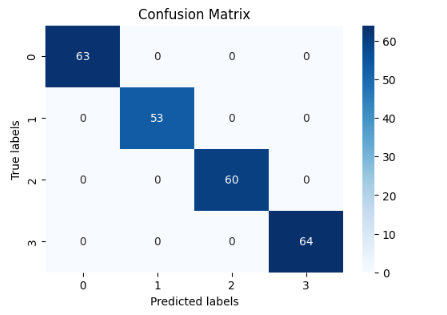


و نمودار دقت به صورت زیر است :



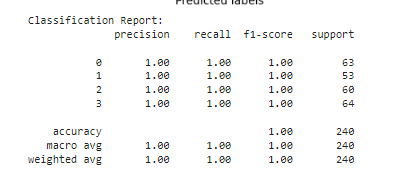
می بینیم که بعد از حدود 8 ایپاک به دقت 100 می رسیم.

ماتریس در هم ریختگی را رسم می کنیم و داریم :

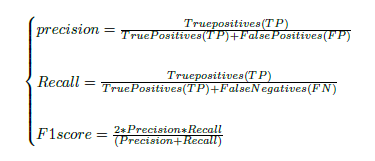


همه مقادیر روی قطر اصلی هستند یعنی به درستی تشخیص داده شده اند.

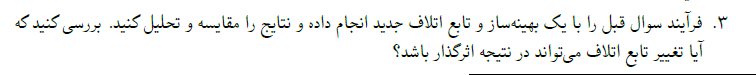
حال برای برسی دقیق از از classification report استفاده می کنیم:



میبینیم که دقت 100 شده است. خوب است راجع به بخش های مختلف آن توضیحی داده شود:



## بخش سوم

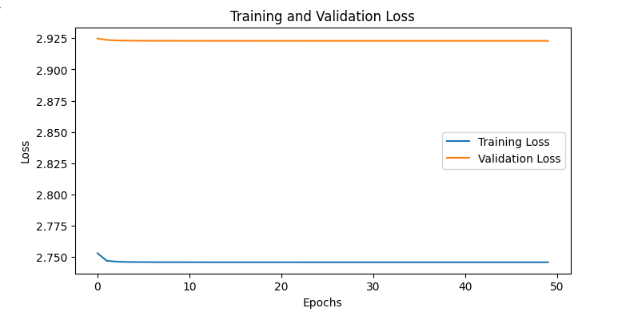


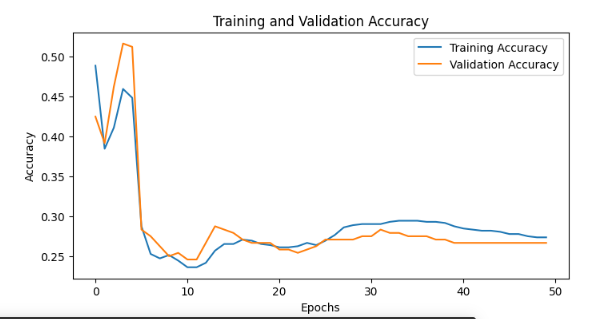
برای حالتی که از sklearn استفاده شده بود، در همان بخش دوم، این قسمت هم بررسی شد.در اینجا حالت دوم بررسی میشود.

حالت اول ) از توابع زیر استفاده می شود: ( بقیه موارد همان حالت قبل است)

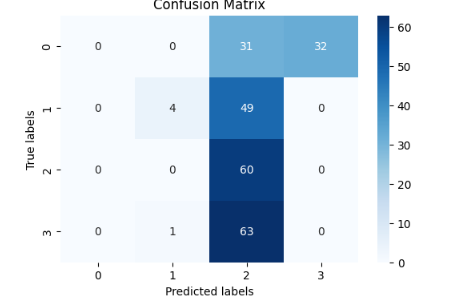
optimizer= SGD(), loss= 'mean\_squared\_error'

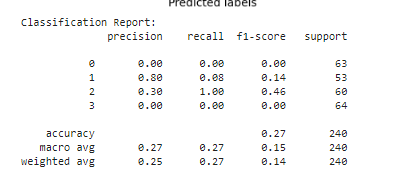
نمودار ها را مشاهده می کنیم :





از شکل ها هم مشخص است که عملکرد مدل بدتر شده است اما به سراغ تحلیل دقیق تر می رویم :





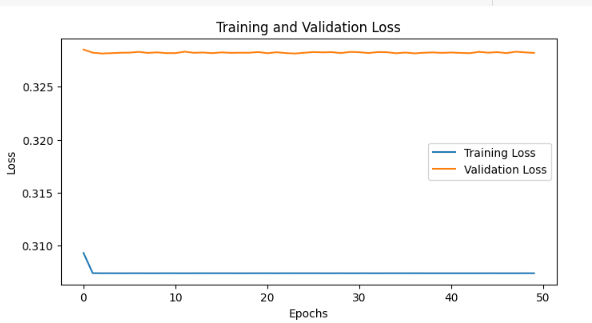
دقت حدود 26 درصد بدست اومد که عملکرد خیلی بدتر شد !( تعداد اعضای عناصر قطر اصلی هم خیلی کم هستند.)

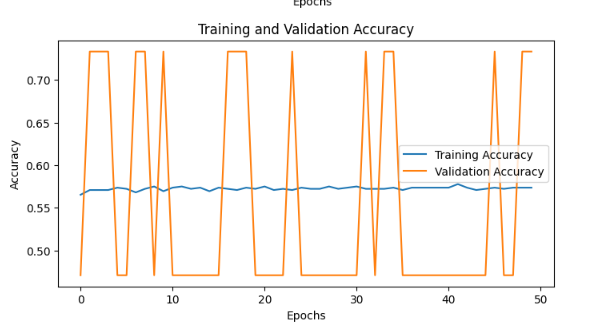
به بررسی حالت دیگر می پردازیم :

حالت دوم ) از توابع زیر استفاده می شود: ( بقیه موارد همان حالت قبل است)

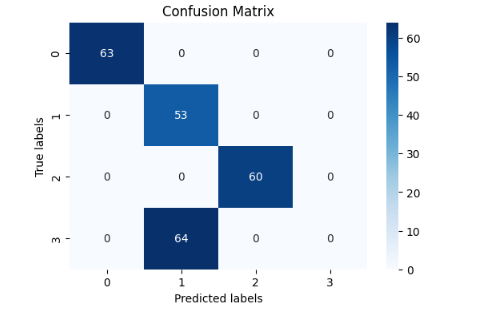
optimizer= RMSprop(), loss='categorical\_hinge'

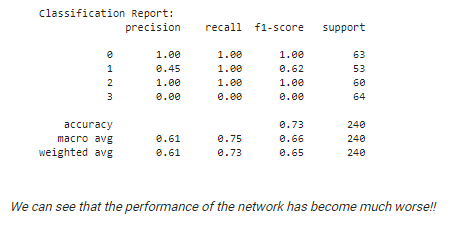
خروجی ها را رسم می کنیم :





عملکرد بد مدل از شکل ها هم مشخص است اما دقیق تر میبینیم :

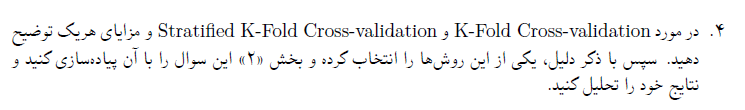




دقت حدود 60 درصد شد و درکل عملکرد مدل طبقه بندی چند کلاسه با این پارامتر ها بسیار بدتر شدند!

همان‌طور که از نتایج برمی‌آید، با عوض کردن تابع هزینه و بهینه ساز مدل، عملکرد مدل تغییر می کند که بر همین اساس می‌توان نتیجه گرفت که انتخاب تابع هزینه و بهینه ساز روی عملکرد مدل تاثیر دارد.

## بخش چهارم



ابتدا روش K-fold cross-validation را توضیح می‌دهیم و ازآنجایی‌که روش Stratified K-fold Cross-validation شباهت زیادی به روش قبلی دارد، بخش متفاوت آن را توضیح می‌دهیم.

K-Fold Cross-validation

K-fold cross-validation یکی از تکنیک‌های اعتبارسنجی مدل‌های یادگیری ماشین است که برای ارزیابی عملکرد یک مدل و اطمینان از عمومی‌سازی (generalization) مناسب آن استفاده می‌شود. در این روش، کل دیتاست به K بخش مساوی تقسیم می‌شود. سپس مدل به تعداد K بار آموزش داده می‌شود. در هر بار آموزش، یکی از این بخش‌ها (foldها) به‌عنوان مجموعه ارزیابی (یا اعتبارسنجی) در نظر گرفته می‌شود و با استفاده از دیگر بخش‌ها مدل آموزش داده می‌شود. یعنی در هر دور آموزش، K-1 مجموعه به عنوان داده آموزش استفاده می‌شود. این فرآیند به صورت زیر است:

1. تقسیم دیتاست: دیتاست به K بخش مساوی (fold) تقسیم می‌شود.

2. آموزش و ارزیابی مدل: برای هر یک از K بخش:

- یکی از بخش‌ها به عنوان مجموعه ارزیابی انتخاب می‌شود.

