بسم تعالی

جزوه خودنویس آموزش یادگیری ماشین

گام های اجرایی پروژه کلاسیفایر ها

تاریخ شروع: 1404/05/15

تاریخ پایان:

نویسنده: عرفان جناب

سلام. بلاخره رسیدیم به فصل بعد، پروژه بعد و یک آموزش دیگه

ما توی این فصل با کلاسیفایرها (classification) ها آشنا می شویم. اما قبل از اینکه کلا بریم روی کد و پروژه رو استارت بزنیم بریم ببینیم فرق این پروژه با پروژه قبلی چیه و اینکه تفاوت اصلی بین regression و classification چی هست.

classification یا "دسته‌بندی" یعنی مدلی بسازیم که داده‌ها را در یکی از چند دسته مشخص قرار دهد. مثلاً تشخیص اینکه یک ایمیل اسپم هست یا نه، یا اینکه یک تصویر مربوط به گربه است یا سگ .خروجی classification همیشه گسسته (Discrete) است؛ یعنی فقط می‌تواند بین چند مقدار از قبل تعریف‌شده انتخاب کند) مثل ۰ یا ۱ یا کلاس A و B وC.

در مقابل، Regression یا "رگرسیون" برای زمانی است که بخواهیم یک مقدار عددی و پیوسته پیش‌بینی کنیم. مثلاً پیش‌بینی قیمت یک خانه یا دمای هوا. خروجی regression می‌تواند هر عددی باشد (مثلاً ۲۵۴.۶ یا ۱۳۰۰۰۰).

خب حالا که با مفهوم اصلی این موضوع آشنا شدیم بایستی به سراغ دیتایی که قرار است با آن کار کنیم برویم. اسم آن دیتای معروف MNIST هست. اما طبق معمول حتما سوال پیش میاد این دیتاست چیه چجوری کار میکنه و... در ادامه توضیحاتی گفته خواهد شد که ابهامات زیادی را برطرف می کند.

دیتاست MNIST مجموعه‌ای از تصاویر عددهای دست‌نویس ۰ تا ۹ است که برای آموزش و تست مدل‌های تشخیص تصویر و دسته‌بندی (classification) استفاده می‌شود. این دیتاست شامل ۶۰,۰۰۰ تصویر برای آموزش و ۱۰,۰۰۰ تصویر برای تست است. هر تصویر یک عدد است که به‌صورت سیاه و سفید و با اندازه‌ی ۲۸ در ۲۸ پیکسل ذخیره شده. چون این دیتاست ساده و معروف است، معمولاً اولین انتخاب برای تمرین و یادگیری الگوریتم‌های یادگیری ماشین مثل شبکه‌های عصبی و SVM و ... است. هدف اصلی در MNIST این است که مدل یاد بگیرد از روی تصویر، تشخیص دهد کدام عدد دست‌نویس است.

این دیتاست که از سال ۱۹۹۴ به وجود آمده تا به اکنون ما می توانیم در پروژه آموزش خود استفاده کنیم. این کار نه تنها باعث تقویت ما در این پروژه می شود بلکه این امید را می دهد که ما در حال کار با دیتاست واقعی هستیم.

خب بدون معطلی بریم سراغ پروژه.

معرفی و دریافت دیتا:

1. اولین کار یا بهتره بگیم مهمترین کاری که اول هر پروژه باید انجام دهیم دسترسی به دیتاست آن پروژه است. برای این کار ما نیاز به دانلود فایلی اعم از csv نداریم بلکه به لطف کتابخانه قدرتمند sklearn با کلاسdatasets این کار به سادگی شدنی است. بنابراین ما کافی است تا از طریق این کلاس به آن دیتاست معروف دسترسی پیدا کنیم و آن را داخل یک متغیر بریزیم.

پس از دسترسی، نوبت به مرحله همیشگی ما یعنی برسی اولیه دیتا می رسد. در این جا ما بایستی یکسری اطلاعات ضروری درباره دیتاست را به دست آوریم. در نخستین کار با استفاده از کتابخانه keys() یک لیستی از متد هایی که در درون دیتاست ما وجود دارد وارد می کنیم، سپس می بینیم که دیتاست کلیدهای متنوعی دارد اعم از data, target, DESCR, details, url, and... که مهمترین آنها را توضیح خواهیم داد:

Data: این متد شامل لیستی از اعداد بین ۰ تا ۲۵۵ است که یک طیف بین سفید تا مشکی را تشکیل می دهد. (بر روی matplot امکان پذیر است) این لیست شامل ۷۰۰۰۰ شماره در ۷۸۴ ردیف و ستون است.

