دانشکده مهندسی برق _گرایش الکترونیک دیجیتال

مینی پروژه شماره دو

یادگیری ماشین

استاد مربوطه

جناب آقای دکتر علیاری

نگارش

سید محمدرضا حسینی

۴۰۲۰۴۵۸۴

لینک گوگل کولب

لینک گیت هاب

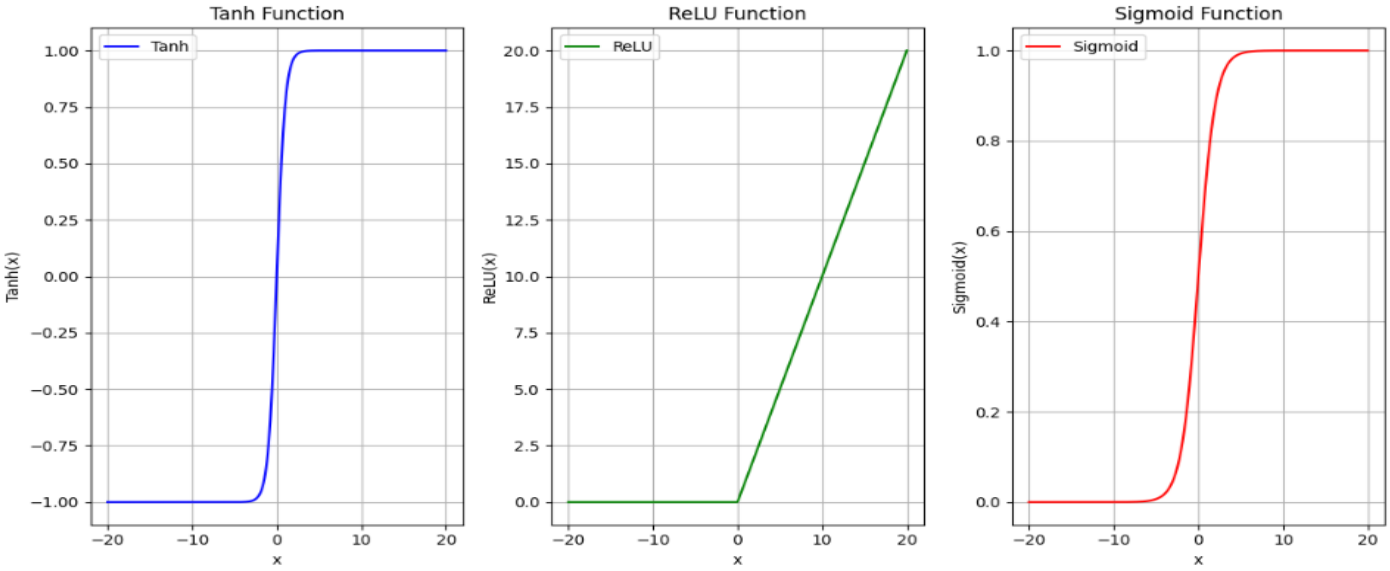
بهار ۱۴۰۳

سوال یک

بخش اول

۱. فرض کنید در یک مسألهٔ طبقه‌بندی دوکلاسه، دو لایهٔ انتهایی شبکهٔ شما فعال‌ساز ReLU و سیگموید است. چه اتفاقی می‌افتد؟

در شکل زیر که با استفاده از کد پایتون بدست آمده است، سه تابع فعال ساز محبوب را میبینیم :



در این سوال پرسیده شده که اگر توابع فعالساز در دو لایه آخر شبکه ما Sigmoid و ReLU باشند، چه اتفاقی می‌افتد؟ در تحقیقی که انجام شد، با مساله‌ای جالب با عنوان آیا استفاده از ReLU بعد از Sigmoid بد است؟ مواجه شدم. در این تحقیق بیان شده که محققان هنگام بررسی تفاوت عملکرد توابع فعالساز مختلف، متوجه شده‌اند که استفاده از ReLU بعد از Sigmoid در دو لایه آخر، عملکرد مدل را کاهش می‌دهد. این مطالعه با استفاده از دیتاست MNIST و یک شبکه ۴ لایه کاملا متصل انجام شده که در همه لایه‌ها به جز لایه اول از ترکیب‌های مختلف توابع غیرخطی استفاده شده و ۶۴ مدل مختلف بررسی شده‌اند. البته، تحقیق انجام شده توابع

فعالساز مورد استفاده را به ۴ تابع ReLU، Sigmoid، Tanh و SeLU محدود کرده است. همچنین، از گرادیان نزولی آماری با نرخ یادگیری ۰.۰۱ استفاده شده است. حال به بررسی نتایج این تحقیق به صورت خلاصه می‌پردازیم:

۱. اگر در لایه اول از ReLU استفاده کنیم و در دو لایه بعدی هر ترکیب دلخواهی بجز Sigmoid و ReLU از ۴ تابع فعالساز ذکر شده استفاده شود، متوسط عملکرد مدل به ۸۵ درصد می‌رسد. به عنوان مثال، برای حالت ReLU, Sigmoid, ReLU به عملکرد ۳۴.۹۱ درصد رسیده است.

۲. اگر در لایه اول از Tanh استفاده شود و در لایه دوم و سوم هر ترکیبی بجز Sigmoid و ReLU استفاده گردد، متوسط عملکرد به ۸۶ درصد می‌رسد. برای حالت Tanh, Sigmoid, ReLU ما عملکرد ضعیف ۵۲.۵۷ درصد را خواهیم داشت.

۳. اگر در لایه اول از Sigmoid استفاده شود و در لایه دوم و سوم همانند دو حالت قبل عمل شود، متوسط عملکرد به ۷۶ درصد می‌رسد. در حالت Sigmoid, Sigmoid, ReLU عملکرد به مقدار باورنکردنی ۱۶.۰۳ درصد رسیده است.

۴. همچنین، اگر در لایه اول از SeLU استفاده کنیم و در دو لایه دیگر مانند حالت‌های قبلی تابع فعالساز را انتخاب کنیم، متوسط عملکرد به ۹۱ درصد می‌رسد که در حالت SeLU, Sigmoid, ReLU به عملکرد ۷۵.۱۶ درصد رسیده‌ایم.

۵. نکته دیگر این است که اگر در دو لایه آخر از Sigmoid و ReLU استفاده شود، پراکندگی دقت بسیار بالاست.

زمانی که ابتدا تابع فعالساز Sigmoid اعمال شود و سپس ReLU، نتایج جالبی به دست می‌آید. ابتدا نگاهی به ویژگی‌های این دو تابع فعالساز می اندازیم:

تابع فعالساز Sigmoid :

تابع Sigmoid به صورت زیر تعریف می‌شود:

$$a_{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

این تابع خروجی را به بازه [۰, ۱] نگاشت می‌کند. به عبارت دیگر، هر ورودی که به تابع Sigmoid داده شود، خروجی آن مقداری بین ۰ و ۱ خواهد بود. این ویژگی به این معناست که همه خروجی‌های این تابع مثبت هستند و هیچ‌گاه منفی نمی‌شوند.

تابع فعالساز ReLU :

تابع ReLU به صورت زیر تعریف می‌شود:

$$a_{ReLU}(z) = \max(0, z)$$

این تابع تمام مقادیر منفی را به صفر تبدیل می‌کند و مقادیر مثبت را بدون تغییر باقی می‌گذارد.

ترکیب Sigmoid و ReLU :

حالا فرض کنید که ابتدا تابع Sigmoid بر روی ورودی اعمال می‌شود و سپس خروجی آن به تابع ReLU داده می‌شود. به طور خاص:

۱. اعمال Sigmoid :

ورودی به تابع Sigmoid داده می‌شود و خروجی مقداری بین ۰ تا ۱ خواهد بود. یعنی:

$$\text{Sigmoid}(x) = y$$

و y همیشه در بازه [۰, ۱] است.

۲. اعمال ReLU :

خروجی Sigmoid به عنوان ورودی به تابع ReLU داده می‌شود. از آنجا که خروجی Sigmoid همیشه مثبت است، تابع ReLU هیچ‌گاه نیاز ندارد که مقادیر منفی را به صفر تبدیل کند. بنابراین:

$$\max(0, y) = \text{ReLU}(y)$$

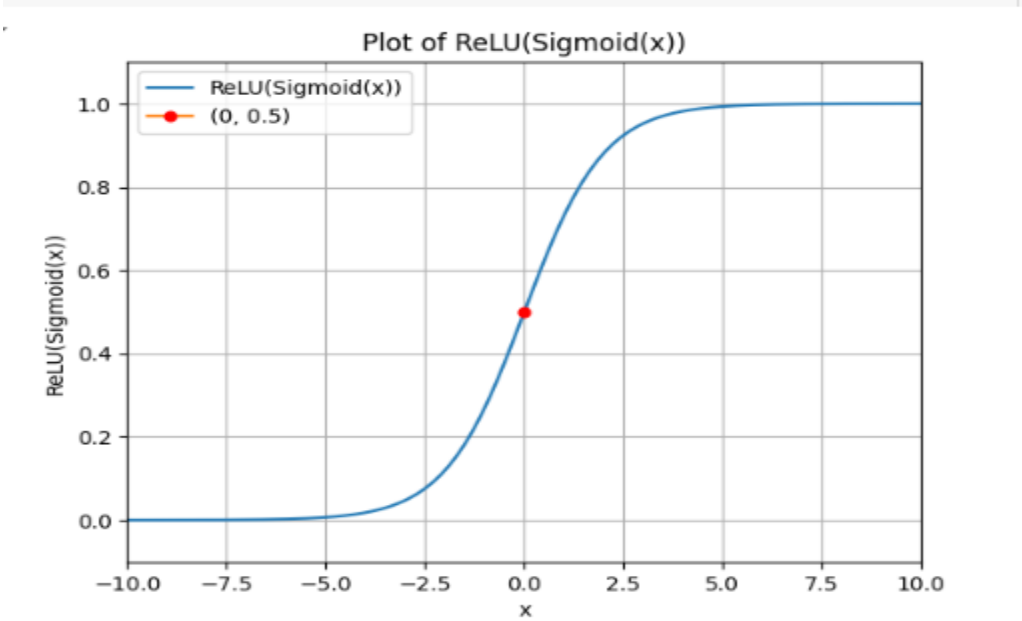
از آنجا که y همیشه مقداری بین ۰ و ۱ است، نتیجه همان y خواهد بود.

به این ترتیب، اعمال ReLU بعد از Sigmoid بی‌فایده است، زیرا تابع ReLU هیچ تغییری در خروجی Sigmoid ایجاد نمی‌کند. به عبارتی، خروجی تابع ReLU همان خروجی تابع Sigmoid خواهد بود. این موضوع به این معناست که استفاده از این دو تابع به صورت پشت سر هم تاثیر مثبتی روی مدل نخواهد داشت و تنها باعث افزایش پیچیدگی محاسباتی بدون بهبود عملکرد می‌شود.

این ترکیب نامناسب از توابع فعالساز می‌تواند به صورت غیرمستقیم باعث کاهش عملکرد مدل شود. زیرا در مراحل آموزش شبکه، مدل تلاش می‌کند تا وزن‌ها و بایاس‌ها را تنظیم کند تا خروجی مناسبی ارائه دهد. استفاده از توابع فعالساز نامناسب می‌تواند فرآیند آموزش را پیچیده‌تر و ناکارآمدتر کند، زیرا مدل باید تلاش بیشتری برای یافتن وزن‌های مناسب انجام دهد و ممکن است در این راه دچار خطا شود.

در نتیجه، انتخاب توابع فعالساز مناسب برای هر لایه از شبکه عصبی اهمیت زیادی دارد و می‌تواند تاثیر زیادی بر عملکرد نهایی مدل داشته باشد.

فرض کنید ابتدا تابع sigmoid اعمال شده است و سپس relu. خروجی لایه اول پس از تابع فعالساز sigmoid مقادیری بین ۰ تا ۱ است. یعنی بجز ابتدای بازه (در منفی بینهایت) متغیر در کل بازه مقدار مثبت دارد. بنابراین اعمال تابع فعالساز relu نمیتواند روی خروجی لایه اول تاثیری بگذارد.