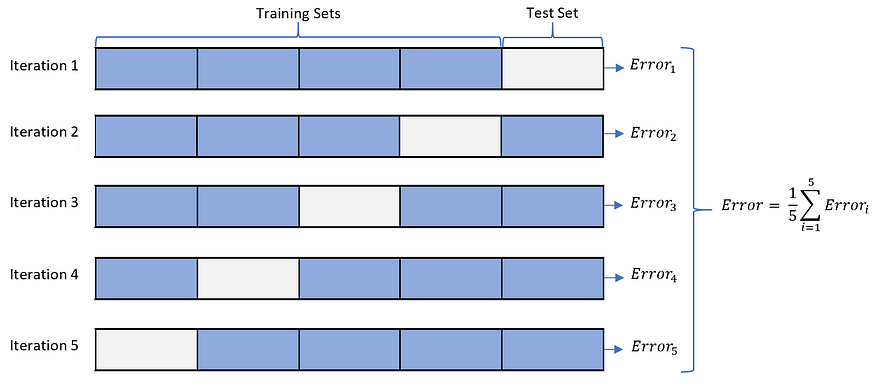
- باقی K-1 بخش به عنوان مجموعه آموزش استفاده می‌شود.

- مدل با استفاده از داده‌های آموزش ساخته می‌شود و سپس بر روی داده‌های ارزیابی تست می‌شود.

3. محاسبه عملکرد: خطا یا عملکرد مدل برای هر یک از K بخش محاسبه و ثبت می‌شود.

4. میانگین‌گیری: در نهایت، میانگین تمامی خطاهای به‌دست‌آمده برای ارائه یک ارزیابی کلی از عملکرد مدل محاسبه می‌شود.

شکل زیر روش k-fold cross-validation را نمایش داده‌است. در هر دور یک fold را به عنوان آزمون انتخاب کرده و دیگران را به عنوان آموزش استفاده می‌کنیم.



اگر قصد داشته باشیم عملکرد مدل‌هایی با ساختارهای مختلف را مقایسه کنیم، از تمامی خطاهای به‌دست‌آمده با استفاده از این روش، میانگین می‌گیریم. این کار باعث می‌شود که ارزیابی ما دقیق‌تر و نسبت به تقسیم‌بندی‌های خاصی از داده‌ها کمتر حساس باشد. اما برای انتخاب مدل نهایی، یکی از K مدل‌های آموزش داده شده را انتخاب می‌کنیم. معمولاً مدلی که بهترین عملکرد را روی داده‌های ارزیابی داشته است، به عنوان مدل نهایی انتخاب می‌شود.

Stratified K-Fold Cross-validation

تفاوتی که این روش با روش قبلی دارد این است که در اینجا، برای هر fold ایجاد شده، تلاش می‌شود که پخش داده موجود، مانند پخش داده کل باشد. یعنی نسبت هر کلاس از داده در هر fold ایجاد شده، با نسبت پخش کلاس‌ها در کل داده یکسان است.

این روش به ویژه در مواردی که داده‌ها دارای کلاسی‌های نامتوازن (imbalanced classes) هستند، مفید است. به عنوان مثال، اگر در دیتاست اصلی 70 درصد از نمونه‌ها به کلاس A و 30 درصد به کلاس B تعلق داشته باشند، در هر fold از روش Stratified K-Fold نیز همین نسبت‌ها حفظ می‌شوند. این کار باعث می‌شود که هر fold نماینده دقیقی از کل دیتاست باشد و عملکرد مدل بهتر و واقعی‌تر ارزیابی شود.

فرآیند Stratified K-Fold Cross-validation به صورت زیر است:

1. تقسیم دیتاست با حفظ نسبت کلاس‌ها : دیتاست به K بخش مساوی تقسیم می‌شود، به طوری که نسبت هر کلاس در هر بخش با نسبت کلاس‌ها در کل دیتاست برابر است.

2. آموزش و ارزیابی مدل: برای هر یک از K بخش:

- یکی از بخش‌ها به عنوان مجموعه ارزیابی انتخاب می‌شود.

- باقی K-1 بخش به عنوان مجموعه آموزش استفاده می‌شود.

- مدل با استفاده از داده‌های آموزش ساخته می‌شود و سپس بر روی داده‌های ارزیابی تست می‌شود.

3. محاسبه عملکرد و میانگین‌گیری: مشابه روش K-Fold Cross-validation، خطاها محاسبه و میانگین گرفته می‌شود.

این روش به ویژه در پروژه‌هایی که داده‌ها به صورت نامتوازن (imbalanced) هستند، دقت ارزیابی مدل را افزایش می‌دهد و از ایجاد بایاس‌های ناشی از عدم تعادل کلاس‌ها جلوگیری می‌کند.

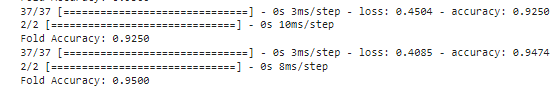
پس در کل K-Fold Cross-validation یک روش موثر برای ارزیابی مدل‌ها است که با استفاده از تقسیم بندی چندگانه دیتاست به بخش‌های مساوی، عملکرد مدل را بر روی داده‌های مختلف بررسی می‌کند. Stratified K-Fold Cross-validation نیز با حفظ نسبت کلاس‌ها در هر بخش، بهبود دقت ارزیابی را در داده‌های نامتوازن ارائه می‌دهد.

در این جا چون داده ها متعادل هستند از kfold استفاده می کنیم:

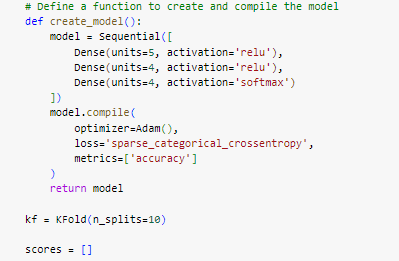
از کد زیر استفاده می شود:

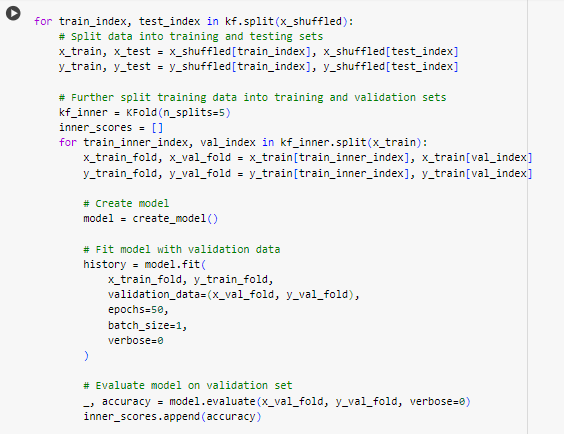


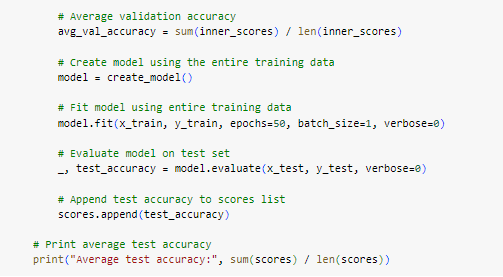
به مقدار دقت 92 درصد رسیده ایم.



یک روش دیگر هم استفاده از کد زیر است که دقت داده های تست را در نهایت 93 درصد داده است:







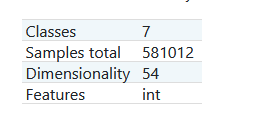


# سوال سه

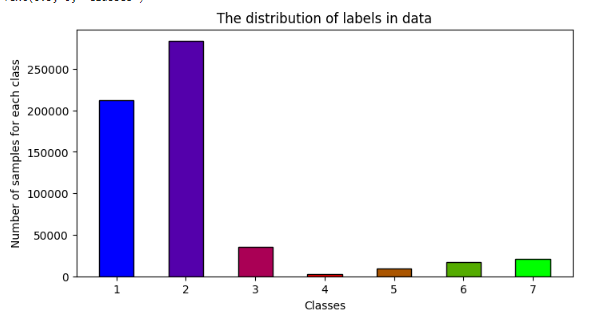
در این سؤال از داده‌های پوشش جنگلی استفاده کردیم. این داده‌ها در کتابخانه scikit-learn موجود هستند و برای دانلود این داده‌ها از دستور fetch-covtype استفاده شد.



داده ها به صورت زیر هستند : که در سایت اطلاعات بیشتری از آن ها موجود است.

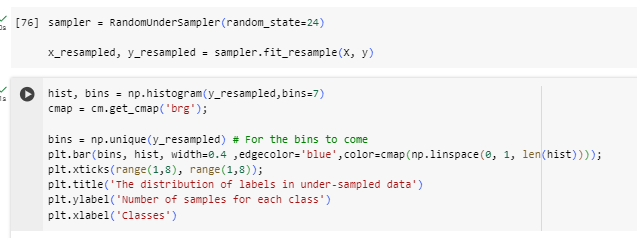


داده ها در 7 کلاس هستند، حال باید بررسی کنیم که وضعیت پخش داده ها در این دیتاست، چگونه است در واقع در هر کلاس چند نمونه وجود دارد!