Target: این متد شامل لیستی است از اعداد بین ۰ تا ۹ است و نشان می دهد در هر ترکیب ۷۸۴ تایی data خروجی چه عددی است.

DESCR: این متد کامل ترین تعریف را نسبت به دیتاست خروجی می دهد.

Details: این متد یکسری مشخصات دیگری مجزا از متد قبل در اختیار ما قرار می دهد.

Url: این متد لینک یا آدرس جایی که از آنجا به دیتا دسترسی پیدا کردیم را نمایش می دهد.

به طور کلی برای نمایش اعداد داخل لیست data به صورت عکس می بایست با استفاده از کتابخانه matplot اقدام کنیم. اما قبل از آن باید بدانیم برای این کار باید دیتایمان را به shape ۲۸*۲۸ تبدیل کنیم.

نمونه ای از دیتای تبدیل شده به عکس دیتای شماره ۲ که این عدد در دیتای target نشان دهنده ۴ است.

نویسنده : عرفان جناب

یکی از نکات مهمی که باید بدانیم این است که دیتاست mnist به طور کلی نرمال شده و پاکسازی یافته است. نرمال شدن از دو جهت صورت گرفته یکی از جهات آن قرار گرفتن همه تصاویر پیکسلی در وسط تصویر عکس است و دیگری تبدیل همه دیتا ها به محدوده ۲۸*۲۸ هستند. با این حال نوبت به آن رسیده است که به سراغ آموزش مدل برویم. (نگران نباشید پروژه به همین سادگی اتمام نمی یابد).