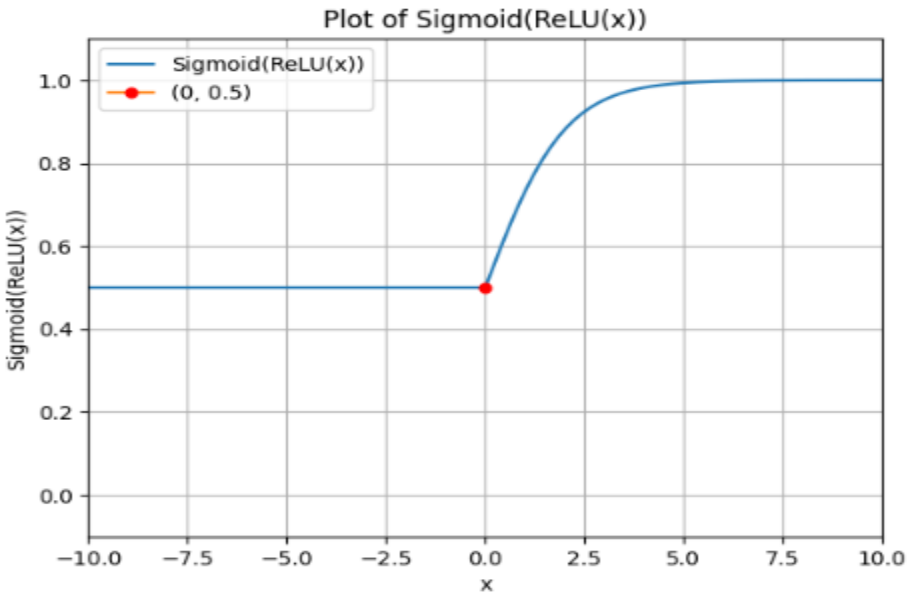


اکنون توابع فعال‌ساز را برعکس کنیم. ابتدا تابع ReLU را اعمال کنیم و سپس تابع Sigmoid خروجی تابع ReLU یا صفر است یا یک مقدار مثبت که برابر با ورودی است (یعنی روی خط $y = x$) وقتی این مقادیر به تابع Sigmoid داده می‌شوند، تابع Sigmoid تلاش می‌کند تا آن‌ها را به بازه‌ی ۰ تا ۱ نگاشت کند. با این حال، مقادیر صفر همگی به یک مقدار کاملاً یکسان و مشخص نگاشت می‌شوند، زیرا تابع Sigmoid در نزدیکی صفر مقدار ۰.۵ را برمی‌گرداند.

این رفتار شبیه به حالتی است که در اشباع اتفاق می‌افتد و به آن "نورون مرده" گفته می‌شود. در این حالت، نورون عملاً غیرفعال می‌شود زیرا خروجی آن همیشه ثابت و بدون تغییر باقی می‌ماند. به این ترتیب، زمانی که ورودی تابع Sigmoid صفر است، خروجی نیز ثابت می‌ماند. از نقطه‌ای که مقادیر مثبت تابع ReLU شروع می‌شوند، تابع Sigmoid می‌تواند عملکرد طبیعی خود را در نگاشت مقادیر انجام دهد.

بنابراین، وقتی ورودی‌های مثبت به تابع Sigmoid داده می‌شوند، این تابع می‌تواند به صورت پیوسته مقادیر ورودی را به بازه‌ی ۰ تا ۱ نگاشت کند. اما برای ورودی‌های صفر، تابع Sigmoid همیشه مقدار ثابتی را برمی‌گرداند. این موضوع باعث می‌شود که نورون‌هایی که ورودی آن‌ها صفر است، هیچ‌گونه اطلاعاتی به لایه‌های بعدی منتقل نکنند و به نوعی "مرده" باشند.

در نتیجه، ترکیب توابع ReLU و Sigmoid به این شکل می‌تواند منجر به مشکلاتی در شبکه عصبی شود. نورون‌هایی که خروجی صفر دارند، اطلاعاتی را به لایه‌های بعدی منتقل نمی‌کنند و به نوعی غیرفعال می‌شوند. این موضوع می‌تواند بر عملکرد کلی شبکه تاثیر منفی بگذارد، زیرا بخش‌هایی از شبکه بدون فعالیت باقی می‌مانند و نمی‌توانند به بهبود و یادگیری مدل کمک کنند. بنابراین، انتخاب توابع فعال‌ساز مناسب و ترتیب صحیح آن‌ها در شبکه عصبی اهمیت زیادی دارد تا عملکرد مدل بهینه باشد و شبکه بتواند به درستی یاد بگیرد و کارایی بالایی داشته باشد.



حال این موضوع را با جزئیات بیشتری بررسی کنیم؛ همانطور که در نمودار فوق مشاهده می شود، عملکرد مدل برای مقادیر کمتر از صفر به طور قابل پیش‌بینی ثابت است. این عملکرد ثابت برای ما پرهزینه و بی‌فایده است، زیرا هیچ اطلاعات خاصی به ما نمی‌دهد. باید توجه داشت که خروجی هر لایه در وزن‌هایی ضرب می‌شود و سپس

با جمع مقادیر ضرب شده در وزن‌ها به لایه بعدی منتقل می‌شود. اما این نکته تاثیری بر مطالب بیان شده ندارد، زیرا ضرب شدن وزن در صفر هیچ تاثیری بر عملکرد ندارد و این اصول همچنان صادق هستند.

اگر بخواهیم دقیق‌تر به این موضوع بپردازیم، می‌توانیم اینگونه بیان کنیم که خروجی تابع ReLU، هر چقدر هم که بزرگ باشد (مقادیر غیر صفر)، با عبور از تابع Sigmoid به سمت ۱ میل می‌کند. این موضوع منجر به ایجاد گرادیان‌های بسیار کوچک می‌شود. این گرادیان‌های کوچک باعث می‌شوند که در مراحل به‌روزرسانی وزن‌ها، تغییرات بسیار کمی رخ دهد که در نتیجه، فرآیند یادگیری شبکه عصبی کند و ناکارآمد می‌شود.

از طرف دیگر، زمانی که مقادیر خروجی از تابع ReLU صفر باشند، تابع Sigmoid این مقادیر را به ۰.۵ میل می‌دهد. این موضوع باعث می‌شود که گرادیان‌ها نه تنها کوچک باشند، بلکه به اندازه کافی موثر نباشند تا تغییرات معنی‌داری در وزن‌ها ایجاد کنند. این حالت باعث می‌شود که بخشی از نورون‌ها به اصطلاح "مرده" باشند و هیچ تاثیری بر فرآیند یادگیری نداشته باشند.

به طور کلی، این وضعیت نه تنها باعث کاهش کارایی مدل می‌شود، بلکه منابع محاسباتی را نیز هدر می‌دهد. زمانی که نورون‌ها هیچ اطلاعات مفیدی را به لایه‌های بعدی منتقل نمی‌کنند، شبکه عصبی نمی‌تواند به درستی آموزش ببیند و بهینه شود. در نتیجه، انتخاب توابع فعال‌ساز مناسب و ترتیب صحیح آن‌ها در شبکه عصبی از اهمیت بالایی برخوردار است تا بتوانیم به عملکرد مطلوبی دست پیدا کنیم.

برای جمع‌بندی، می‌توان گفت که اعمال تابع Sigmoid بعد از ReLU باعث می‌شود که مقادیر بزرگ خروجی ReLU به سمت ۱ میل کنند و منجر به گرادیان‌های کوچک شوند. همچنین، مقادیر صفر خروجی ReLU به ۰.۵ میل می‌کنند که باعث می‌شود این نورون‌ها عملاً غیرفعال شوند و تاثیری در فرآیند یادگیری نداشته باشند. بنابراین، استفاده نادرست از توابع فعال‌ساز می‌تواند عملکرد شبکه عصبی را به شدت تحت تاثیر قرار دهد و منجر به کاهش کارایی و افزایش هزینه محاسباتی شود.

$$sigmoid(x) = \frac{1}{1+e^{-x}} \xrightarrow{x \to 0} sigmoid(x) \to \frac{1}{2}$$

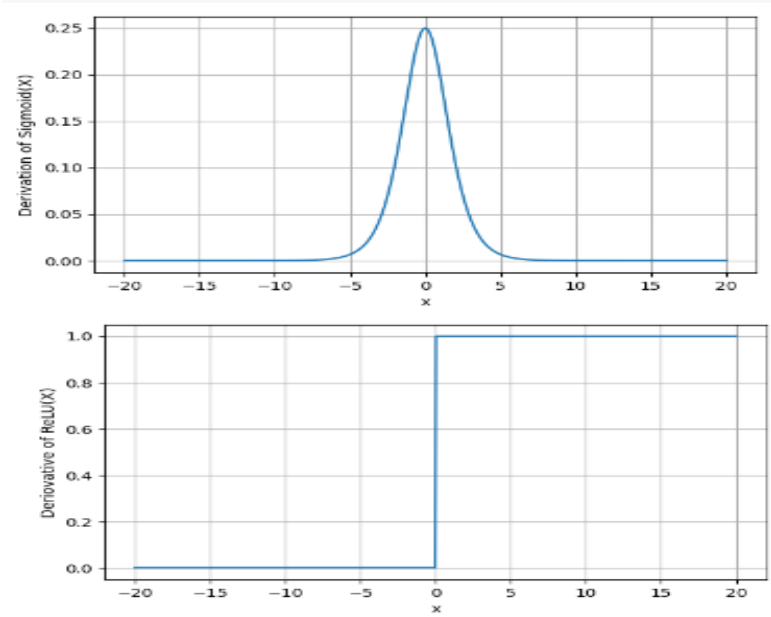
استفاده از توابع فعال‌ساز نادرست می‌تواند به یادگیری ناکارآمد منجر شود. به عنوان مثال، زمانی که خروجی‌های صفر شده توسط تابع فعال‌ساز اول به یک مقدار ثابت مپ می‌شوند، گرادیان بسیار کوچکی ایجاد می‌شود که عملاً یادگیری را متوقف می‌کند. به طور مشابه، در حالت معکوس، مشکلات گرادیان و اثرات اشباع می‌تواند مانع آموزش درست مدل شود و اطلاعات مفید را از دست بدهد. این مشکلات می‌تواند منجر به کند شدن فرآیند همگرایی و گیر کردن در مینیمم‌های محلی شود که توانایی و کارایی شبکه را محدود می‌کند، به ویژه در شبکه‌های عمیق.

هر یک از توابع فعال‌ساز مزایا و معایب خاص خود را دارند. تابع فعال‌ساز ReLU در یادگیری عمیق بسیار محبوب شده است، زیرا مشکل گرادیان کوچک را حل می‌کند و بار محاسباتی را کاهش می‌دهد، همچنین سرعت همگرایی شبکه را افزایش می‌دهد و به مراحل backpropagation بهتر کمک می‌کند. در مقابل، تابع فعال‌ساز Sigmoid، که معمولاً در لایه آخر استفاده می‌شود، به دلیل ایجاد گرادیان‌های کوچک و مشکل اشباع، می‌تواند یادگیری را کند کند اما تفسیرپذیری شبکه را افزایش می‌دهد. استفاده همزمان از این دو تابع فعال‌ساز می‌تواند مشکلات جدی ایجاد کند و عملکرد سیستم را به شدت محدود کند.