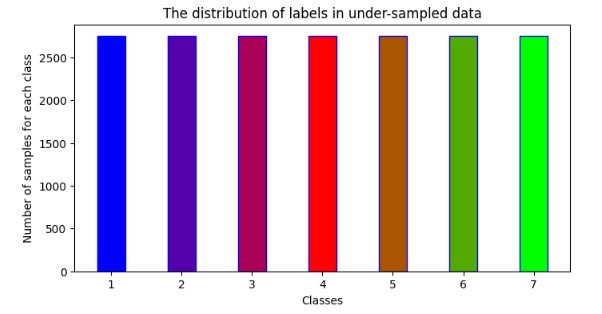


میبینیم که تعداد داده ها در هر کلاس خیلی پراکنده است و متعادل نیستند، این باعث می شود که مدل ما عملکرد خوبی نداشته باشد. مثلا داده ها در یک کلاس کمتر از 50000 تا و در کلاسی دیگر بیشتر از 250000 تا می باشند! اگر این عدم تعادل در داده‌های آموزشی نیز وجود داشته باشد، مدل ایجاد شده به طور قابل‌توجهی به سمت کلاسی که بیشترین تعداد داده را دارد متمایل خواهد شد (دارای bias می‌شود) و عملکرد مطلوبی نخواهد داشت. برای رفع این مشکل و متعادل کردن این دیتاست، از روش Under-sampling استفاده می‌کنیم.

این روش شامل کاهش تعداد نمونه‌ها از کلاسی است که بیشترین تعداد داده را دارد تا تعداد نمونه‌های آن با کلاس‌های دیگر برابر شود. به این ترتیب، مدل ما می‌تواند به‌طور یکسانی از همه کلاس‌ها یاد بگیرد و تعادل بهتری در پیش‌بینی‌ها داشته باشد. البته باید توجه داشت که Under-sampling ممکن است منجر به از دست دادن اطلاعات مهم شود، بنابراین انتخاب دقیق نمونه‌ها برای حذف بسیار حائز اهمیت است. در برخی موارد، ترکیب روش‌های دیگری مانند Over-sampling یا استفاده از الگوریتم‌های مقاوم به عدم تعادل داده نیز می‌تواند موثر باشد. برای حل این مشکل از کد زیر استفاده می کنیم :



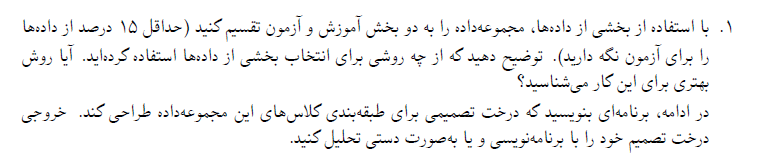
خروجی را مشاهده می کنیم :



پس از اعمال روش Under-sampling، تعداد نمونه‌ها در هر کلاس یکسان شده و دیتاست متعادل می‌شود. با این حال، این روش منجر به کاهش تعداد کل داده‌ها می‌شود. به عبارت دیگر، برای متعادل کردن دیتاست، مجبور به حذف برخی از نمونه‌های کلاس‌هایی که بیشترین تعداد را دارند هستیم.

پس اکنون 7 کلاس داریم که هر کلاس شامل حدود 2700 نمونه است. به سراغ ادامه حل سوال می رویم ...

## بخش اول



برای تقسیم‌بندی داده‌ها از روش نمونه‌برداری تصادفیrandom sampling) ) استفاده می‌کنیم. تابع مورد استفاده برای این تقسیم‌بندی، train\_test\_split نام دارد. در این روش، داده‌های آموزشی به صورت کاملاً تصادفی انتخاب می‌شوند. در این بخش، ۱۵ درصد از داده‌ها به مجموعه آزمون اختصاص داده شده و باقی داده‌ها به عنوان مجموعه آموزش استفاده می‌شوند.

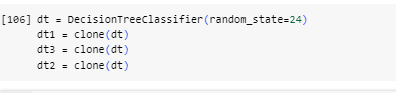
علاوه بر نمونه‌برداری تصادفی، دو استراتژی دیگر نیز برای انتخاب داده‌های آموزشی و آزمونی وجود دارد:

۱. اعتبارسنجی متقابل Cross Validation) ) : در این روش، دیتاست به چندین بخش تقسیم می‌شود و مدل به تعداد بخش‌های تقسیم شده آموزش داده می‌شود. سپس بهترین مدل از میان این مدل‌ها انتخاب می‌شود. هنگامی که از این روش برای مقایسه الگوریتم‌های مختلف یا مدل‌هایی با فراپارامترهای مختلف استفاده می‌شود، خطای تمامی مدل‌ها محاسبه و میانگین گرفته می‌شود تا الگوریتم‌ها بر اساس این میانگین با هم مقایسه شوند. این روش باعث می‌شود ارزیابی مدل‌ها دقیق‌تر باشد و از همه داده‌ها به طور مؤثرتری استفاده شود.

۲. بوت‌استرپ Bootstrap) ) : این روش مبتنی بر انتخاب زیرمجموعه‌هایی از دیتاست اصلی با جایگذاری است. به این معنا که از یک دیتاست اصلی، چندین زیردیتاست ایجاد می‌شود که هر کدام با قابلیت جایگذاری داده‌ها انتخاب می‌شوند. یعنی برای انتخاب داده‌ها در هر زیردیتاست، ممکن است برخی داده‌ها چندین بار انتخاب شوند و برخی اصلاً انتخاب نشوند. این روش به ارزیابی مدل‌ها کمک می‌کند تا با تنوع بیشتری از داده‌ها مواجه شوند و برآورد دقیق‌تری از عملکرد مدل بر روی داده‌های نادیده به دست آید.

استفاده از هر کدام از این روش‌ها به هدف و نوع پروژه بستگی دارد و می‌تواند تأثیر زیادی بر عملکرد و دقت نهایی مدل داشته باشد. عموماً، این روش‌ها زمانی به کار گرفته می‌شوند که تعداد داده‌های آموزشی کم باشد، اما اگر پس از کاهش تعداد داده‌ها under-sampling) ) به اندازه‌ای مناسب (مثلاً 2700 داده برای هر کلاس) برسیم، نیازی به استفاده از دو روش دیگر نداریم.

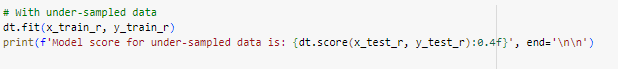
* در فرآیند آموزش مدل درخت تصمیم‌گیری، چندین روش مختلف برای آموزش مدل مقایسه شد.(4 روش) پس در ابتدا :



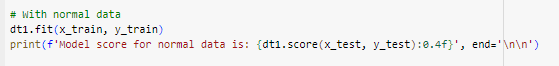
این کد برای ایجاد چهار مدل درخت تصمیم‌ استفاده می‌شود. توابع clone() برای ایجاد نسخه‌های مستقل از یک مدل موجود استفاده می‌شود تا هر مدل مستقل از دیگری باشد و تغییرات در یک مدل تأثیری بر سایر مدل‌ها نگذارد. در اینجا، dt1، dt2، و dt3 به ترتیب سه نسخه از مدل اصلی DecisionTreeClassifier با همان تنظیمات و تصمیمات مولد تولید می‌شوند.

از میان این روش‌ها، بهترین آن‌ها برای مراحل بعدی انتخاب شدند. این روش‌ها شامل موارد زیر بودند:

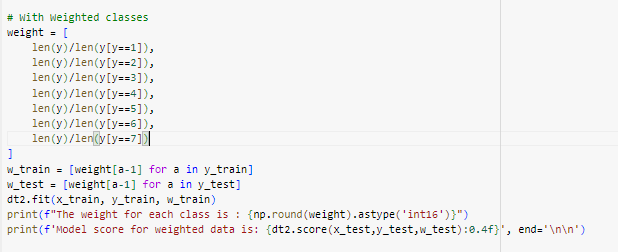
1. آموزش داده‌هایی که تحت عملیات under-sampling قرار گرفته‌اند؛ به این معنا که تعداد نمونه‌های هر کلاس را به حدی کاهش داده‌ایم که تعادل بین کلاس‌ها برقرار شود.



2. آموزش داده‌ها بدون هیچ کار و عملیات اضافه‌ای؛ به عبارت دیگر، استفاده از داده‌های اصلی بدون تغییر در توزیع یا تعداد آن‌ها.

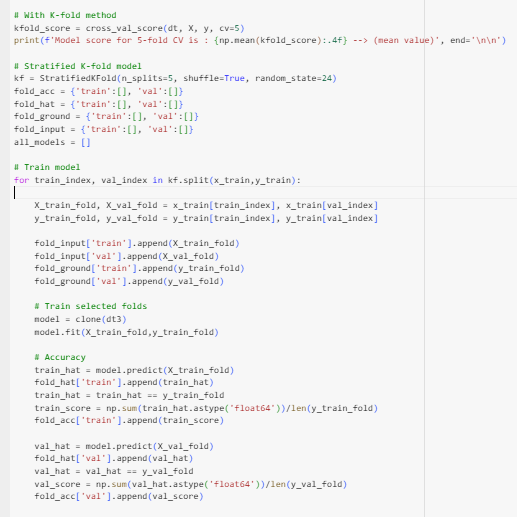


3. آموزش داده‌ها با اعمال وزن به الگوریتم؛ به این معنا که وزن‌هایی به هر کلاس اختصاص داده شده و در فرآیند آموزش مورد استفاده قرار گرفته‌اند، به‌طوری‌که تأثیر هر کلاس بر مدل متوازن باشد.