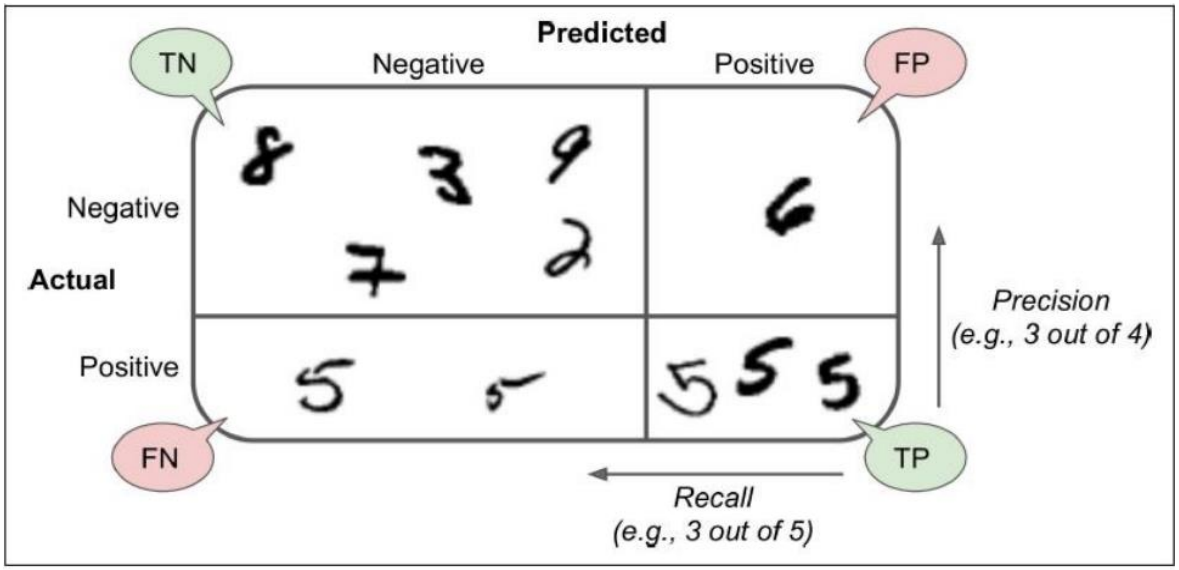
2. قبل آموزش باید بدانیم هدف از این پروژه این است که اگر دیتای جدیدی از اعداد ۰ تا ۹ را به مدل دادیم توانایی تشخیص آن را داشته باشد. با این حساب بایستی دیتایمان را قبل از شروع یادگیری به دسته های ۰ تا ۹ تبدیل کنیم. اما برای شروع ما بهتر است به جای تبدیل سریع ابتدا با تبدیل ساده ۲تایی شروع کنیم. عدد پیشنهادی برای شروع جداسازی عدد ۵ هستش که می توان استفاده کرد. یعنی از دیتای target اعم از بخش train و test یک لیستی از true و false تشکیل دهیم. تا با دادن به مدل آموزشی SGDClassifier در sklearn.linear_model یادگیری را آغاز کنیم. اما قبل از همه این کار ها بایستی هردو دیتای data و target را به دو بخش train و test جدا کنیم. نکته قابل توجه این است که دیتای target به طور کلی یک آرایه string است و ما بایستی با کتابخانه numpy آن را تبدیل به اعداد کنیم. بایستی بدانیم که دیتاست mnist جه در بخش data و در بخش target ۶۰۰۰۰ هزار دیتای اول دیتای آموزش و ۱۰۰۰۰ دیتای بعد از آن دیتای تست هستند. با این حساب پس از دسترسی به کلاس SGDClassifier میتوانیم با دادن دو متغیر x_train و y_train_5 یادگیری مدل را آغاز کنیم. و سپس با استفاده از یک دیتای درون متغیر x برسی میکنیم که ببینیم آیا درست یادگرفته است یا خیر. اصولا اساس کار این کلاس با استفاده از گرادیان کاهشی است. اما گرادیان کاهشی چیست و چگونه کار می کند؟ در نمودار های ما (ما در پروژه آن هارا نمی بینیم یا به صورت های دیگر ظاهر می شوند) این کلاس با استفاده از یک رابطه ساده پایین ترین نقطه در آن را پیدا می کند. این نشان می دهد که ما گرادیان افزایشی نیز داریم. اما چرا کمترین؟ بسته به نوع دیتا ما می توانیم بین گرادیان کاهشی و افزایشی یکی را انتخاب کنیم بنابراین در اینجا ما از کاهشی استفاده می کنیم. و اما آن رابطه چیست؟ $x_{n+1}=x_n-\gamma_n\nabla F(x_n)$ در این رابطه x دیتای ورودی ما و دلتا f تابع مشتق ما و γ گاما ضریبی است که ما مشخص می کنیم در هر بار چه مقدار به سمت پایین ترین نقطه حرکت کند. مثال: فرض کنید در یک نمودار پایین ترین نقطه، نقطه ۳- است و مدل برای شروع از نقطه ۳ شروع می کند. بنابراین مدل بسته به مثبت یا منفی بودن مشتق شروع به حرکت می کند تا به پایین ترین سطح نمودار برسد، هرچه γ کمتر باشد دقت بیشتر است. {مطالب بیشتر در پیوست A }

پس از آموزش مدل و انجام یک تست ساده متوجه شدیم که مدل درست پیش بینی کرده است، اما این معیار کافی برای اینکه بگوییم مدل ما خوب است نیست بلکه باید با استفاده از اعتبارسنجی تست هایی بر روی دیتای آموزش انجام دهیم ولی چطور؟ ما با استفاده از کلاس cross_val_score از داخل کتابخانه sklearn.model_selection می توانیم به راحتی دیتای آموزش را به چند دسته تقسیم کرده و هر بار یک دسته را به عنوان اعتبارسنجی و الباقی دسته ها را به عنوان آموزش نگه داریم. این کار باعث می شود آموزش و اعتبارسنجی بر روی تمامی دیتا انجام شود تا عملکرد نهایی آموزش مدل برسی شود.{مطالب بیشتر در پیوست B }