پس به طور خلاصه ، هنگامی که داده‌ها از تابع ReLU عبور می‌کنند، مقادیر مثبت حفظ شده و مقادیر منفی به صفر تبدیل می‌شوند. بنابراین، خروجی این لایه شامل اعداد غیرمنفی است. سپس خروجی در وزن‌های لایه نهایی ضرب شده و به تابع فعال‌سازی سیگموید وارد می‌شود. در یک طبقه‌بندی دوکلاسه، لایه نهایی یک نورون دارد و خروجی آن بین ۰ و ۱ قرار می‌گیرد. برای طبقه‌بندی دوکلاسه، استفاده از یک نورون کافی است زیرا عدم تعلق به یک کلاس به معنای تعلق به کلاس دیگر است، که باعث کاهش حجم محاسبات می‌شود. خروجی نورون با یک آستانه مقایسه شده تا کلاس نهایی تعیین شود. تابع سیگموید در فرآیند Backward Propagation مشکلاتی ایجاد می‌کند زیرا شیب آن در نزدیکی صفر کوچک است، که منجر به "مشکل ناپدید شدن گرادیان" و یادگیری کند می‌شود. در مقابل، تابع ReLU برای مقادیر ورودی مثبت، خروجی و شیب برابر با ورودی دارد، که به حفظ گرادیان‌ها در طول Backward Propagation کمک می‌کند و یادگیری را سریع‌تر می‌کند. با این حال، ReLU نیز می‌تواند منجر به "ReLU مرده" شود، یعنی نورون‌هایی که همیشه صفر هستند و غیرفعال می‌مانند.

ترکیب این دو تابع فعال‌سازی می‌تواند مفید باشد: استفاده از ReLU در لایه‌های مخفی برای کاهش مشکل ناپدید شدن گرادیان و استفاده از سیگموید در لایه خروجی برای تفسیر احتمال خروجی بین ۰ و ۱. این ترکیب به بهبود عملکرد مدل، به ویژه در مسائل پیچیده، کمک می‌کند.

نمودار مشتق این دو تابع هم به صورت زیر است :



بخش دوم

بخش سوم

سوال دو

بخش اول

بخش دوم

بخش سوم

بخش چهارم

سوال سه

مجموعه داده مربوط به **طبقه بندی پوشش جنگلی** را در نظر گرفتم.

بخش اول

برای دانلود این داده‌ها از دستور fetch-covtype استفاده کریدم و داریم :

```
[ ]  from sklearn.datasets import fetch_covtype

     #download data
     dataset = fetch_covtype()
     dataset
```

اطلاعات زیر را داریم :

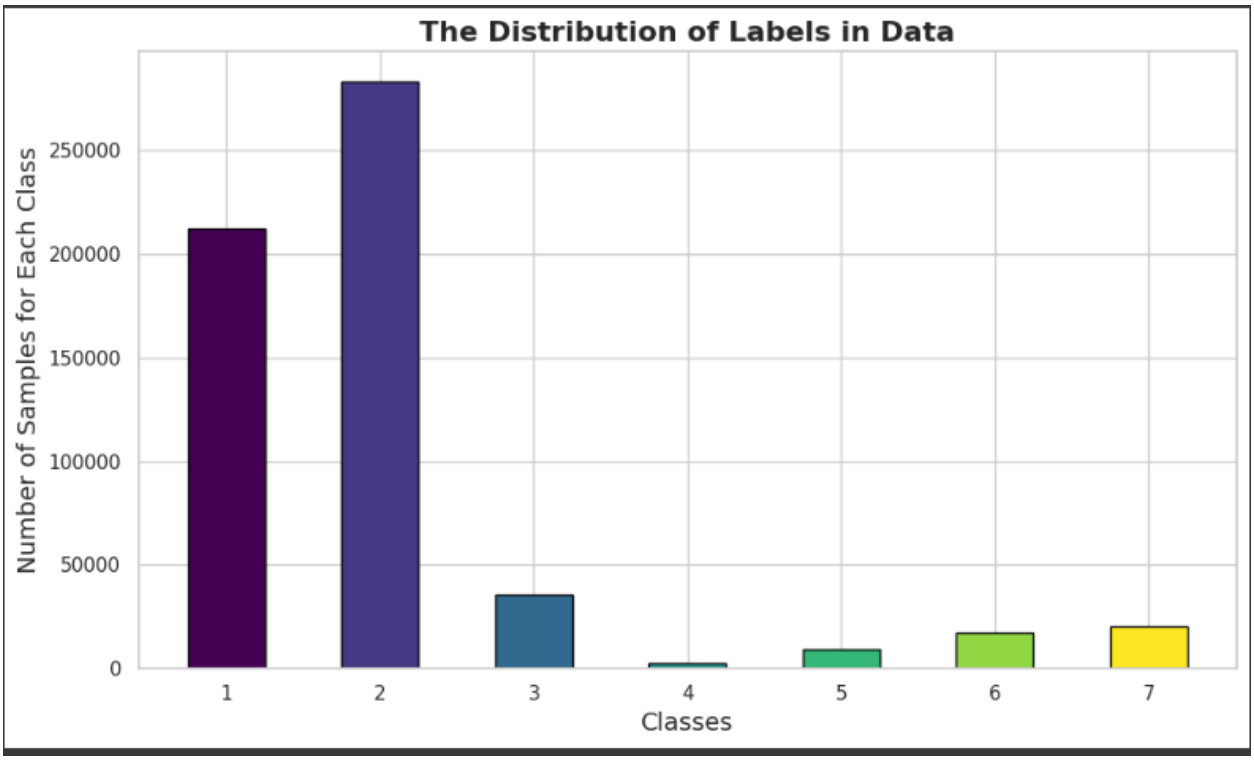
```
  'Soil_Type_39'],
 'DESCR': ".. _covtype_dataset:\n\nForest covertypes\n-----------------\n\nThe samples in this dataset correspond to 30×30m patches of forest in the US,\ncollected for the task of predicting each patch's
cover type,\ni.e. the dominant species of tree.\nThere are seven covertypes, making this a multiclass classification problem.\nEach sample has 54 features, described on the\n`dataset's homepage
<https://archive.ics.uci.edu/ml/datasets/Covertype>`__.\nSome of the features are boolean indicators,\nwhile others are discrete or continuous measurements.\n\n**Data Set Characteristics:**\n\n
===============   ===========\n    Classes                    7\n    Samples total           581012\n    Dimensionality             54\n    Features                  int\n
===============   ===========\n\n:func:`sklearn.datasets.fetch_covtype` will load the covertype dataset;\nit returns a dictionary-like 'Bunch' object\nwith the feature matrix in the ``data``
member\nand the target values in ``target``. If optional argument 'as_frame' is\nset to 'True', it will return ``data`` and ``target`` as pandas\ndata frame, and there will be an additional member
``frame`` as well.\nThe dataset will be downloaded from the web if necessary.\n"}
```

این مجموعه داده شامل نمونه‌هایی از قطعات جنگل به ابعاد ۳۰×۳۰ متر در ایالات متحده است که برای پیش‌بینی نوع پوشش جنگلی هر قطعه جمع‌آوری شده‌اند. این نوع پوشش جنگلی، گونه غالب درختان را نشان می‌دهد. این مجموعه داده دارای هفت نوع پوشش جنگلی مختلف است و بنابراین مسئله دسته‌بندی چندکلاسه (multiclass classification)محسوب می‌شود. هر نمونه دارای ۵۴ ویژگی است که شامل نشانگرهای بولی، مقادیر گسسته و پیوسته می‌شود. مشخصات مجموعه داده شامل ۷ کلاس، ۵۸۱۰۱۲ نمونه و ۵۴ ویژگی است. با استفاده از تابع fetch_covtypeدر کتابخانه scikit-learn می‌توان این مجموعه داده را بارگیری کرد. این تابع یک شیء Bunchبرمی‌گرداند که ماتریس ویژگی‌ها در عضو dataو مقادیر هدف در عضو targetقرار دارند. اگر پارامتر as_frameبه Trueتنظیم شود، داده‌ها و مقادیر هدف به صورت یک DataFrame از pandas بازگردانده می‌شوند و یک عضو اضافی به نام frameنیز وجود خواهد داشت.

```
# Selecting needed elements
X = dataset['data']
y = dataset['target']
features = dataset['feature_names']

# pandas dataframe
df = pd.DataFrame(X, columns=features)
df['Target'] = y

print(f'The shape of the dataframe is: {df.shape}')
```

```
The shape of the dataframe is: (581012, 55)
```

همان طور که گفتیم داده‌ها به ۷ کلاس تقسیم می‌شوند. اکنون باید بررسی کنیم که توزیع داده‌ها در این مجموعه چگونه است، یعنی در هر کلاس چند نمونه وجود دارد.



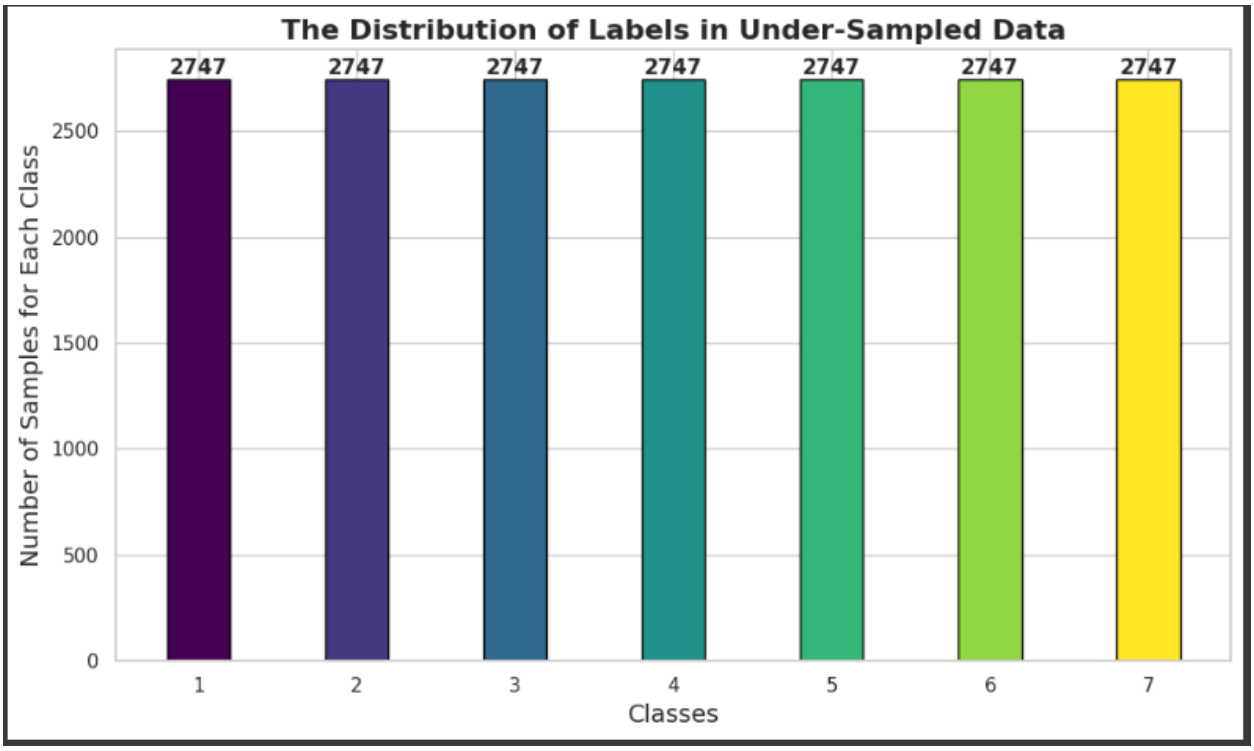
مشاهده می شود که داده‌ها به طور متعادل توزیع نشده‌اند و تعداد نمونه‌ها در هر کلاس بسیار متفاوت است؛ به عنوان مثال، یک کلاس کمتر از ۵۰،۰۰۰ نمونه دارد در حالی که کلاس دیگر بیش از ۲۵۰،۰۰۰ نمونه دارد. این عدم تعادل می‌تواند باعث شود مدل به سمت کلاسی که بیشترین تعداد داده را دارد متمایل شود و عملکرد مطلوبی نداشته باشد. برای رفع این مشکل، از روش Under-sampling استفاده می‌کنیم که شامل کاهش تعداد نمونه‌ها در کلاس‌هایی است که بیشترین داده را دارند تا تعداد نمونه‌ها در تمام کلاس‌ها برابر شود. این کار به مدل کمک می‌کند تا به‌طور یکسان 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 مهم است. در برخی موارد، ترکیب این روش با Over-sampling یا استفاده از الگوریتم‌های مقاوم به عدم تعادل داده نیز می‌تواند موثر باشد. برای حل این مشکل از کد زیر استفاده می‌کنیم:

```
[7]  sampler = RandomUnderSampler(random_state=84)
     x_resampled, y_resampled = sampler.fit_resample(X, y)
```

```
[8]  hist, bins = np.histogram(y_resampled, bins=7)
     bins = np.unique(y_resampled)

     # Set the style
     sns.set(style="whitegrid")

     # Create the color map
     cmap = cm.get_cmap('viridis')

     # Plot the bar chart
     plt.figure(figsize=(10, 6))
     bars = plt.bar(bins, hist, width=0.4, edgecolor='black', color=cmap(np.linspace(0, 1, len(hist))))

     # Add labels and title
     plt.xticks(range(1, 8), range(1, 8))
     plt.title('The Distribution of Labels in Under-Sampled Data', fontsize=16, fontweight='bold')
     plt.ylabel('Number of Samples for Each Class', fontsize=14)
     plt.xlabel('Classes', fontsize=14)

     # Add value labels on the bars
     for bar in bars:
         height = bar.get_height()
         plt.text(bar.get_x() + bar.get_width() / 2.0, height, f'{height}', ha='center', va='bottom', fontsize=12, fontweight='bold')

     # Enhance the plot aesthetics
     plt.tight_layout()
     plt.show()
```