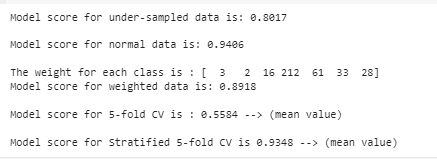
4. آموزش داده‌ها با استفاده از روش ۵-fold Cross-validation؛ به این معنا که داده‌ها به ۵ بخش تقسیم شده و مدل به‌طور متوالی بر روی ۴ بخش آموزش داده می‌شود و بر روی یک بخش دیگر آزمایش می‌شود، و این فرآیند برای ۵ بار تکرار می‌شود.

5. آموزش داده‌ها با استفاده از روش Stratified ۵-fold Cross-validation؛ که علاوه بر انجام ۵-fold Cross-validation، تضمین می‌کند که توزیع کلاس‌ها در هر بخش حفظ شود.



این مقایسه‌ها به منظور انتخاب بهترین روش آموزش برای مدل درخت تصمیم‌گیری انجام شد.

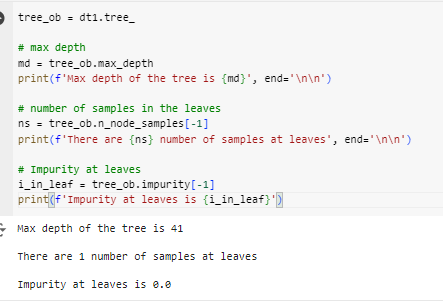
نتایج حاصل از این روش ها را در ادامه خواهیم دید :



اگرچه عملکرد مدل‌ها برای کلاس‌های مختلف متفاوت است و هیچ مدلی به طور کامل از بقیه عملکرد بهتری ندارد، اما به طور کلی می‌توانیم بر اساس عملکرد کلی، حالت دوم را انتخاب کنیم چرا که دقت حدود 94 درصد دارد که بهتر از بقیه حالت هاست. به عبارت دیگر، استفاده از داده‌های اصلی بدون هیچ وزنی، نتیجه‌ای بهتر و مطلوب تر را در کلیت مدل ارائه می‌دهد. بنابراین، برای ادامه محاسبات، تصمیم گرفته شده است که از تمامی داده‌ها بدون در نظر گرفتن وزن‌دهی استفاده شود. البته حالت پنجم هم عملکرد نزدیک به این مدل دارد ولی از حالت دوم استفاده می کنیم ...

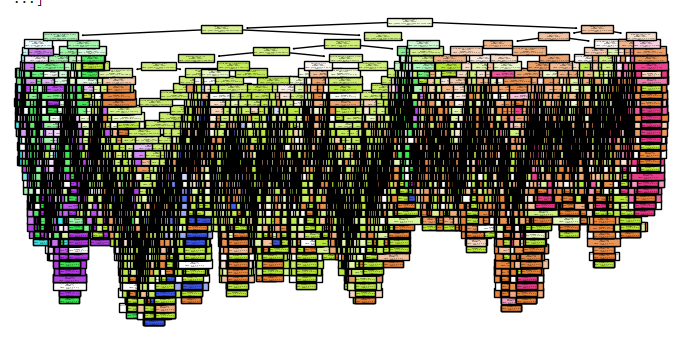
اکنون اطلاعات اساسی درخت را استخراج می کنیم :

ابتدا، با استفاده از متغیر dt1.tree\_، اطلاعات مربوط به درخت را به دست می‌آوریم. سپس، سه ویژگی اصلی را استخراج می‌کنیم: ۱) Max depth of the tree که عمق بیشترین مسیر در درخت را نمایش می‌دهد، ۲) Number of samples at leaves که تعداد نمونه‌های موجود در برگ‌های درخت را نشان می‌دهدو ۳) Impurity at leaves که میزان اصطکاک یاImpurity در برگ‌های درخت را نشان می‌دهد. این اطلاعات اساسی معمولاً برای ارزیابی و فهم بهتر عملکرد مدل درخت تصمیم استفاده می‌شوند که به صورت زیر هستند :



شکل کلی درخت به صورت زیر است:

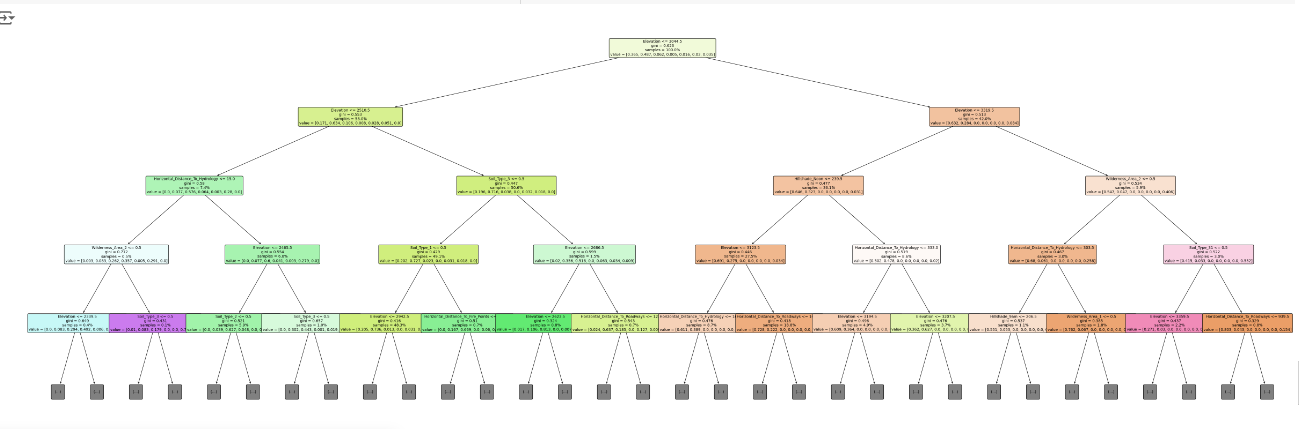


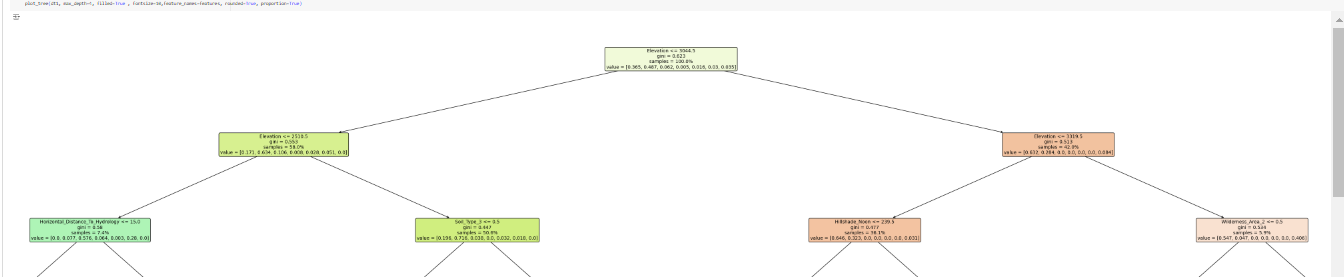


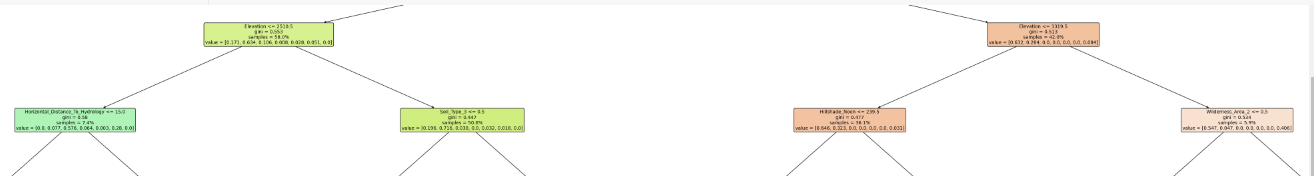
اما از کد زیر استفاده می کنیم:



در واقع محدود تر می کنیم و به صورت کامل رسم نمی کنیم( به علت پیچیدگی محاسبات و سخت ران شدن مدل)

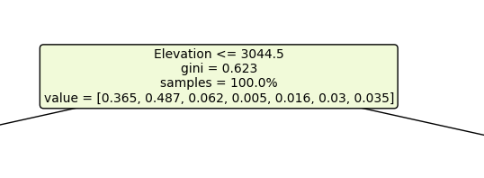
و درخت به فرم زیر می شود :

در واقع لایه اول درخت به صورت زیر است :

تقسیم بندی برای لایه دوم درخت به صورت زیر است :

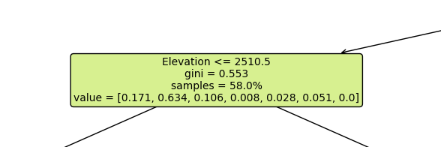
در هر بلوک اطلاعاتی مانند نام ویژگی، مقدار threshold، مقدار gini و تعداد نمونه‌‌های موجود در آن گره به نمایش درآمده‌اند.