با خروجی گرفتن از عملکرد مدل SGDClassifier متوجه می شویم مدل بسیار عملکرد خوبی داشته است، زیرا خروجی درصد پیش بینی برابر با ٪ ۹۶ است. اما نکته قابل توجه اینجاست که ما نباید به این عملکرد خوب، اعتماد کنیم. اما چرا؟ اولین دلیل بخاطر این است که ما برسی عملکرد را بر روی تنها ۰.۱ دیتای کل مان سنجیدیم (y_train_5). یعنی ما دیتای آموزش را برای تنها دیتاهایی که عدد ۵ هستند ارزیابی کردیم و این یعنی از بین اعداد ۰ تا ۹ تنها با ۰.۱ اعداد این تست انجام شده است. دلیل دوم این است که دیتای در حال حاضر ما skewed dataset است.(skewed dataset به معنای مجموعه داده هایی که تعداد نمونه های کلاس هایشان نامتعادل باشد. مثلا ۹۰ درصد دیتا عدد ۱ و ۱۰ درصد دیتا عدد ۲ باشد) برای اینجور دیتا ها cross validation عملکرد مناسبی ندارد. برای اطمینان از این موضوع که عملکرد این کلاس مناسب دیتاست ما نیست می توانیم از کلاس Dummyclassifier از کتابخانه sklearn.dummy استفاده کنیم. اما این کلاس چه کاری انجام می دهد؟ این مدل یک مدل خیلی ساده و ابتدایی هستش که اصلاً دنبال یادگیری واقعی از داده‌ها نمی باشد، فقط یک استراتژی از پیش تعیین‌شده را برای پیش‌بینی دنبال می کند. هدف نهایی این کلاس این است که بفهمد آیا پیش بینی مدل واقعی از یک مدل نسبتا احمق بهتر کار می کند یا خیر. اگر خروجی این مدل از مدل SGDClassifier بهتر باشد یعنی بیشتر از ٪۹۶ باشد. مدل یا دیتا نیاز به اصلاح دارد.{مطالب بیشتر در پیوست C }

3. در ادامه برای اینکه بفهمیم مدل ما تا چه حد می تواند کلاس A را از B تشخیص دهد بایستی به سراغ چیزی به اسم ماتریس درهم ریختگی برویم. اما این ماتریس چه کاری انجام می دهد؟ بزارید با توضیح یک شکل شروع کنیم. این شکل به خوبی نمایانگر یک ماتریس درهم ریخته در پروژه ما است. این ماتریس از بخش ستون ها و ردیف ها تفسیر می شود. بخش ستون ها بخش پیش بینی شده و بخش ردیف ها بخش مقادیر واقعی است. در ستون اول negative به معنی مقادیری هستند که مدل آنها را غیر از ۵ پیش بینی کرده یا (تشخیص داده است)، در ستون دوم positive مقداری است که مدل آنها را ۵ پیش بینی کرده و تشخیص داده است. در ردیف ها هم به همین شکل است ردیف اول negative مقادیری هستند که واقعا ۵ نیستند و ردیف بعدی مقادیری هستند که واقعا ۵ هستند، با این وجود همانطور که در سلول ها مشخص است مقادیری که ۵ نبودند و مدل آنها را درست تشخیص داده در سلول بالا سمت چپ هستند. در این حالت که مقدار پیش بینی درست باشد خروجی TN یا همان true negative است. باقی ستون ها هم به همین شکل برسی می شوند.

نویسنده : عرفان جناب



شکل ۲-۳ یک ماتریس درهم‌ریختگی.

اما قبل از خروجی ماتریس گرفتن بایستی با استفاده از روش cross validation دیتای آموزش را به فولد های مختلف تقسیم (بسته بندی) کرده و روی آنها ارزیابی انجام دهیم. در کلاس cross_val_score برای ما مشخص می کند که مدل در هر بخش چه امتیازی دریافت کرده است، اما کلاس cross_val_predict با cross validation دیتا ها را جدا کرده و در آخر با پیش بینی و ارزیابی کلی بر روی کل دیتا ها یک خروجی متریکی (ماتریس) می دهد. با چنین خروجی برای به نمایش در آوردن آن به صورت یک ماتریس درهم ریخته باید از کلاس confusion_matrix در کتابخانه sklearn.metrics استفاده کرد. این به گونه ای است که در کلاس cross_val_score متغیر x_train و y_train_5 را داده و با قرار دادن cv = 3 آنها را به ۳ فولد تقسیم می کنیم و خروجی آن را در یک متغیر ریخته و آن را به عنوان اولین ورودی و مقادیر واقعی (y_train_5) را به عنوان دومین ورودی به کلاس confusion_matrix می دهیم. خروجی چنین چیزی است:

[[۶۸۷ و ۵۳۸۹۲]

[۳۵۳۰ و ۱۸۹۱]]

همانطور که قبلا توضیح دادیم این خروجی ماتریس ما است. عدد ۵۳۸۹۲ مقداری است که هم مقدار پیش بینی و هم مقدار واقعی negative بوده است یعنی از ۶۰ هزار دیتا نزدیک به ۵۴ هزار دیتا عدد ۵ نیستند بنابراین مقادیر آن TN می شود. سپس عدد ۶۸۷ مقداری است که در واقعیت عدد ۵ نیستند اما مدل آنها را به اشتباه positive یا ۵ تشخیص داده است و در نتیجه خروجی آن FP در نظر گرفته می شود. عدد ۱۸۹۱ مقادیری است که در واقعیت ۵ هستند اما مدل به اشتباه آن ها را عدد دیگری پیش بینی یا تشخیص داده است و خروجی آن FN در نظر گرفته می شود. عدد ۳۵۳۰ مقادیری است که در واقعیت عدد ۵ هستند و همچنین مدل آن ها را ۵ تشخیص داده است در نتیجه خروجی TP است.

نکته مهم: در خروجی های ماتریسی مقادیری که false positive یا همان FP ارور نوع ۱ هستند.

و در ماتریس هایی که مقادیر false negative یا همان FN ارور نوع ۲ هستند.

اما شاید سوالی در ذهنتان ایجاد شود و آن این است که از کجای ماتریس می توان به عالی بودن مدل پی برد؟ پاسخ این است که مدلی در پیش بینی حرف اول را می زند که مقادیر FN و FP آن برابر ۰ باشد. برای این آزمایش می توانیم علاوه بر اینکه مقدار واقعی را به کلاس ماتریس می دهیم یک کپی از مقدار واقعی گرفته و آن را به عنوان مقدار پیش بینی به آن بدهیم. اتفاقی که می افتد این است که مقادیر پیش بینی و واقعی با هم برابرند (یعنی بهترین مدل ممکن) در نتیجه ماتریس خروجی به این شکل می شود.

[[۰ و ۵۴۵۷۹]

[۵۴۲۱ و ۰]]

همانطور که می بینید مقادیر FN و FP برابر صفر است.

به طور کلی فرمول محاسبه accuracy یا همان دقت کل برابر با جمع مقدار TN و TP تقسیم بر کل دیتا است که قبلا حساب کردیم چیزی حدود ۹۶٪ شد که مورد قبول و یاری کننده نبود. با این حساب از همین ماتریس درهم ریخته میتوان اطلاعاتی بسیار مفیدی از طریق فرمول استخراج کرد.

یکی از این فرمول هایی که می تواند به ما در تحلیل مدل کمک کنند precision است، اما این precision چه کاری انجام می دهد؟

این فرمول که از تقسیم TP بر جمع TP و FP به دست می آید نشان دهنده این است که مدل ما از میان اعداد پنجی که توانسته پیش بینی کند چند درصد آن ها را درست (مثبت) پیش بینی یا همان تشخیص داده است. هرچه عدد این فرمول بیشتر باشد یعنی به ۱۰۰٪ نزدیک تر باشد بدین معناست که مدل بیشتر اعدادی ۵ توانسته بشناسد درست تشخیص داده است.

فرمول دیگری که با آن سر و کار داریم فرمول recall است این فرمول از رابطه تقسیم TP بر جمع TP و FN به دست می آید و نشان دهنده این است که مدل چقدر توانسته از تمام نمونه هایی که واقعا مثبت بوده اند درست تشخیص دهد.