خروجی در این حالت که بالانس کردیم به صورت زیر است :



پس از به‌کارگیری روش Under-sampling، تعداد نمونه‌ها در هر کلاس برابر می‌شود و دیتاست متعادل می‌گردد. با این حال، این روش باعث کاهش کل حجم داده‌ها می‌شود، زیرا برای ایجاد تعادل، برخی از نمونه‌های کلاس‌های دارای تعداد بالا حذف می‌شوند. در حال حاضر، ما دارای ۷ کلاس هستیم که هر یک ۲۷۴۷ نمونه دارند.

برای تقسیم داده‌ها، از روش نمونه‌برداری تصادفی (random sampling)با تابع train_test_split استفاده می‌کنیم که در آن **۱۵ درصد داده‌ها به مجموعه آزمون و بقیه به مجموعه آموزش** اختصاص می‌یابد.

برای دو حالت (حالت اولیه و حالت بالانس شده) در نظر می گیریم و داریم :

```
# Split resmapled data
x_train_r, x_test_r, y_train_r, y_test_r = train_test_split(
    x_resampled,
    y_resampled,
    random_state = 84,
    test_size = 0.15,
    shuffle = True
)
```

```
# Split main data
x_train, x_test, y_train, y_test = train_test_split(
    X,
    y,
    random_state = 84,
    test_size = 0.15,
    shuffle = True
)
```

همچنین شافل و نرمالایز نیز می کنیم.

دو روش دیگر نیز برای انتخاب داده‌ها وجود دارد که ما لحاظ کردیم تا دقیق تر بررسی کنیم:

۱. اعتبارسنجی متقابل (Cross Validation) : دیتاست به بخش‌های متعدد تقسیم شده و مدل‌ها به تعداد بخش‌ها آموزش داده می‌شوند. میانگین خطای مدل‌ها برای ارزیابی دقیق‌تر استفاده می‌شود.

۲. بوت‌استرپ (Bootstrap) : زیرمجموعه‌هایی از دیتاست اصلی با قابلیت جایگذاری انتخاب می‌شود که به ارزیابی تنوع مدل‌ها کمک می‌کند.

استفاده از این روش‌ها بسته به نوع پروژه و تعداد داده‌ها متفاوت است. اگر تعداد داده‌ها پس از روش Under-sampling به حد مناسبی (مثلاً ۲۷۴۷ برای هر کلاس) برسد، نیازی به این روش‌ها نیست.

در فرآیند آموزش مدل درخت تصمیم‌گیری، پنج روش مختلف برای آموزش مدل مورد بررسی قرار گرفت.

```
dt = DecisionTreeClassifier(random_state=84 )
dt1 = clone(dt)
dt3 = clone(dt)
dt2 = clone(dt)
```

این کد به منظور ایجاد چهار مدل درخت تصمیم‌گیری طراحی شده است. از تابع clone()برای ساخت نسخه‌های مستقل از یک مدل موجود استفاده می‌شود تا هر مدل به طور جداگانه عمل کند و تغییرات یک مدل بر سایر مدل‌ها تأثیری نگذارد. در اینجا، dt1، dt2 و dt3 به ترتیب سه نسخه از مدل اصلی DecisionTreeClassifier با تنظیمات و تصمیمات یکسان تولید می‌شوند. بهترین مدل‌ها از میان این روش‌ها برای مراحل بعدی انتخاب شدند. روش‌های انتخابی شامل موارد زیر بودند:

این روش ها به صورت زیر هستند :

۱) آموزش داده‌هایی که با روش under-sampling متعادل شده‌اند؛ به این صورت که تعداد نمونه‌های هر کلاس را کاهش داده‌ایم تا تعادل میان کلاس‌ها حفظ شود.
۲) آموزش داده‌ها بدون هیچ گونه تغییر یا عملیات اضافی؛ به عبارت دیگر، استفاده از داده‌های اصلی بدون تغییر در توزیع یا تعداد آن‌ها.
۳) آموزش داده‌ها با استفاده از وزن‌دهی به الگوریتم؛ بدین معنا که برای هر کلاس وزنی تعیین شده و در فرآیند آموزش به کار رفته است تا تأثیر هر کلاس بر مدل به طور متوازن حفظ شود.
۴) آموزش داده‌ها با روش ۵-fold Cross-validation؛ به این صورت که داده‌ها به ۵ بخش تقسیم می‌شوند و مدل به‌طور متوالی بر روی ۴ بخش آموزش داده شده و بر روی بخش باقی‌مانده آزمایش می‌شود. این فرآیند به طور کامل ۵ بار تکرار می‌گردد.
۵) آموزش داده‌ها با استفاده از روش ۵Stratified fold Cross-validation؛ که علاوه بر اجرای ۵-fold Cross-validation، اطمینان می‌دهد که توزیع کلاس‌ها در هر بخش حفظ شود.

کد این ۵ حالت به صورت زیر است : (هر حالت با # مشخص شده است)

```
# With under-sampled data
dt.fit(x_train_r, y_train_r)
print(f'Model score for under-sampled data is: {dt.score(x_test_r, y_test_r):0.4f}', end='\n\n')

# With normal data
dt1.fit(x_train, y_train)
print(f'Model score for normal data is: {dt1.score(x_test, y_test):0.4f}', end='\n\n')

# With Weighted classes
weight = [
    len(y)/len(y[y==1]),
    len(y)/len(y[y==2]),
    len(y)/len(y[y==3]),
    len(y)/len(y[y==4]),
    len(y)/len(y[y==5]),
    len(y)/len(y[y==6]),
    len(y)/len(y[y==7])
]
w_train = [weight[a-1] for a in y_train]
w_test = [weight[a-1] for a in y_test]
dt2.fit(x_train, y_train, w_train)
print(f"The weight for each class is : {np.round(weight).astype('int16')}")
print(f'Model score for weighted data is: {dt2.score(x_test,y_test,w_test):0.4f}', end='\n\n')

# With K-fold method
kfold_score = cross_val_score(dt, X, y, cv=5)
print(f'Model score for 5-fold CV is : {np.mean(kfold_score):.4f} --> (mean value)', end='\n\n')

# Stratified K-fold model
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=84 )
fold_acc = {'train':[], 'val':[]}
fold_hat = {'train':[], 'val':[]}
fold_ground = {'train':[], 'val':[]}
fold_input = {'train':[], 'val':[]}
all_models = []
```

مدل را برای همه این حالت ها اموزش می دهیم و دقت را بررسی می کنیم. از کد زیر استفاده می کنیم:

```python
# Train model
for train_index, val_index in kf.split(x_train,y_train):

    X_train_fold, X_val_fold = x_train[train_index], x_train[val_index]
    y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]

    fold_input['train'].append(X_train_fold)
    fold_input['val'].append(X_val_fold)
    fold_ground['train'].append(y_train_fold)
    fold_ground['val'].append(y_val_fold)

    # Train selected folds
    model = clone(dt3)
    model.fit(X_train_fold,y_train_fold)

    # Accuracy
    train_hat = model.predict(X_train_fold)
    fold_hat['train'].append(train_hat)
    train_hat = train_hat == y_train_fold
    train_score = np.sum(train_hat.astype('float64'))/len(y_train_fold)
    fold_acc['train'].append(train_score)

    val_hat = model.predict(X_val_fold)
    fold_hat['val'].append(val_hat)
    val_hat = val_hat == y_val_fold
    val_score = np.sum(val_hat.astype('float64'))/len(y_val_fold)
    fold_acc['val'].append(val_score)

    # Saving all models
    all_models.append(model)

skf_score = np.mean(fold_acc['val'])
print(f'Model score for Stratified 5-fold CV is {skf_score:0.4f} --> (mean value)')
```

دقت برای این ۵ حالت به صورت زیر است :

```
Model score for under-sampled data is: 0.8014

Model score for normal data is: 0.9393

The weight for each class is : [  3   2  16 212  61  33  28]
Model score for weighted data is: 0.8920

Model score for 5-fold CV is : 0.5561 --> (mean value)

Model score for Stratified 5-fold CV is 0.9338 --> (mean value)
```

این مقایسه‌ها به منظور انتخاب بهترین روش آموزش برای مدل درخت تصمیم‌گیری انجام شد. هرچند عملکرد مدل‌ها برای کلاس‌های مختلف متفاوت است و هیچ کدام به طور کامل بهتر از سایرین عمل نمی‌کنند، اما به طور کلی، می‌توانیم حالت دوم را انتخاب کنیم که دقت حدود ۹۳ ٪ دارد و از دیگر حالت‌ها بهتر است. به عبارت دیگر، استفاده از داده‌های اصلی بدون وزن‌دهی، نتیجه بهتری برای کلیت مدل به ارمغان می‌آورد. بنابراین، برای ادامه محاسبات، تصمیم گرفته‌ایم از تمامی داده‌ها بدون در نظر گرفتن وزن استفاده کنیم. اگرچه حالت پنجم نیز عملکردی نزدیک به این مدل دارد، اما ما از حالت دوم استفاده خواهیم کرد.