مانند : (اولین بلوک)



که در تصویر بالا ، value یک آرایه با7 عنصر عددی است که نشان دهنده فراوانی یا احتمال هر کلاس/برچسب در این گره از درخت است. مجموع این عناصر برابر با 1 است. به عنوان مثال، اگر برچسب های داده ها [0، 1] باشند، اولین دو عنصر ممکن است فراوانی کلاس 0 و 1 در این گره باشند. elevation=3044.5 یک معیار تصمیم گیری برای جدا کردن این گره از گره قبلی براساس یک ویژگی خاص است. برای مثال، اگر ویژگی پیشگو، درآمد باشد، بالای 3044.5 دلار درآمد به سمت راست درخت و کمتر از آن به سمت چپ درخت قرار می گیرد. gini=0.623 یک شاخص ناهمگنی داده ها در این گره است که بین 0 (کاملاً همگن) و 0.5 (کاملاً ناهمگن) است. یک مقدار gini بالا نشان می دهد که داده ها در این گره متنوع هستند. samples=100.0% نشان می دهد که همه نمونه های داده در این گره از درخت قرار گرفته اند.

بلوک بعدی سمت چپ:



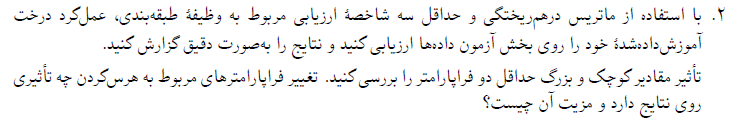
58 درصد نمونه ها در این گره هستند. لایه بعدی این بلوک به صورت زیر است:

از اون 58 درصد، در این دولایه به نسبت 7.4 و 50.6 تقسیم شدند. این روال تا انتها ادامه می باید.

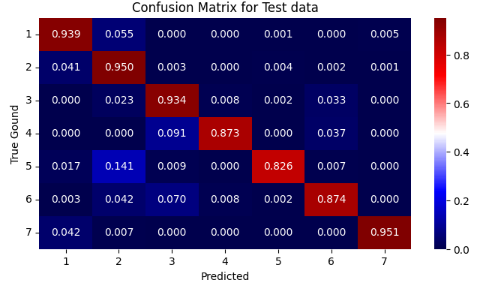
این اطلاعات برای درک چگونگی ساخت، اعتبارسنجی و ارزیابی عملکرد یک مدل درخت تصمیم در مسائل دسته بندی یا پیش بینی استفاده می شوند.

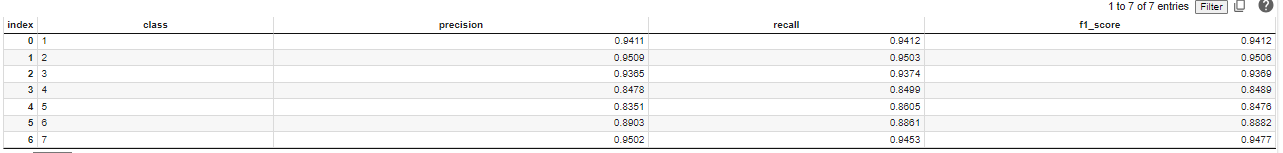
این درخت آموزش‌داده‌شده تا 43 سطح عمق جداکنندگی داشته و در هر برگ آن، تنها ۲ نمونه وجود دارد. از این اطلاعات، می‌توان چند نکته مهم را استخراج کرد: اولاً، به دلیل بزرگ بودن تعداد سطوح این درخت، امکان نمایش همه گره‌ها و برگ‌های آن به صورت تصویر یا متن وجود ندارد. دوماً، این مدل به شدت دچار overfitting شده است؛ به عبارت دیگر، بیش از حد به داده‌های آموزش خود برازش شده و احتمالاً نمی‌تواند با داده‌های جدید به خوبی کار کند.

## بخش دوم



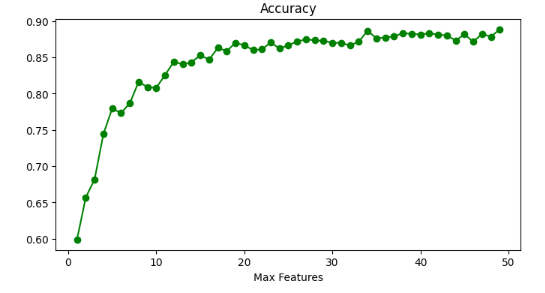
ماتریس درهم ریختگی درخت تصمیم را برای داده های تست به صورت زیر نمایش می دهیم :

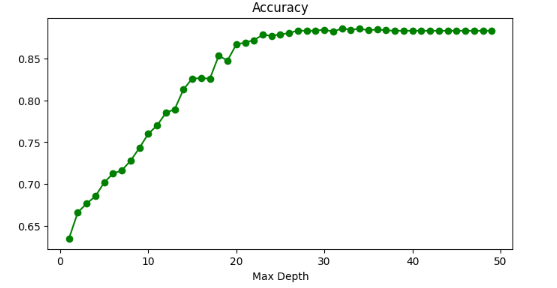


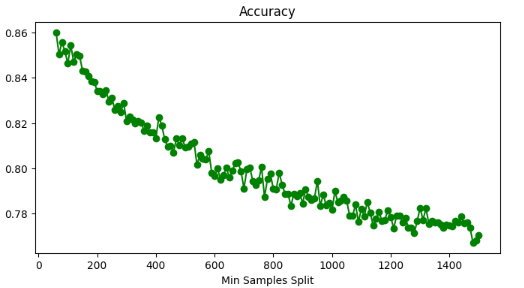
از سه معیار دیگر برای بررسی مدل استفاده می شود :

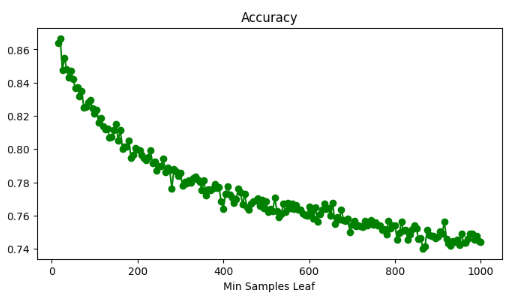
* **بررسی هایپرپارامتر ها**

در ادامه توسط کد پایتون یک آنالیز برای هایپرپارامترهای مدل Decision Tree انجام می‌دهیم. این آنالیز از متغیرهای مختلفی مانند حداکثر عمق درخت max\_depth، تعداد نمونه‌های لازم برای تقسیم یک گره min\_samples\_split، تعداد نمونه‌های لازم برای یک برگ min\_samples\_leaf و تعداد ویژگی‌های مجاز برای جستجو در هر تقسیم max\_features را در نظر می‌گیر و به طور کلی یک حلقه برای هر پارامتر استفاده می‌کند تا مقادیر مختلف آن را بررسی کند و دقت مدل را برای هر مقدار از هر پارامتر ذخیره کند. سپس، نمودارهایی از دقت مدل بر حسب مقادیر هر پارامتر رسم می‌شود. به عنوان مثال، نمودار "Accuracy vs Max Depth" نشان می‌دهد که چطور دقت مدل تغییر می‌کند در واکنش به افزایش عمق درخت. همچنین دیگر نمودارها هم نشان‌دهنده تأثیر تغییر هر پارامتر بر دقت مدل هستند. در ادامه نتایج را مشاهده می کنیم :









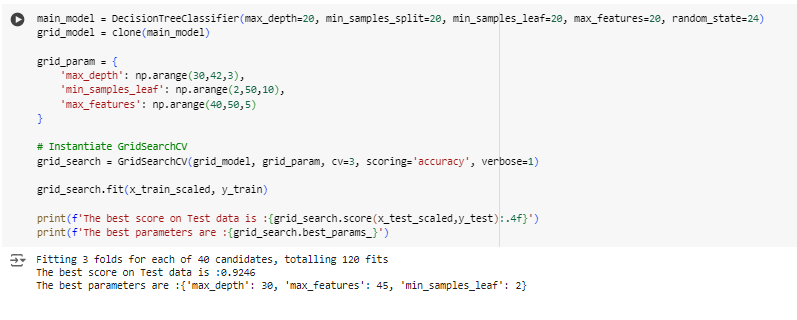
وقتی مدل را با محدودیت‌هایی مثل حداکثر عمق درخت max depth، تعداد نمونه‌های لازم برای تقسیم یک گره min samples split، و تعداد نمونه‌های لازم برای یک برگ min samples leaf محدود می‌کنیم، این محدودیت‌ها می‌توانند از بیش‌برازش یا overfitting جلوگیری کنند، یعنی مدل به جای یادگیری الگوهای واقعی در داده‌ها، الگوهاینمونه های تصادفی را هم یاد بگیرد.

نکته‌ای که در اینجا مطرح شده این است که حداکثر عمق درخت یا max depth می‌تواند تأثیر متفاوتی بر روی دقت مدل داشته باشد. از طرفی، نمودار آن نشان می‌دهد که با افزایش عمق درخت، دقت مدل بر روی داده‌های آزمون افزایش می‌یابد، اما در عین حال، با محدود کردن دقت مدل، از بیش‌برازش جلوگیری می‌شود.

با استفاده از نمودارهایی که عملکرد مدل برای مقادیر مختلف hyperparameterها را نشان می‌دهند، می‌توان انواع مختلفی از محدودیت‌ها را در نظر گرفته و با استفاده از روش‌های مثل GridSearch، بهترین مقادیر این hyperparameterها را برای مدل انتخاب کرد. این کار معمولاً بهبود عملکرد مدل را در پیش‌بینی داده‌های جدید و ناشناخته بهبود می‌بخشد.