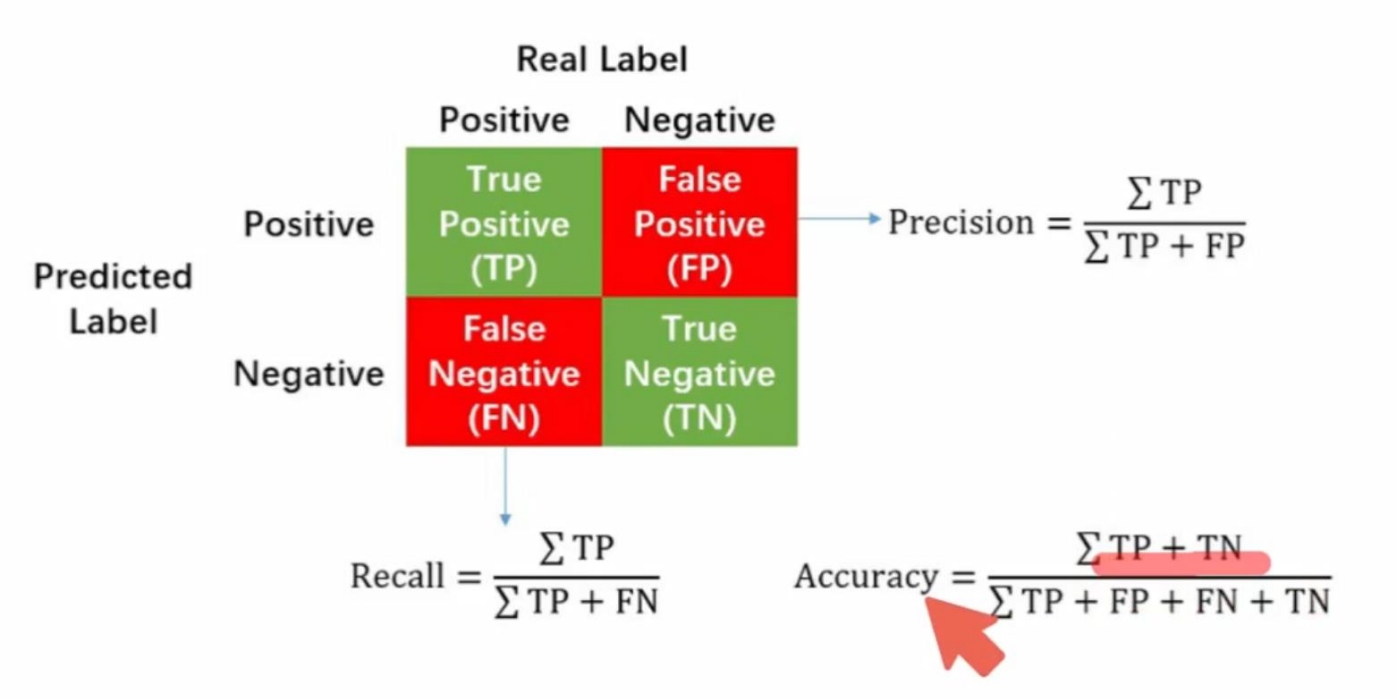
تفاوت اصلی precision و recall این است که precision مقادیری که درست تشخیص داده تقسیم بر جمع همان به علاوه مقادیری می شود که در واقعیت ۵ نبودند اما مدل به اشتباه آن ها را ۵ پیش بینی کرده است و در حقیقت (اشتباه) پیش بینی کرده است.

در مقابل آن recall مقادیری که درست تشخیص داده تقسیم بر جمع همان به علاوه مقادیری می شود که در واقعیت ۵ بودند اما مدل آن ها را به اشتباه یک عدد دیگری تشخیص داده است یعنی در حقیقت مدل در اینجا اشتباه نکرده بلکه آن ها را یعنی نمونه های مثبت را (از دست داده) است.

پس به طور کلی فرمول اول درصد اشتباهات مدل را در مواردی که توانسته ۵ تشخیص دهد (هرچند به غلط) و مدل دوم درصد از دست دادن یا (تشخیص ندادن) مدل را ارزیابی می کند.

در پروژه ما مقدار خروجی precision برابر با ٪ ۸۳.۷ و مقدار خروجی racall برابر با ٪۶۵.۱ می باشد. این یعنی ۱۷ درصد نمونه ها واقعا ۵ نبوند اما به اشتباه ۵ تشخیص داده شدند و ۳۵ درصد نمونه ها ۵ بودند اما از دست مدل در رفته و عدد دیگری تشخیص داده شدند.