اکنون اطلاعات اساسی درخت را استخراج می‌کنیم: ابتدا با استفاده از متغیر dt1.tree_، اطلاعات مربوط به درخت را به دست می‌آوریم. سپس سه ویژگی اصلی را استخراج می‌کنیم: ۱) عمق بیشترین مسیر درخت (Max depth of the tree)، ۲) تعداد نمونه‌های موجود در برگ‌های درخت (Number of samples at leaves)و ۳) میزان ناخالصی در برگ‌های درخت (Impurity at leaves).این اطلاعات معمولاً برای ارزیابی و درک بهتر عملکرد مدل درخت تصمیم به کار می‌روند، که به صورت زیر ارائه می‌شوند:

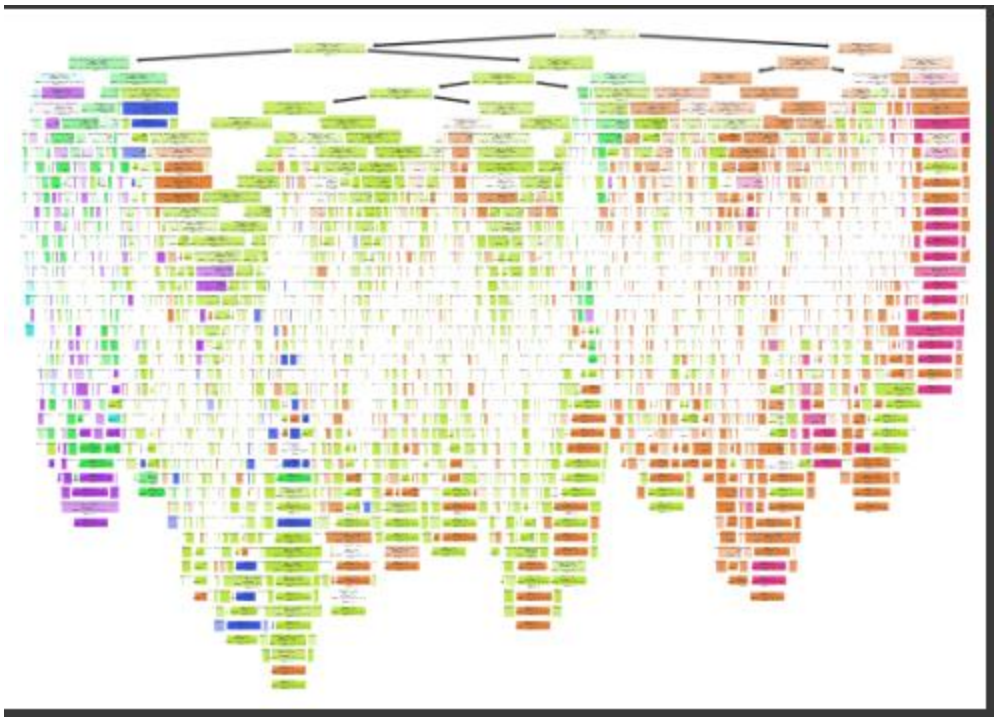
```
tree_ob = dt1.tree_

# max depth
md = tree_ob.max_depth
print(f'Max depth of the tree is {md}', end='\n\n')

# number of samples in the leaves
ns = tree_ob.n_node_samples[-1]
print(f'There are {ns} number of samples at leaves', end='\n\n')

# Impurity at leaves
i_in_leaf = tree_ob.impurity[-1]
print(f'Impurity at leaves is {i_in_leaf}')
```

```
Max depth of the tree is 44

There are 2 number of samples at leaves

Impurity at leaves is 0.0
```

عمق بیشترین مسیر در درخت ۴۴ است که نشان‌دهنده پیچیدگی و ظرفیت یادگیری مدل می‌باشد؛ عمق زیاد می‌تواند به مدل این امکان را بدهد که روابط غیرخطی پیچیده‌تری را یاد بگیرد. با این حال، وجود تنها ۲ نمونه در برگ‌های درخت نشان‌دهنده این است که ممکن است مدل به شدت به داده‌های آموزشی وابسته شده و دچار Overfitting شود، زیرا تعداد کم نمونه‌ها می‌تواند منجر به نتایج غیرقابل تعمیم شود. از سوی دیگر، میزان ناخالصی در برگ‌ها برابر با ۰.۰ است، که نشان‌دهنده این است که همه نمونه‌ها در این برگ‌ها متعلق به یک کلاس هستند و مدل توانسته است به تفکیک کاملاً دقیقی دست یابد. به طور کلی، این ویژگی‌ها به ما می‌گوید که اگرچه مدل قادر به شناسایی دقیق نمونه‌ها بوده، اما با توجه به عمق و تعداد کم نمونه‌ها، نیاز به بررسی بیشتر برای ارزیابی عمومی‌سازی و کارایی آن وجود دارد.

شکل کلی درخت به صورت زیر است :

```
plot_tree(dt1, filled=True, feature_names=dataset.feature_names, class_names=[str(i) for i in np.unique(y)])
```



هر گام در این درخت به بررسی یک ویژگی خاص می‌پردازد و مقادیر Gini (معیار نابرابری) نشان‌دهنده خلوص یا اختلاط کلاس‌ها در آن نقطه است. به‌طور کلی، با کاهش مقدارGini ، کیفیت تفکیک کلاس‌ها در داده‌ها بهبود می‌یابد. تعداد نمونه‌ها و مقادیر مربوط به کلاس‌ها در هر گام نشان‌دهنده توزیع داده‌ها و توانایی مدل در تشخیص کلاس‌های مختلف است. این مدل می‌تواند برای پیش‌بینی یا طبقه‌بندی داده‌ها بر اساس ویژگی‌های محیطی مورد استفاده قرار گیرد و به کمک آن می‌توان الگوهای جغرافیایی و اکولوژیکی را شناسایی کرد.

همینطور بخشی از خروجی به صورت زیر است :

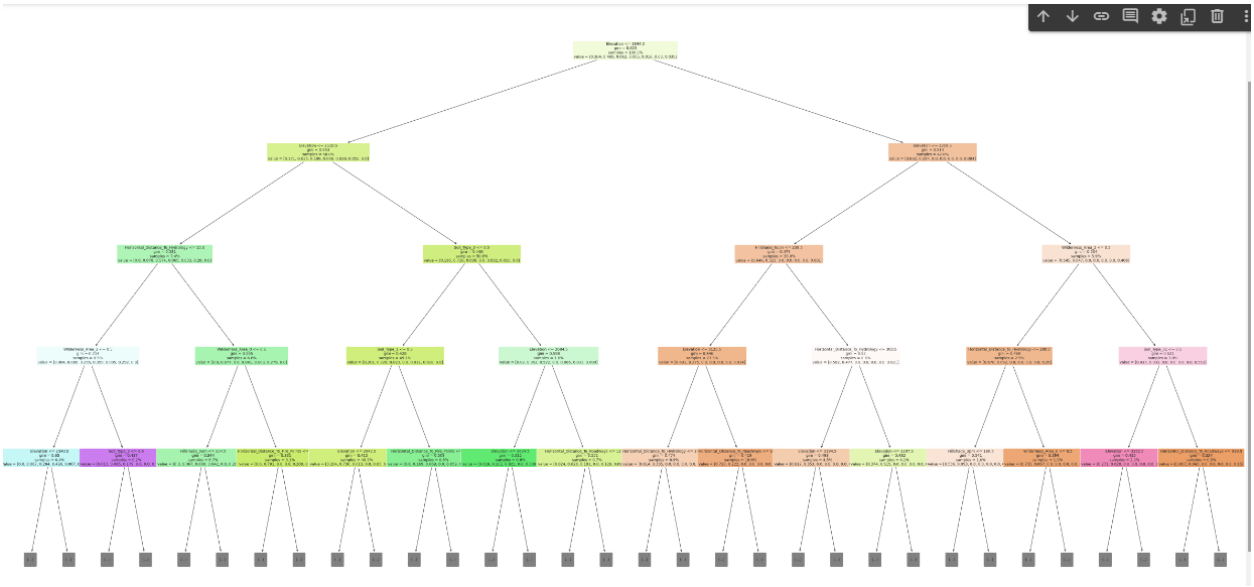
```
[Text(0.6143890309959978, 0.9888888888888889, 'Elevation <= 3044.5\ngini = 0.623\nsamples = 493860\nvalue = [179844, 240831, 30509, 2354, 8033, 14784, 17505]\nclass = 2'),
 Text(0.31453825294231913, 0.9666666666666667, 'Elevation <= 2510.5\ngini = 0.553\nsamples = 286561\nvalue = [48893, 181917, 30509, 2354, 8033, 14784, 71]\nclass = 2'),
 Text(0.05581919136958949, 0.9444444444444444, 'Horizontal_Distance_To_Hydrology <= 15.0\ngini = 0.581\nsamples = 36451\nvalue = [11, 2837, 20935, 2352, 108, 10208, 0]\nclass =
3'),
```

گام اول، بررسی ارتفاع بالاتر از ۳۰۴۴.۵ متر است که با Gini برابر ۰.۶۲۳ نشان‌دهنده نابرابری نسبتا بالای کلاس‌ها در این نقطه است. تعداد نمونه‌ها (۴۹۳۸۶۰) و توزیع کلاس‌ها نیز اطلاعات مهمی درباره تنوع داده‌ها ارائه می‌دهد. در ادامه، با کاهش ارتفاع به ۲۵۱۰.۵ متر، Giniبه ۰.۵۵۳ کاهش می‌یابد که نشان‌دهنده بهبود در خلوص کلاس‌هاست. همچنین، در مرحله سوم، با بررسی فاصله افقی به منابع آبی، Giniبه ۰.۵۸۱ تغییر می‌کند، که به تغییرات قابل توجه در ساختار داده‌ها اشاره دارد. این درخت می‌تواند به درک الگوهای جغرافیایی و زیست‌محیطی کمک کند و ابزار مفیدی برای پیش‌بینی رفتارهای مربوط به اکوسیستم‌ها باشد.

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

```
plt.figure(figsize=(60,30))
plot_tree(dt1, max_depth=4, filled=True , fontsize=10,feature_names=features, rounded=True, proportion=True)
```

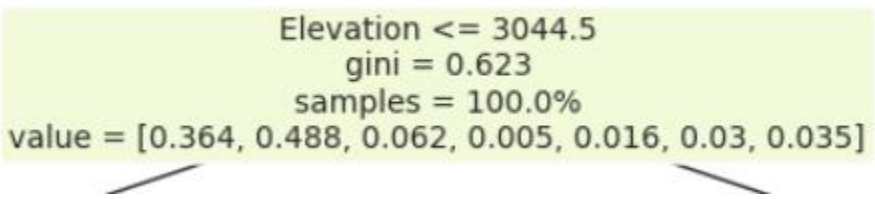
درخت به صورت زیر می شود :



برای دقت بیشتر، لایه اول را زوم می کنیم :

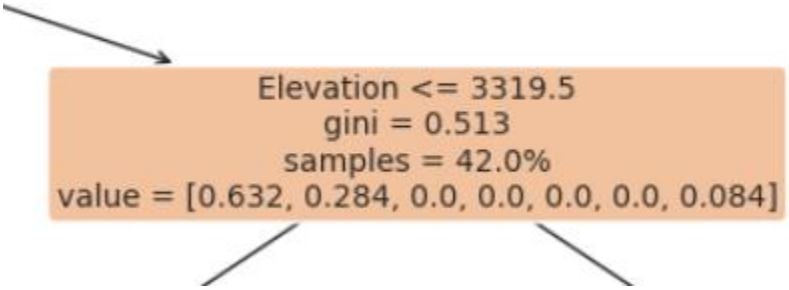
تقسیم‌بندی لایه دوم درخت به این صورت است:

در هر بلوک، اطلاعاتی از قبیل نام ویژگی، مقدار آستانه (threshold)، مقدار Gini و تعداد نمونه‌های موجود در آن گره نمایش داده شده است. به عنوان مثال، در اولین بلوک:

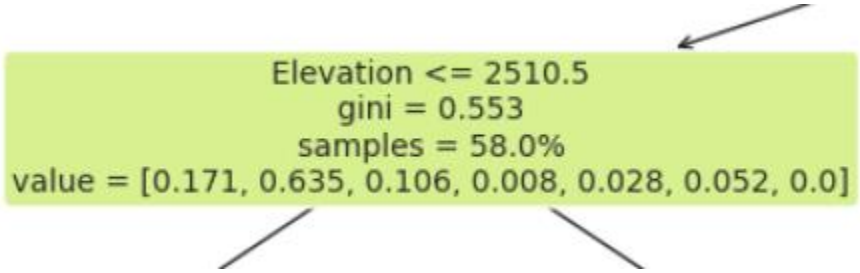


در تصویر بالا، مقدار valueیک آرایه متشکل از ۷ عنصر عددی است که نشان‌دهنده فراوانی یا احتمال هر کلاس/برچسب در این گره از درخت است. مجموع این عناصر برابر با ۱ است. به عنوان مثال، اگر برچسب‌های داده‌ها شامل [۰، ۱] باشند، دو عنصر اول می‌توانند نشان‌دهنده فراوانی کلاس‌های ۰ و ۱ در این گره باشند. مقدار elevation=3044.5به عنوان یک معیار تصمیم‌گیری برای تفکیک این گره از گره قبلی بر اساس یک ویژگی خاص عمل می‌کند. به عنوان مثال، اگر ویژگی پیشگو، درآمد باشد، درآمد بالاتر از ۳۰۴۴.۵ دلار به سمت راست درخت و درآمد کمتر از آن به سمت چپ درخت می‌رود. مقدار gini=0.623نشان‌دهنده ناهمگنی داده‌ها در این گره است و بین ۰ (کاملاً همگن) و ۰.۵ (کاملاً ناهمگن) متغیر است. مقدار بالای Gini نشان‌دهنده تنوع بالای داده‌ها در این گره است. همچنین، samples=100.0%به این معناست که تمامی نمونه‌های داده در این گره از درخت قرار دارند.