GridSearch:

از کد زیر استفاده می کنیم :

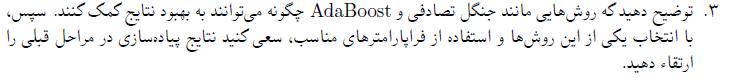
در این کد GridSearch با استفاده از پارامترها، مدل را با استفاده از اعتبارسنجی متقاطع (cv=3) و بر اساس معیار دقت (scoring='accuracy') بهینه می‌کند و سپس مدل بهینه شده با داده‌های آموزشی ،آموزش داده می‌شود و بهترین نمره و پارامترهای بهینه یافته بر روی **داده‌های تست چاپ می‌شود.**

**The best score on Test data is :0.9246**

با توجه به کد بالا دقت حدود 92 درصد می شود، به ازای هایپر پارامتر های مناسب که به صورت زیر هستند:

**The best parameters are :{'max\_depth': 30, 'max\_features': 45, 'min\_samples\_leaf': 2}**

## بخش سوم



روش‌های AdaBoost و RandomForest از دسته روش‌های ensemble learning هستند. این روش‌ها از مجموعه‌ای از مدل‌ها برای بهبود دقت و پایداری پیش‌بینی‌ها استفاده می‌کنند. به طور کلی، ensemble learning با استفاده از ایجاد زیرمجموعه‌هایی از مجموعه داده اصلی و جایگشت داده، مدل‌های مختلف و مستقلی تولید می‌کند. این فرآیند به دست آوردن مدل‌هایی با ویژگی‌های مکمل و ترکیب آن‌ها منجر می‌شود که خاصیت‌های زیر را به همراه دارد:

کاهش واریانس (جلوگیری از overfitting)

یکی از مزایای اصلی استفاده از روش‌های ensemble learning، کاهش واریانس مدل است. در مدل‌های تک‌تک(single models)، مانند درخت تصمیم، احتمال overfitting بالاست؛ به این معنی که مدل به جای یادگیری الگوهای کلی داده، به جزئیات و نویز‌های داده تمرکز می‌کند. با ترکیب چندین مدل در روش‌های ensemble، نتایج نهایی به میانگین نتایج مدل‌های مختلف نزدیک می‌شود. این امر باعث می‌شود که تأثیر نویز‌ها و جزئیات غیرمفید کاهش یابد و در نتیجه، مدل کلی عملکرد بهتری بر روی داده‌های جدید داشته باشد.

Generalization

روش‌های ensemble با ایجاد و ترکیب مدل‌های مختلف، توانایی تعمیم‌دهی را افزایش می‌دهند. هر مدل به صورت جداگانه بخش‌های مختلفی از داده را می‌آموزد و با ترکیب این مدل‌ها، توانایی شناسایی الگوهای متنوع و پیچیده‌تر در داده‌ها فراهم می‌شود. به عبارت دیگر، ترکیب چندین مدل به کشف و پوشش دادن جوانب مختلفی از توزیع آماری موجود در داده کمک می‌کند که باعث بهبود توانایی پیش‌بینی مدل نهایی می‌شود.

تأثیر کمتر نویز روی مدل

از آنجایی که مدل‌های individual در روش‌های ensemble بر روی زیرمجموعه‌های مختلفی از داده‌ها آموزش داده می‌شوند، تأثیر نویز در داده‌های آموزشی کاهش می‌یابد. این روش‌ها با درنظرگرفتن عملکرد تمامی مدل‌ها، ناهنجاری‌ها و داده‌های نویزی را تضعیف می‌کنند. در نتیجه، مدل نهایی از پایداری و دقت بالاتری برخوردار است و بهتر می‌تواند الگوهای اصلی داده را شناسایی کند.

افزایش دقت مدل

یکی دیگر از مزایای مهم استفاده از روش‌های ensemble learning، افزایش دقت مدل است. ترکیب مدل‌های مختلف با رویکردهایی مثل میانگین‌گیری ( bagging) یا تقویت (boosting)، معمولاً منجر به مدل‌هایی با دقت بالاتر می‌شود. این افزایش دقت به دلیل ترکیب قدرت پیش‌بینی مدل‌های مختلف و جبران نقاط ضعف هر مدل توسط مدل‌های دیگر است.

انعطاف‌پذیری و تطبیق‌پذیری

روش‌های ensemble learning به دلیل توانایی استفاده از مدل‌های مختلف و ترکیب آن‌ها، انعطاف‌پذیری بالایی دارند. این روش‌ها می‌توانند با انواع مختلف داده‌ها و مسائل یادگیری تطبیق پیدا کنند و بهبود عملکرد قابل توجهی را در مسائل مختلف به همراه داشته باشند.

مقابله با مشکلات کلاس‌بندی نامتعادل

در مسائل دسته‌بندی با کلاس‌های نامتعادل، روش‌های ensemble می‌توانند با تخصیص وزن‌های مختلف به کلاس‌های مختلف یا با استفاده از تکنیک‌های خاصی مانند oversampling و undersampling، عملکرد بهتری را نشان دهند و تعادل بهتری بین کلاس‌ها برقرار کنند.

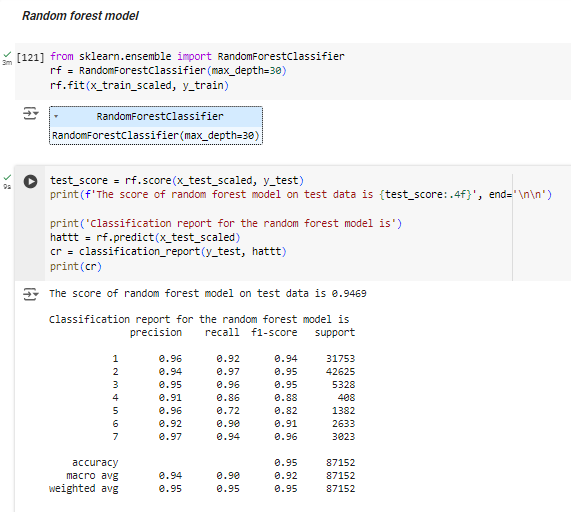
روش‌های AdaBoost و RandomForest به عنوان نمونه‌های برجسته‌ای از روش‌های ensemble، نشان‌دهنده قدرت و مزایای این رویکردها هستند. AdaBoost با تقویت مدل‌های ضعیف‌تر و تاکید بر داده‌هایی که به درستی طبقه‌بندی نشده‌اند و RandomForest با ایجاد مجموعه‌ای از درختان تصمیم مستقل و ترکیب نتایج آن‌ها، هر دو بهبود دقت و پایداری مدل نهایی را تضمین می‌کنند. این روش‌ها به‌ویژه در مسائل پیچیده و داده‌های نویزی یا نامتعادل بسیار مؤثر هستند.

برای این بخش از الگوریتم جنگل تصادفی استفاده می‌کنیم و نتایج را با مدل بخش قبل مقایسه می‌نماییم.

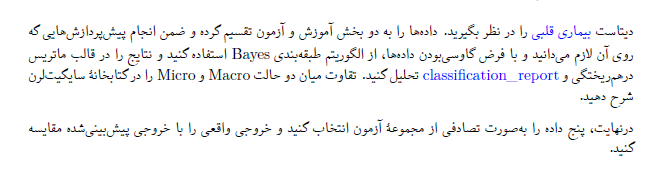
کد این قسمت به صورت زیر است و مقدار max-depth را برابر 30 در نظر می گیریم.

دقت حدود 94-95 درصد بدست می آید و میبینیم که عملکرد نهایی مدل جنگل تصادفی نسبت به درخت تصمیم‌گیری، بهبود یافته است.

**The score of random forest model on test data is 0.9469**



# سوال چهار

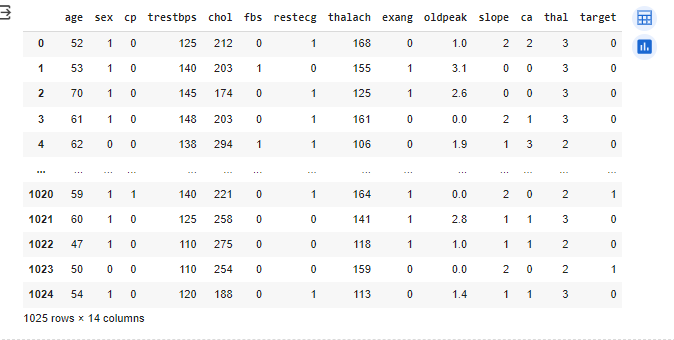


در ابتدا دیتاست بیماری قلبی را از Kaggle دانلود می کنیم.

درباره مجموعه داده‌ها می توان گفت : این مجموعه داده از سال ۱۹۸۸ تاریخ دارد و شامل چهار پایگاه داده است: کلیولند، مجارستان، سوئیس و لانگ بیچ. این مجموعه شامل ۷۶ ویژگی است که شامل ویژگی پیش‌بینی شده است. اما تمام آزمایشات منتشر شده به استفاده از یک زیرمجموعه از ۱۴ ویژگی اشاره دارند. هدف این مجموعه داده تشخیص بیماری قلبی در بیمار است. فیلد "target" به وجود بیماری قلبی در بیمار اشاره دارد. این فیلد دارای مقادیر صحیح است: ۰ = بدون بیماری و ۱ = بیماری

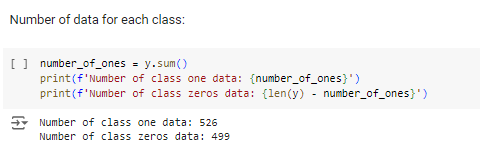
۱۴ ویژگی استفاده شده در مجموعه داده عبارتند از: سن، جنسیت، نوع درد سینه، فشار خون استراحتی، کلسترول سرمی، قند خون، نتایج الکتروکاردیوگرافی استراحتی، بیشینه ضربان قلب، آنژین تحریکی، زیرفشار، شیب قله اس‌تی تمرین، تعداد عروق اصلی، تال (0= نرمال; 1 = نقص ثابت؛ 2 = نقص قابل برگشت) و ویژگی پیش‌بینی شده (بیماری قلبی).