بلوک بعدی سمت راست : ۴۲ درصد داده ها متعلق به این هستند.



و بلوک سمت چپ : ۵۸ درصد داده ها متعلق به این هستند.

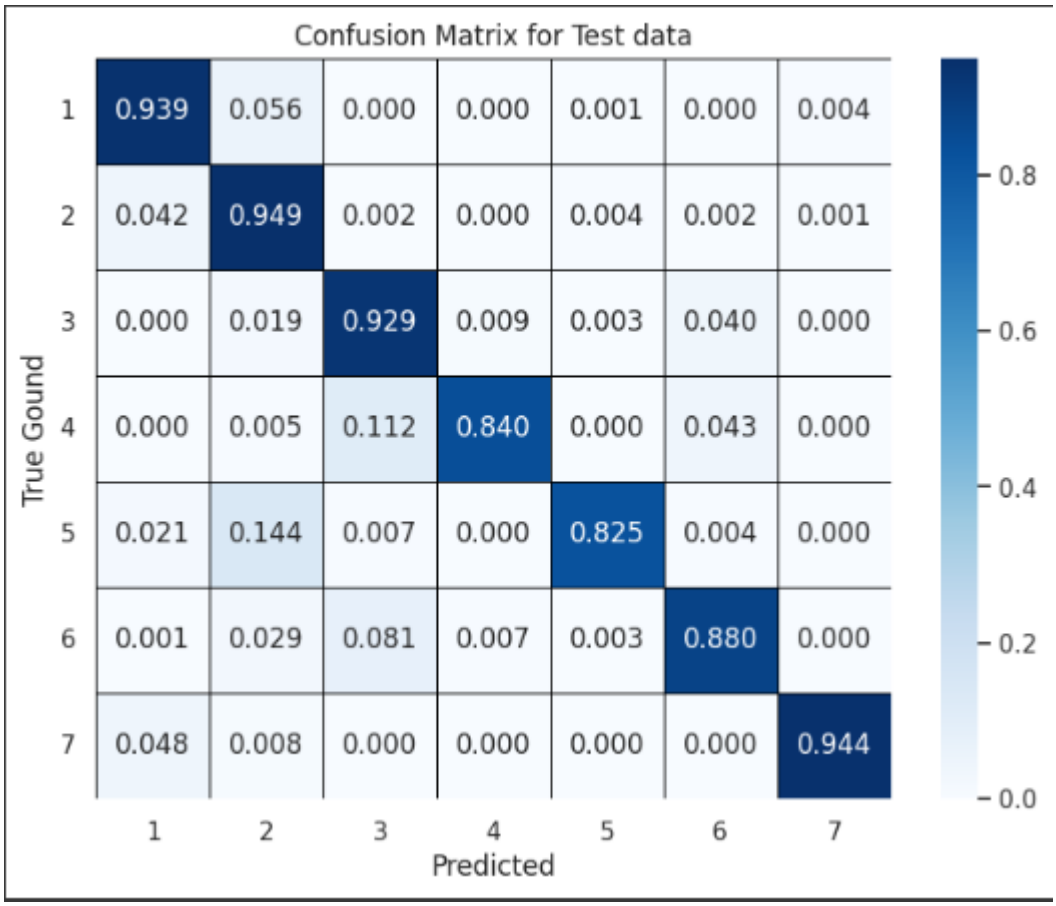


باز این روال ادامه دارد تا انتها (مثلا از ۵۸ درصد این بلوک، دوباره به نسبت هایی در بلوک های پایینی تقسیم شدند.)

این اطلاعات به درک نحوه ساخت، اعتبارسنجی و ارزیابی عملکرد یک مدل درخت تصمیم در مسائل دسته‌بندی یا پیش‌بینی کمک می‌کند. این درخت به عمق ۴۳ سطح تفکیک رسیده و در هر برگ آن تنها ۲ نمونه وجود دارد. از این داده‌ها می‌توان نکات مهمی استخراج کرد: اولاً، به دلیل عمق زیاد این درخت، نمایش تمامی گره‌ها و برگ‌های آن به صورت تصویر یا متن امکان‌پذیر نیست. دوماً، این مدل به شدت دچار overfitting شده است؛ به این معنا که بیش از حد به داده‌های آموزشی خود برازش یافته و احتمالاً در مواجهه با داده‌های جدید عملکرد مناسبی نخواهد داشت.

بخش دوم

عمل کرد درخت آموزش داده را روی بخش تست داده ها، توسط ماتریس درهم ریختگی و سه خاصه ارزیابی به صورت زیر نشان می دهیم :



Classification Metrics

	class	precision	recall	f1_score
0	1	0.9384	0.9391	0.9388
1	2	0.9483	0.9488	0.9486
2	3	0.9303	0.9289	0.9296
3	4	0.8312	0.8397	0.8354
4	5	0.8467	0.8247	0.8355
5	6	0.8790	0.8800	0.8795
6	7	0.9460	0.9441	0.9450

بخش سوم

سوال چهار

ابتدا دیتاست بیماری قلبی را از Kaggle دانلود می‌کنیم. این مجموعه داده که از سال ۱۹۸۸ جمع‌آوری شده، شامل اطلاعات از چهار پایگاه داده مختلف (کلیولند، مجارستان، سوئیس و لانگ بیچ) است و دارای ۷۶ ویژگی است. اما معمولاً از ۱۴ ویژگی کلیدی برای تحلیل‌ها استفاده می‌شود. هدف این مجموعه داده، تشخیص بیماری قلبی در بیماران است. فیلد " "targetوجود یا عدم وجود بیماری قلبی را نشان می‌دهد (۰ = بدون بیماری، ۱ = بیماری). ویژگی‌های کلیدی شامل سن، جنسیت، نوع درد سینه، فشار خون استراحتی، کلسترول سرمی، قند خون ناشتا، نتایج الکتروکاردیوگرافی استراحتی، بیشینه ضربان قلب، آنژین تحریکی، زیرفشار، شیب قله ST در تمرین، تعداد عروق اصلی، تال (۰ = نرمال، ۱ = نقص ثابت، ۲ = نقص قابل برگشت) و بیماری قلبی است.

دیتاست به صورت زیر است :

```
df = pd.read_csv('/content/drive/MyDrive/MachineLearning/HW2/heart.csv')
df
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

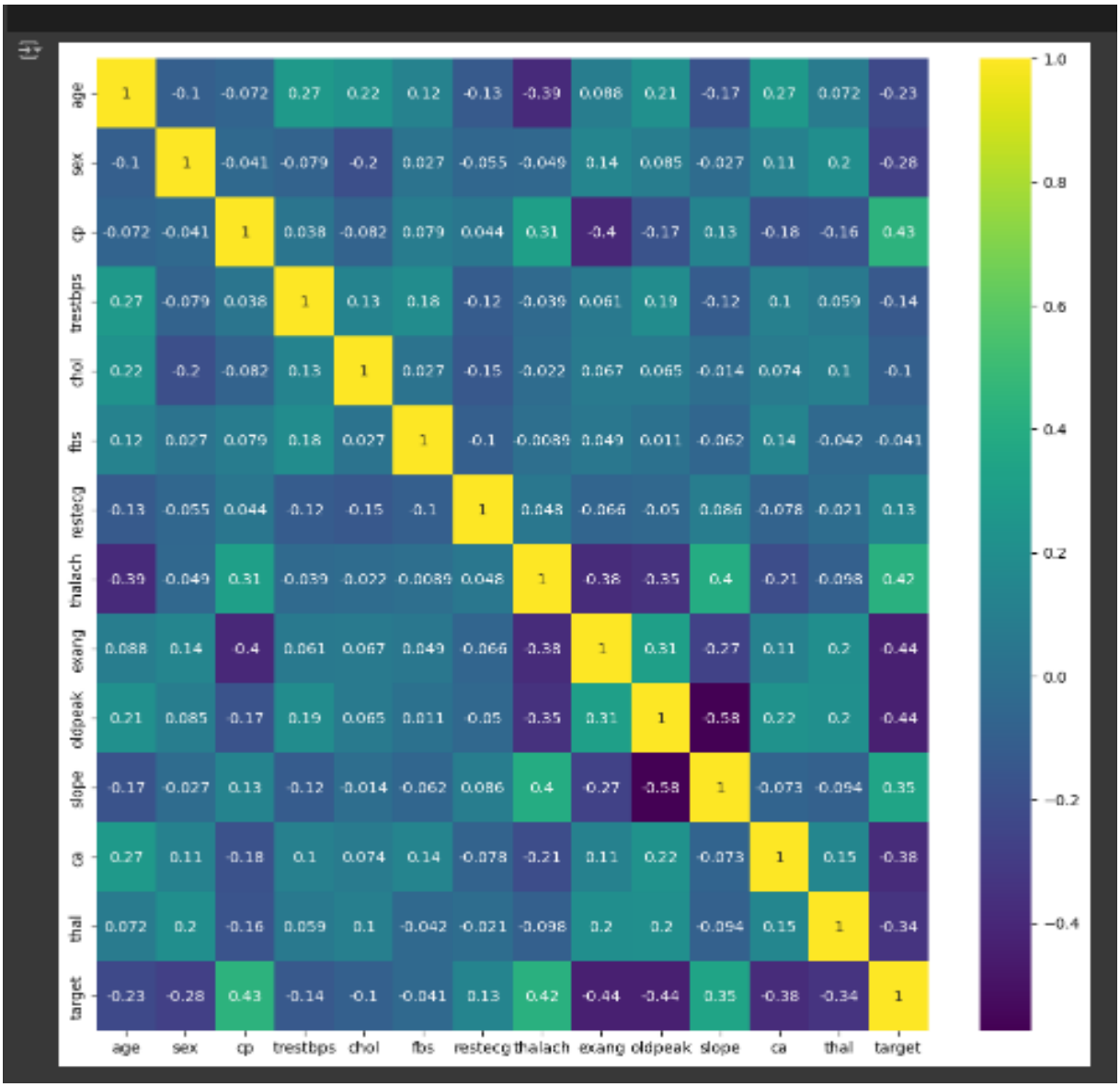
این مجموعه داده شامل ۱۰۲۵ نمونه و ۱۳ ویژگی است و ستون targetنقش لیبل را دارد. بنابراین، لیبل‌ها به عنوان y و ۱۳ ویژگی به عنوان x استفاده می‌شوند. در مرحله بعد، تعداد نمونه‌ها در هر کلاس را بررسی می‌کنیم تا وضعیت توزیع داده‌ها در هر کلاس را مشاهده کنیم: (میبینیم که خوب تقسیم شدند و تقریبا بالانس هستند.)

```
[6]  X = df.drop(columns=['target'])
     y = df['target']

[7]  number_of_ones = y.sum()
     print(f'Number of class one data: {number_of_ones}')
     print(f'Number of class zeros data: {len(y) - number_of_ones}')

     Number of class one data: 526
     Number of class zeros data: 499
```

اکنون همبستگی بین این ویژگی‌ها را بررسی می‌کنیم تا ارتباطات بین آن‌ها را شناسایی کنیم. در واقع، نیازی نیست که از تمام ۱۳ ویژگی در مدل خود استفاده کنیم و می‌توانیم از مناسب‌ترین ویژگی‌ها بهره ببریم.

پس من یک بار بررسی کردم که اگر از همه فیچر ها استفاده کنم چجوری میشه و یک بار هم یک تابع نوشتم که مقدار threshold را بهش می دهیم و با توجه به آن فیچر های مناسب را در نظر می گیرد که در ادامه بیشتر توضیح داده می شود.