در نمایی کلی دیتاست به صورت زیر است :



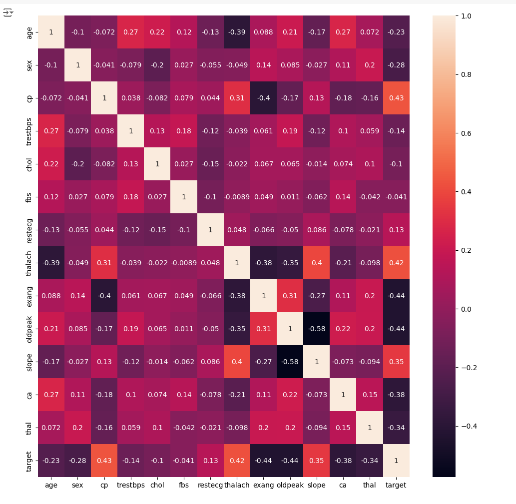
این دیتاست شامل 1025 نمونه و 13 فیچر است و ستون target هم لیبل آن است. پس لیبل به عنوان y و 13 فیچر به عنوان x در نظر گرفته می شوند.

در گام بعد چک می کنیم ببینیم تعداد داده های مربوط به هر کلاس چند تاست :

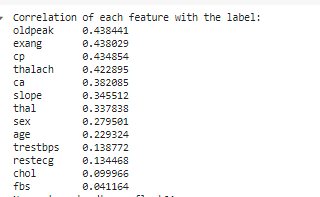


که به نسبت خوبی تقسیم شده اند.

حالا هم بستگی بین این فیچر هارا بررسی می کنیم تا ببینیم چه ارتباطی با هم دارند، درواقع ما لازم نیست که از هر 13 فیچر برای شبکه استفاده کنیم و می تونیم از مناسب ترین فیچر ها استفاده کنیم.



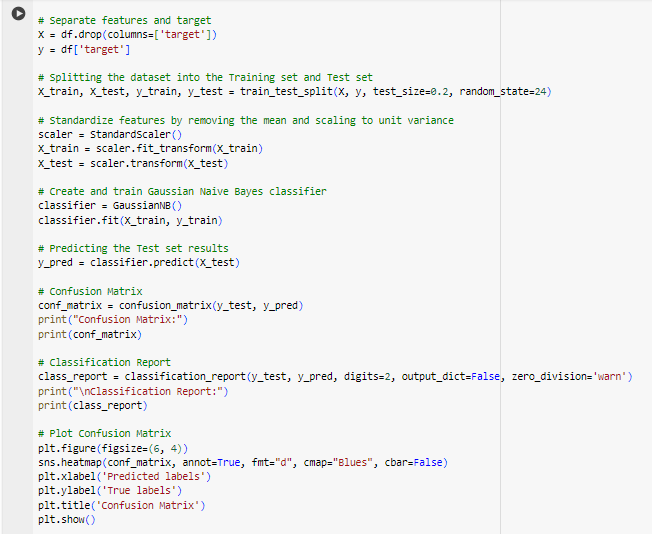
به صورت بالا همبستگی فیچر ها مشخص میشود. در واقع در زیر واضح تر است :



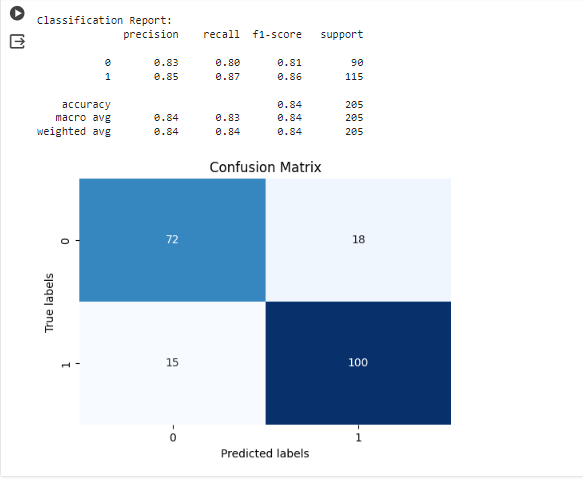
حالا در چند حالت بررسی می کنیم.

یک بار از تمام فیچر ها استفاده می کنیم :

اکنون از الگوریتم Bayes استفاده می کنیم و کد آن به صورت زیر است :



و نتایج دو فرم خواسته شده به صورت زیر است :



* **نتایج به دست آمده را به صورت زیر تحلیل می کنیم :**

***ماتریس درهم ریختگی :***

- مثبت واقعی (TP) : 72 (کلاس ۰ واقعی که به درستی به عنوان کلاس ۰ پیش‌بینی شده است)

- مثبت غلط (FP) : 18 (کلاس ۰ واقعی که به اشتباه به عنوان کلاس ۱ پیش‌بینی شده است)

- منفی غلط (FN) : 15 (کلاس ۱ واقعی که به اشتباه به عنوان کلاس ۰ پیش‌بینی شده است)

- منفی واقعی (TN) : 100 (کلاس ۱ واقعی که به درستی به عنوان کلاس ۱ پیش‌بینی شده است)

***classification\_report***

شامل چند بخش است که در ادامه به توضیح آن می پردازیم :

- دقت (Precision): نسبت مثبت‌های صحیحی که به درستی تشخیص داده شدند به کل مثبت‌هایی که پیش‌بینی شده‌اند است. برای کلاس ۰، دقت ۰.۸۳ است که به این معنی است که ۸۳٪ از مثبت‌هایی که به عنوان کلاس ۰ پیش‌بینی شدند، واقعاً کلاس ۰ بودند. برای کلاس ۱، دقت ۰.۸۵ است که نشان می‌دهد ۸۵٪ از مثبت‌هایی که به عنوان کلاس ۱ پیش‌بینی شدند، واقعاً کلاس ۱ بودند.

- بازخوانی (Recall): بازخوانی (همچنین به عنوان حساسیت شناخته می‌شود) نسبت مثبت‌های صحیحی که به درستی تشخیص داده شدند به کل مثبت‌ها در کلاس واقعی است. برای کلاس ۰، بازخوانی ۰.۸۰ است که به این معنی است که ۸۰٪ از کل مثبت‌های واقعی کلاس ۰ به درستی به عنوان کلاس ۰ تشخیص داده شدند. برای کلاس ۱، بازخوانی ۰.۸۷ است که نشان می‌دهد ۸۷٪ از کل مثبت‌های واقعی کلاس ۱ به درستی به عنوان کلاس ۱ تشخیص داده شدند.

- امتیاز F1: امتیاز F1 میانگین هندسی دقت و بازخوانی است. این امتیاز تعادلی بین دقت و بازخوانی ارائه می‌دهد. برای کلاس ۰، امتیاز F1 ۰.۸۱ است و برای کلاس ۱، ۰.۸۶ است.

- کلاس ۰: ۲ \* (دقت \* بازخوانی) / (دقت + بازخوانی) = ۲ \* (۰.۸۰ \* ۰.۸۳) / (۰.۸۰ + ۰.۸۳) = ۰.۸۱

- کلاس ۱: ۲ \* (دقت \* بازخوانی) / (دقت + بازخوانی) = ۲ \* (۰.۸۷ \* ۰.۸۵) / (۰.۸۷ + ۰.۸۵) = ۰.۸۶

- پشتیبانی (Support): پشتیبانی تعداد وقوع واقعی کلاس در مجموعه داده مشخص شده است. برای کلاس ۰، پشتیبانی ۹۰ و برای کلاس ۱، ۱۱۵ است.

- دقت کل (Accuracy): دقت نسبت مثبت‌های صحیحی که به درستی پیش‌بینی شده‌اند به کل نمونه‌ها است. دقت کل ۰.۸۴ است که نشان می‌دهد مدل ۸۴٪ از نمونه‌ها را به درستی پیش‌بینی کرده است.

- میانگین ماکرو (Macro avg): این میانگین را محاسبه می‌کند که میانگین غیروزن‌دار دقت و بازخوانی برای همه کلاس‌ها است. برای دقت، بازخوانی و امتیاز F1، این مقدار ۰.۸۴ است.

- میانگین وزنی (Weighted avg): این میانگین را محاسبه می‌کند که میانگین وزن‌دار دقت، بازخوانی و امتیاز F1 برای همه کلاس‌ها است، با وزن‌دهی توسط پشتیبانی. برای دقت، بازخوانی و امتیاز F1، این مقدار همچنین ۰.۸۴ است.

پس ، با دقت، بازخوانی و امتیاز F1 مناسب برای هر دو کلاس ، نشان می‌دهد که مدل به خوبی عمل کرده است.