```
plt.rcParams["figure.figsize"] = (12, 12)
cor = df.corr()
fig, ax = plt.subplots()
ax = sns.heatmap(cor, annot=True, cmap="viridis")
```



از الگوریتم Bayes استفاده می کنیم و کد آن به صورت زیر است : (در این حالت از همه فیچر ها استفاده می کنیم)

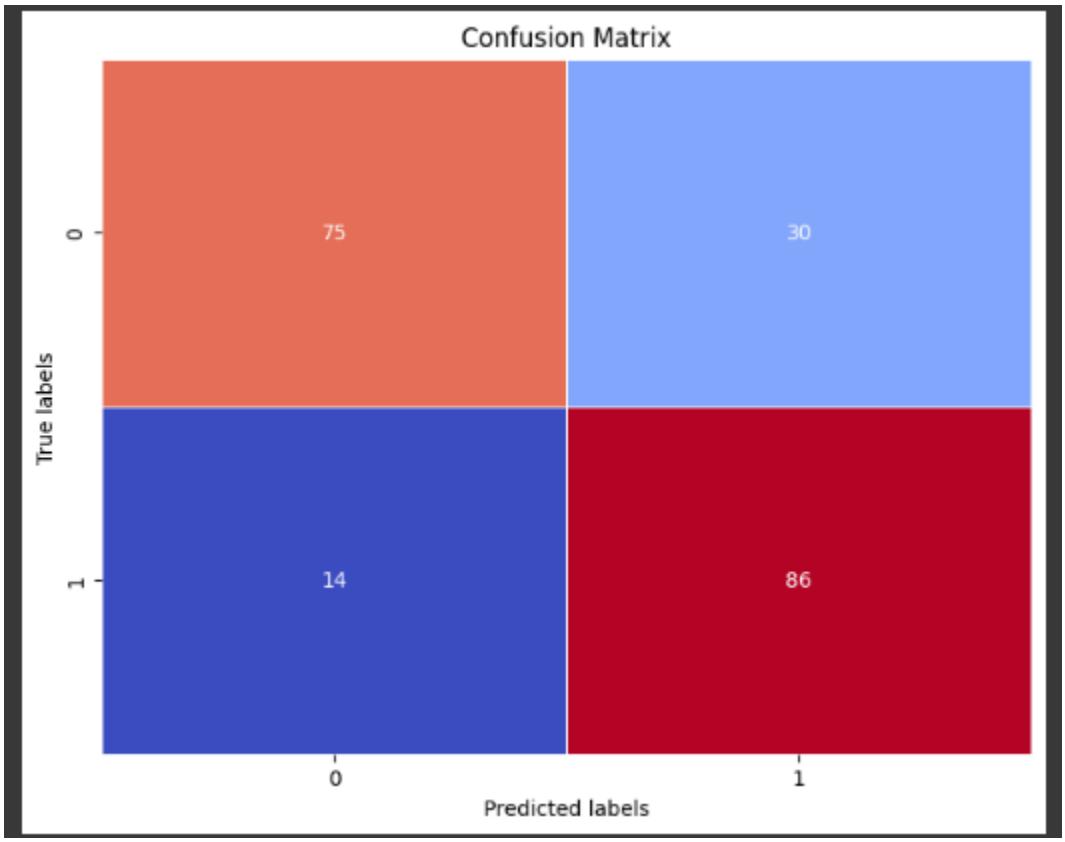
```python
# Separate features and target
X = df.drop(columns=['target'])
y = df['target']

# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=84)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train Gaussian Naive Bayes classifier
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Classification Report
class_report = classification_report(y_test, y_pred, digits=2, zero_division=1)
print("\nClassification Report:")
print(class_report)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="coolwarm", cbar=False, linewidths=0.5)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

حالا نتایج را مشاهده می کنیم :

Classification report به صورت زیر است :

```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.71      0.77       105
           1       0.74      0.86      0.80       100

    accuracy                           0.79       205
   macro avg       0.79      0.79      0.78       205
weighted avg       0.79      0.79      0.78       205
```

Confusion matrix هم به صورت زیر است :



حالا نتایج را تحلیل می کنیم :

۱. گزارش طبقه‌بندی :(Classification Report)

گزارش طبقه‌بندی شامل چهار ستون است: precision(دقت)، recall(بازیابی)، f1-scoreو support(پشتیبانی).

این گزارش برای هر کلاس (۰ و ۱) و همچنین میانگین‌های کلی نمایش داده شده است. در ادامه بخش های مختلف آن را تحلیل می کنم :

Precision یا دقت : نسبت تعداد پیش‌بینی‌های درست از کلاس مثبت به کل پیش‌بینی‌های کلاس مثبت. برای کلاس ۰، دقت ۰.۸۴ است، به این معنی که ۸۴٪ از نمونه‌هایی که به عنوان کلاس ۰ پیش‌بینی شده‌اند، واقعاً کلاس ۰ هستند. برای کلاس ۱، این مقدار ۰.۷۴ است.

Recall یا بازیابی : نسبت تعداد پیش‌بینی‌های درست از کلاس مثبت به کل نمونه‌های واقعی کلاس مثبت. برای کلاس ۰، بازیابی ۰.۷۱ است، به این معنی که ۷۱٪ از نمونه‌های واقعی کلاس ۰ به درستی پیش‌بینی شده‌اند. برای کلاس ۱، این مقدار ۰.۸۶ است.

F1-score : میانگین هارمونیک دقت و بازیابی. برای کلاس ۰، امتیاز F1 برابر با ۰.۷۷ است و برای کلاس ۱، این مقدار ۰.۸۰ است.

Support یا پشتیبانی : تعداد نمونه‌های واقعی هر کلاس در مجموعه داده. برای کلاس ۰، تعداد ۱۰۵ نمونه و برای کلاس ۱، تعداد ۱۰۰ نمونه وجود دارد.

Accuracy یا دقت کلی : نسبت تعداد پیش‌بینی‌های درست به کل نمونه‌ها، که برابر با ۰.۷۹ یا ۷۹٪ است.

Macro avg یا میانگین کل : میانگین ساده دقت، بازیابی و امتیاز F1 برای هر کلاس. هر کدام برابر با ۰.۷۹، ۰.۷۹ و ۰.۷۸ است.

Weighted avg یا میانگین وزن‌دار : میانگین وزن‌دار دقت، بازیابی و امتیاز F1 برای هر کلاس، که به تعداد نمونه‌های هر کلاس وزن داده می‌شود. هر کدام برابر با ۰.۷۹، ۰.۷۹ و ۰.۷۸ است.

۲. ماتریس درهم‌ریختگی :(Confusion Matrix)

ماتریس درهم‌ریختگی، پیش‌بینی‌های مدل را در برابر مقادیر واقعی نشان می‌دهد. این ماتریس یک جدول دو در دو است که به صورت زیر تفسیر می‌شود: (با توجه به اعداد بدست آمده در ماتریس

۷۵ (بالا سمت چپ): تعداد نمونه‌های واقعی کلاس ۰ که به درستی به عنوان کلاس ۰ پیش‌بینی شده‌اند.

۳۰ (بالا سمت راست): تعداد نمونه‌های واقعی کلاس ۰ که به اشتباه به عنوان کلاس ۱ پیش‌بینی شده‌اند.

۱۴ (پایین سمت چپ): تعداد نمونه‌های واقعی کلاس ۱ که به اشتباه به عنوان کلاس ۰ پیش‌بینی شده‌اند.

۸۶ (پایین سمت راست): تعداد نمونه‌های واقعی کلاس ۱ که به درستی به عنوان کلاس ۱ پیش‌بینی شده‌اند.

پس این مدل درکل عملکرد مناسبی دارد اما بهبودهای بیشتری نیز ممکن است. دقت کلی مدل ۷۹٪ است، اما بازیابی برای کلاس ۰ نسبت به کلاس ۱ کمتر است (۰.۷۱ در مقابل ۰.۸۶). این نشان می‌دهد که مدل در پیش‌بینی نمونه‌های کلاس ۱ عملکرد بهتری دارد.

حالا برای اینکه بررسی کنم آیا با انتخاب بهینه فیچر ها، می توانم دقت مدل را افزایش دهم یا نه (یا کلا کار کردن با فیچر کمتر بهتر است و حتی اگر با همین دقت هم بتوان فیچر کمتری داشت، بهینه تر است) یک تابع به صورت زیر تعریف می کنم که میاد یک مقدار تحت عنوان threshold میگیرد و فیچر هایی که با لیبل هم بستگی بیشتر از مقدار threshold دارند را نگه میدارد.

برای هر مرحله هم confiusion matrix و classification report را ارائه می دهد. همچنین مشخص می کند که کدام فیچر ها شامل این شرط می شوند و در مدل باقی می مانند.

کد تابع به صورت زیر است :

```
def process_and_train(df, threshold=0.25):
    # Calculate the correlation of each feature with the label
    correlations = df.corr()['target'][:-1]  # Exclude the correlation of
Y with itself

    # Get the absolute value of correlations
```

```
    abs_correlations = correlations.abs()

    # Sort the correlations by absolute value in descending order
    sorted_correlations = abs_correlations.sort_values(ascending=False)

    # Print the sorted correlations
    print("Correlation of each feature with the label:")
    print(sorted_correlations)

    # Define a threshold for relevance, e.g., keep features with
correlation > threshold
    relevant_features = sorted_correlations[sorted_correlations >
threshold].index.tolist()

    # Print the most relevant features
    print("Most relevant features based on correlation:")
    print(relevant_features)

    # Select relevant features
    x_new = df[relevant_features]
    y_new = df['target']

    # Splitting the dataset into the Training set and Test set
    X_train_new, X_test_new, y_train_new, y_test_new =
train_test_split(x_new, y_new, test_size=0.2, random_state=84)

    # Standardize features by removing the mean and scaling to unit
variance
    scaler = StandardScaler()
    X_train_new = scaler.fit_transform(X_train_new)
    X_test_new = scaler.transform(X_test_new)

    # Create and train Gaussian Naive Bayes classifier
    classifier = GaussianNB()
    classifier.fit(X_train_new, y_train_new)

    # Predicting the Test set results
    y_pred = classifier.predict(X_test_new)

    # Confusion Matrix
    conf_matrix = confusion_matrix(y_test_new, y_pred)
    print("Confusion Matrix:")
    print(conf_matrix)

    # Classification Report
```

```
    class_report = classification_report(y_test_new, y_pred, digits=2,
zero_division=1)
    print("\nClassification Report:")
    print(class_report)

    # Plot Confusion Matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="coolwarm",
cbar=False, linewidths=0.5)
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title('Confusion Matrix')
    plt.show()
```

همچنین به کمک کد زیر (که بخشی از تابع است و برای هر threshold بررسی می شود.) ، این تابع میاد ۵ داده به صورت تصادفی از مجموعه تست انتخاب می کند و خروجی واقعی را با خروجی پیش بینی شده مقایسه می کند.

Random.seed را هم برابر دو رقم آخر شماره دانشجویی قرار دادم که با هر بار اجرای برنامه، یک خروجی خاص حاصل شود.

```
# Set the random seed
random.seed(84)

# Randomly select 5 indices from the test set
random_indices = random.sample(range(len(X_test_new)), 5)

print("Randomly selected 5 data points:")
for idx in random_indices:
    print(f"Index: {idx}")
    print(f"Actual Output: {y_test_new.iloc[idx]}")
    print(f"Predicted Output: {y_pred[idx]}")
    print()
```

حال مقادیر مختلفی برای threshold قرار می دهیم تا مقایسه کنیم و بهترین حالت و بهترین فیچر هارا انتخاب کنیم :

threshold=0.15

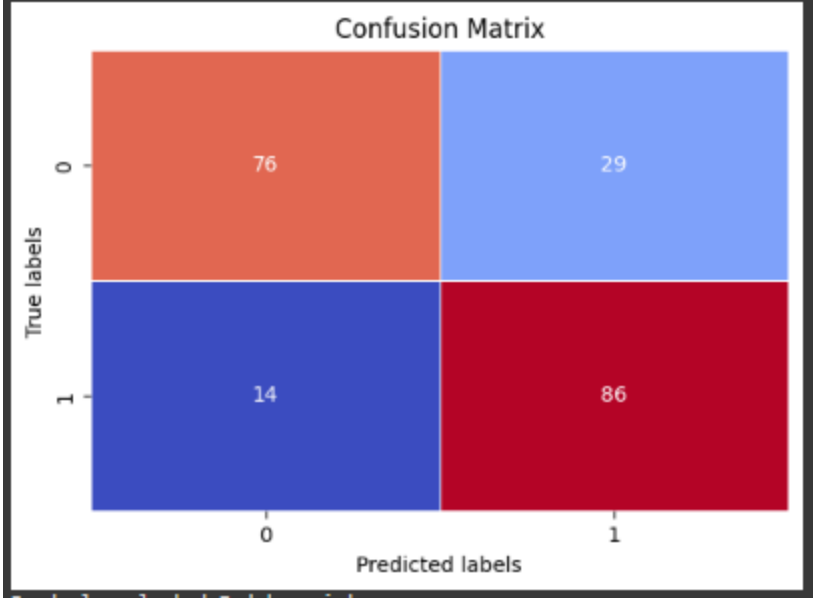
در این حالت ۹ فیچر میماند که هم بستگی بیشتر از ۰.۱۵ با لیبل دارند که به صورت زیر اند:

```
process_and_train(df, threshold=0.15)
```

```
Name: target, dtype: float64
Most relevant features based on correlation:
['oldpeak', 'exang', 'cp', 'thalach', 'ca', 'slope', 'thal', 'sex', 'age']
```

خروجی و عملکرد این حالت به صورت زیر است :

```
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.72      0.78       105
           1       0.75      0.86      0.80       100

    accuracy                           0.79       205
   macro avg       0.80      0.79      0.79       205
weighted avg       0.80      0.79      0.79       205
```



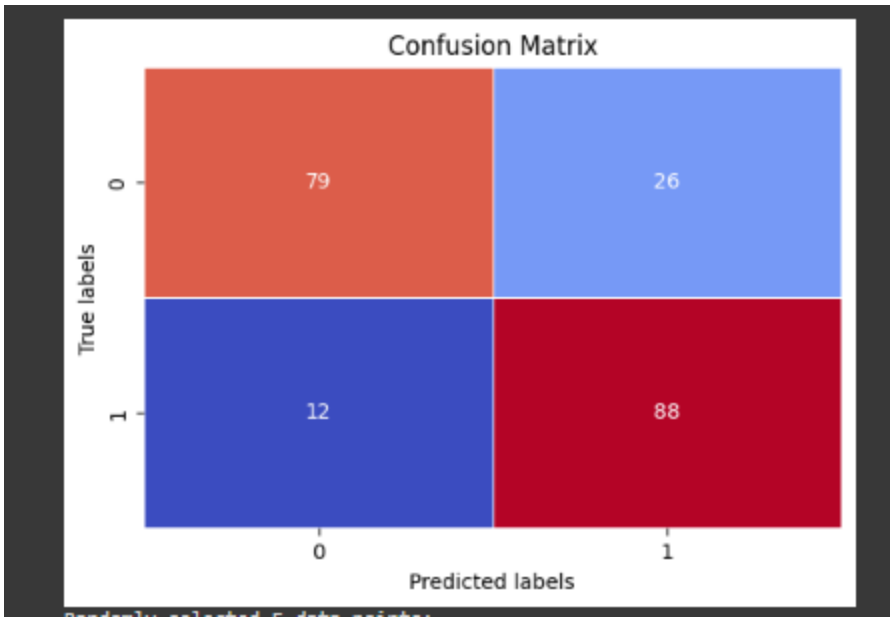
threshold=0.2

نتایج مانند حالت قبل است.

threshold=0.25

در این حالت تعداد فیچر ها ۸ تا شد و داریم :

```
process_and_train(df, threshold=0.25)

Most relevant features based on correlation:
['oldpeak', 'exang', 'cp', 'thalach', 'ca', 'slope', 'thal', 'sex']
Confusion Matrix:
[[79 26]
 [12 88]]

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.75      0.81       105
           1       0.77      0.88      0.82       100

    accuracy                           0.81       205
   macro avg       0.82      0.82      0.81       205
weighted avg       0.82      0.81      0.81       205
```



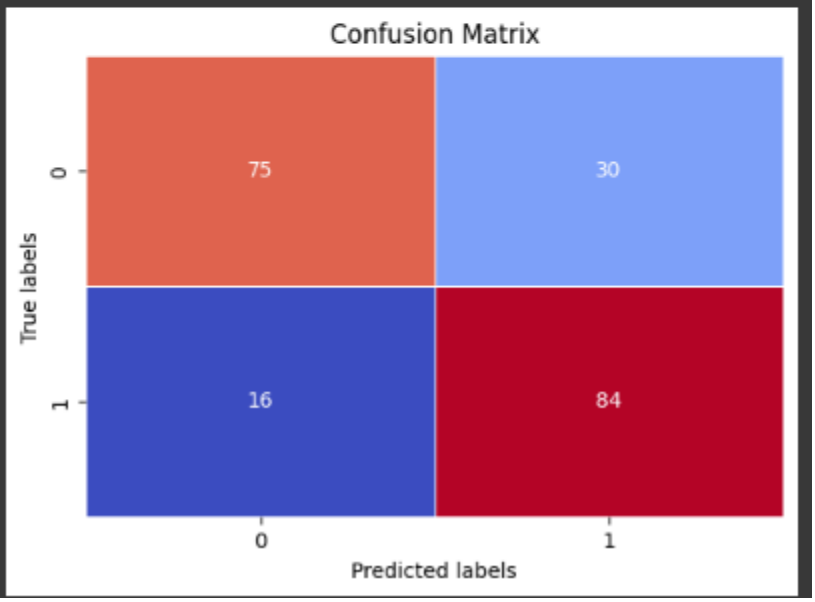
threshold=0.3

در این حالت تعداد فیچر ها ۷ تا شد و داریم:

threshold=0.35

در این حالت هم تعداد فیچر ها ۵ تاشد و داریم :

```
process_and_train(df, threshold=0.35)

Most relevant features based on correlation:
['oldpeak', 'exang', 'cp', 'thalach', 'ca']
Confusion Matrix:
[[75 30]
 [16 84]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.71      0.77       105
           1       0.74      0.84      0.79       100

    accuracy                           0.78       205
   macro avg       0.78      0.78      0.78       205
weighted avg       0.78      0.78      0.77       205
```



Threshold از این مقدار بیشتر دقت مدل را زیر ۷۵ درصد میاره.. پس در نتیجه کار رو بیشتر ازین ادامه نمی دهیم و بین این حالت هایی که تا الان بیان کردیم، بهترین را انتخاب می کنیم.

در حالتی که از تمام ۱۳ فیچر استفاده کردیم دقت کلی ۷۹ درصد شد، وقتی از فیچر هایی که هم بستگی بالای ۰.۱۵ درصد یا ۰.۲ درصد با لیبل دارند استفاده شد، دقت همان ۷۹ درصد شد ولی تعداد فیچر ها به ۹ فیچر کاهش یافت. با افزایش threshold و استفاده از فیچر هایی با همبستگی بالا ۰.۲۵ درصد با لیبل ، تعداد فیچر ها ۸ فیچر شد و دقت مدل ۸۱ درصد شد، در حالت استفاده از هم بستگی بالای ۰.۳ هم تعداد فیچر ها ۷ فیچر شد و دقت مدل ۸۰ درصد شد و با هم بستگی بالاتر ازین مقدار، دقت روال نزولی

گرفت **پس بهترین حالت همان انتخاب ۸ فیچر و هم بستگی ۰.۲۵ درصد فیچر ها با لیبل ها هستند که دقت حدود ۸۱ درصد حاصل شد.**

حالا با توجه به این حالت، ۵ داده تست به صورت تصادفی در این مدل (که ۸ فیچر دارد) انتخاب می کنیم و داریم :

```
Randomly selected 5 data points:
Index: 187
Actual Output: 1
Predicted Output: 1

Index: 72
Actual Output: 0
Predicted Output: 0

Index: 198
Actual Output: 0
Predicted Output: 1

Index: 9
Actual Output: 0
Predicted Output: 1

Index: 125
Actual Output: 0
Predicted Output: 0
```

دیتاهایی با ایندکس هایی مانند شکل فوق به طور رندوم انتخاب شدند که مقدار واقعی و پیش بینی شده آن ها را می بینیم.

سه تا از این ۵ تا درست تشخیص داده شده اند اما این صرفا یک ازمایش رندوم است و دقت کلی این حالت حدود ۸۱ درصد است. یعنی حدودا ۴ تا از ۵ دیتا درست پیش بینی خواهند شد.

تفاوت دو حالت میانگین گیری در کتاب خانه سایکیت لرن :

در کتابخانه scikit-learn، هنگام محاسبه معیارهایی مانند دقت، بازخوانی و امتیاز F1 برای مسائل دسته‌بندی چندکلاسه، دو روش میانگین‌گیری وجود دارد: ماکرو و میکرو . در ادامه به توضیح این دو حالت و مقایسه آن ها میپردازیم :

میانگین ماکرو (Macro Average)

در روش میانگین ماکرو، معیارها برای هر کلاس به صورت مستقل محاسبه شده و سپس میانگین آن‌ها برای تمام کلاس‌ها گرفته می‌شود. این روش، تمامی کلاس‌ها را به طور یکسان و بدون توجه به توازن و تعداد آن‌ها در مجموعه داده در نظر می‌گیرد. میانگین ماکرو زمانی مفید است که بخواهیم عملکرد کلی مدل را برای همه کلاس‌ها بدون در نظر گرفتن تفاوت‌های تعداد نمونه‌ها ارزیابی کنیم. در این روش، میانگین دقت، بازخوانی یا امتیاز F1 برای هر کلاس، بدون توجه به توازن کلاسی، محاسبه می‌شود.

میانگین میکرو (Micro Average)

در روش میانگین میکرو، ابتدا مجموع مثبت‌های واقعی، مثبت‌های کاذب و منفی‌های کاذب برای همه کلاس‌ها محاسبه می‌شود و سپس دقت، بازخوانی و امتیاز F1 بر اساس این مقادیر کلی محاسبه می‌گردد. این روش با در نظر گرفتن تفاوت‌های تعداد نمونه‌ها در کلاس‌ها عمل می‌کند، زیرا هر نمونه به طور مساوی وزن داده می‌شود. میانگین میکرو زمانی مفید است که بخواهیم عملکرد کلی مدل را در نظر بگیریم و تفاوت‌های تعداد نمونه‌ها در کلاس‌ها را نیز مد نظر داشته باشیم. در این روش، دقت، بازخوانی یا امتیاز F1 برای تمامی کلاس‌ها با توجه به مقادیر کلی محاسبه می‌شود.

به طور کلی، میانگین ماکرو تمام کلاس‌ها را به طور یکسان در نظر می‌گیرد و زمانی که اهمیت هر کلاس یکسان باشد، مناسب است. از طرفی، میانگین میکرو با توجه به تفاوت تعداد نمونه‌ها در کلاس‌ها عمل می‌کند و زمانی که مجموعه داده ناهموار باشد و بخواهیم وزن بیشتری به کلاس‌های با نمونه‌های بیشتر بدهیم، مناسب است.

در classification_reportکتابخانه scikit-learn، می‌توان با استفاده از پارامتر averageنوع میانگین‌گیری را مشخص کرد. تنظیم average=macro معیارها را با استفاده از میانگین ماکرو محاسبه می‌کند، در حالی که average=microاز میانگین میکرو استفاده می‌کند. اگر average=Noneباشد، معیارها برای هر کلاس به صورت جداگانه بازگردانده می‌شوند.