در نتیجه می توان گفت که :

- ماتریس در هم ریختگی تفکیک دقیقی از تعداد مثبت‌های واقعی، منفی‌های واقعی، مثبت‌های غلط و منفی‌های غلط را ارائه می‌دهد.

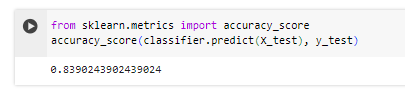
- گزارش دسته‌بندی (classification\_report) دقت، بازخوانی، امتیاز F1 ، پشتیبانی و همچنین میانگین ماکرو و وزن‌دهی شده برای همه کلاس‌ها را ارائه می‌دهد.

- دقت، بازخوانی و امتیاز F1 از گزارش دسته‌بندی می‌تواند با استفاده از مقادیر ماتریس ابهام محاسبه شود.

- دقت از گزارش دسته‌بندی همان مقدار محاسبه شده از ماتریس درهم ریختگی است.

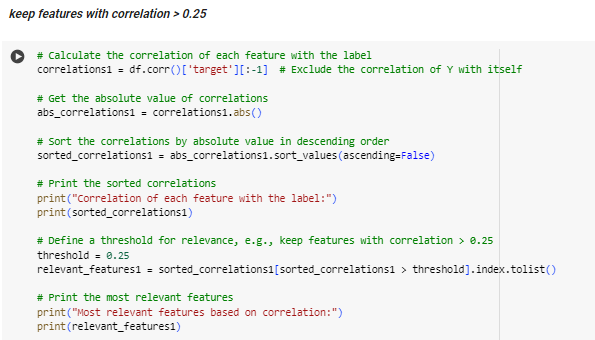
به طور کلی، هر دو ماتریس درهم ریختگی و گزارش دسته‌بندی به مشاهده‌ی مشابهی درباره‌ی عملکرد دسته‌بندی می‌انجامند و تأیید می‌کنند که مدل با دقت، بازخوانی و امتیاز F1 خوب برای هر دو کلاس عمل کرده است.

دقت این مدل هم به صورت زیر است : حدود 84 درصد



این بار تلاش می کنیم تا فیچر های مناسب را استخراج کنیم :

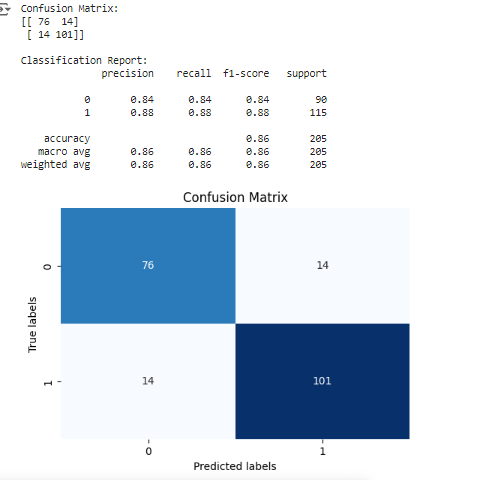
با استفاده از کد زیر، فیچر هایی که لیبل بیشتر از 0.25 هم بستگی دارند را نگه می داریم.



8 فیچر از 13 فیچر باقی می ماند :



حالا این 8 فیچر را به عنوان x به مدل می دهیم و خروجی را بررسی می کنیم :



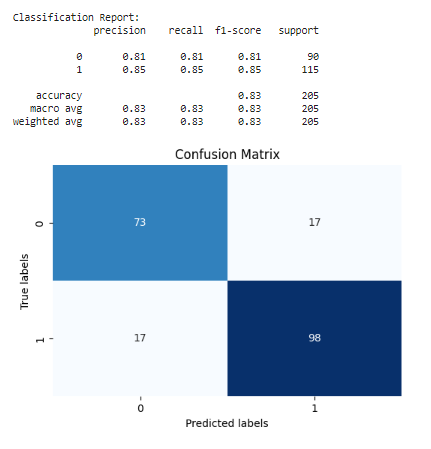
دقت حدود 86 درصد می شود که بهتر از حالت قبل که 84 درصد بود، شد.

حالا مقدار threshold را افزایش می دهیم تا ببینیم که اگر از فیچر هایی که هم بستگی بیشتری دارند استفاده کنیم، خروجی چه تغییری می کند! مقدار هم بستگی را 0.3 در نظر می گیریم درواقع فیچر های بیشتر از 0.3 هم بستگی را نگه می داریم :

این 7 فیچر به عنوان x در نظر گرفته می شود :



حال مدل را در این حالت بررسی می کنیم و داریم :



می بینیم که دقت 83 درصد شده و کاهش یافته است. اگر مقدار threshold را بیشتر از 0.3 در نظر بگیریم، دقت زیر 80 درصد می شود، پس نتیجه می گیریم که همان 8 فیچر بهترین انتخاب برای مدل است. و از همان مدل برای ادامه استفاده می شود.

* **تفاوت حالت های میانگین گیری:**

در کتابخانه scikit-learn، هنگام محاسبه معیارهایی مانند دقت، بازخوانی و امتیاز F1 برای مسائل دسته‌بندی چندکلاسه، دو حالت میانگین‌گیری وجود دارد: ماکرو و میکرو.

در ادامه توضیحی درباره تفاوت این دو حالت آمده است:

* میانگین ماکرو یا Macro Average :

در میانگین ماکرو، معیارها برای هر کلاس به طور مستقل محاسبه شده و سپس میانگین آنها برای تمام کلاس‌ها گرفته می‌شود؛ این روش تمام کلاس‌ها را بدون در نظر گرفتن تراز و تعداد آنها در مجموعه داده، به صورت یکسان در نظر می‌گیرد.

میانگین ماکرو زمانی که می‌خواهیم عملکرد کلی دسته‌بند را برای همه کلاس‌ها بدون در نظر گرفتن ناهمواری‌های کلاسی ارزیابی کنیم، مفید است. این میانگین به عنوان میانگین دقت، بازخوانی یا امتیاز F1 برای هر کلاس بدون در نظر گرفتن ناهمواری کلاسی محاسبه می‌شود.

* میانگین میکرو یا Micro Average :

در میانگین میکرو، مجموع مثبت‌های واقعی، مثبت‌های غلط و منفی‌های غلط برای همه کلاس‌ها محاسبه شده و سپس دقت، بازخوانی و امتیاز F1 بر اساس این تعداد‌های جهانی محاسبه می‌شوند.

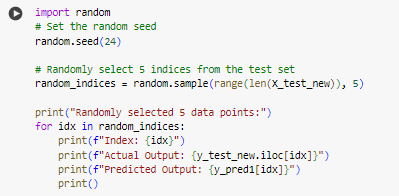
این روش با در نظر گرفتن ناهمواری کلاسی عمل می‌کند زیرا هر نمونه را به طور مساوی وزن می‌دهد( بدون در نظر گرفتن کلاس) . میانگین میکرو زمانی که می‌خواهیم عملکرد کلی دسته‌بند را در نظر بگیریم در حالی که ناهمواری‌های کلاسی را در نظر می‌گیریم، مفید است. این میانگین به عنوان دقت، بازخوانی یا امتیاز F1 برای تمام کلاس‌ها ترکیب شده، با در نظر گرفتن تعداد جهانی محاسبه می‌شود.

پس در کل میانگین ماکرو تمام کلاس‌ها را به یکسان در نظر می‌گیرد و زمانی که هر کلاس اهمیت یکسانی دارد، مناسب است و میانگین میکرو با در نظر گرفتن ناهمواری کلاسی عمل می‌کند و زمانی که مجموعه داده ناهموار است و می‌خواهیم وزن بیشتری به کلاس‌های با پشتیبانی بیشتر بدهیم، مناسب است.

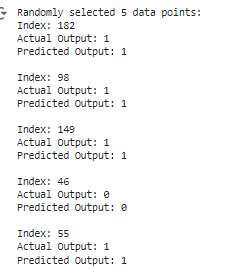
در `classification\_report` scikit-learn، می‌توانیم با استفاده از پارامتر `average` حالت میانگین‌گیری را انتخاب کنیم. تنظیم `average='macro'` معیارها را با استفاده از میانگین ماکرو محاسبه می‌کند، در حالی که `average='micro'` آنها را با استفاده از میانگین میکرو محاسبه می‌کند. اگر `average=None` باشد، معیارها برای هر کلاس به صورت جداگانه بازگردانده می‌شوند.

* **در گام آخر، ۵ داده را به صورت تصادفی از مجموعه آزمون انتخاب میکنیم و خروجی واقعی را با خروجی پیش بینی شده مقایسه میکنیم :**

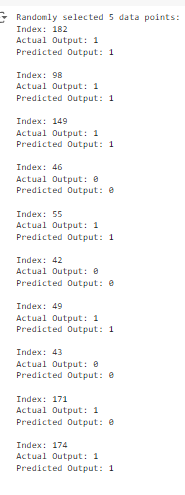
با استفاده از کد زیر 5 داده ی رندوم از داده های تست انتخاب می کنیم :



خروجی به صورت زیر است :



میبینیم که هر 5 داده به خوبی طبقه بندی شده اند در حالیکه دقت مدل 86 درصد است. پس تعداد بیشتری داده رندوم انتخاب می کنیم تا بهتر درک کنیم :



میبینیم که داده رندوم با ایندکس 171 به درستی طبقه بندی نشده است چون در کل دقت مدل 86 درصد است و قرار نیست تمام داده ها درست طبقه بندی شوند.