اگر بخواهیم به سراغ کدینگ این بخش برویم بایستی به این نکته توجه کنیم که نیازی به محاسبه دستی اعداد این دو فرمول نیست بلکه ما می توانیم با استفاده از دو کلاس precision_score و recall_score در کتابخانه sklearn.metris این کار را انجام دهیم و ما کافیست مقادیر خروجی از کلاس cross_val_prediction را به همراه مقادیر واقعی به آنها داده و آن ها را دریافت کنیم.

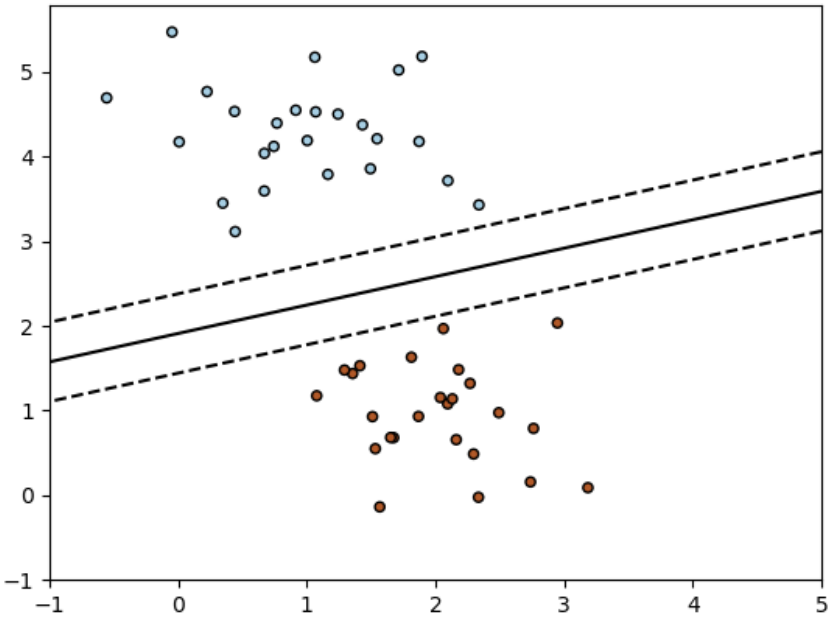


در ادامه ما برای اینکه نتیجه درست تری نسبت به این دو عدد به دست آمده بگیریم می توان با استفاده از هر دوی آنها از طریق روش میانگین هارمونی به خروجی مطلوب تری برسیم. اما یعنی چی آیا این دو شاخص به تنهایی اطلاعات خوبی به ما نمی دهند؟ خیر اینطور نیست. بلکه ما در پروژه های مختلف مجبور به نظر گرفتن هردوی این شاخص ها هستیم و باید هردوی اینها در معیار عملکرد مل تاثیر بگذارند و برای اینکه یک چکیده ای از معیار عملکرد به دست آوریم می توانیم از طریق روش میانگین هم ساز یا هارمونی به معیار کلی تر و متعادل می رسیم. روش محاسبه آن هم از طریق ضرب این دو شاخص تقسیم بر جمع آنها ضربدر ۲ به دست می آید. هرچه precision و recall کمتر باشد خروجی آن هم عدد کوچک تری است. خروجی f1_score برابر با ٪۷۳ است که از رابطه بین این دو شاخص به دست آمده است. ولی در بسیاری از مواقع یک شاخص کلی (مثل f1_score) کافی نیست. برای مثال فرض کنید قصد ساختن مدلی را داریم که با برسی ویدیو های مختلف صحنه هایی را که دارای خشونت و ۱۸+ هستند (که برای کودکان مناسب نیست) را تشخیص دهد در اینجا ما تنها به precision نیاز داریم زیرا میخواهیم از همان نمونه هایی که تشخیص می دهیم درست تشخیص داده باشیم ولا اینکه تعدادی از آنها را تشخیص نداده باشیم. مورد دیگر این است که فرض کنید در دوربین های مداربسته در تشخیص دزدی مهم ترین عامل از بین این دو شاخص recall است زیرا هرچه تعداد بیشتری دزدی تشخیص داده شود در اینجور مواقع بهتر است هرچند که رابطه بین recall و precision برعکس هم دیگر هستند یعنی با افزایش تشخیص تعداد دزدی

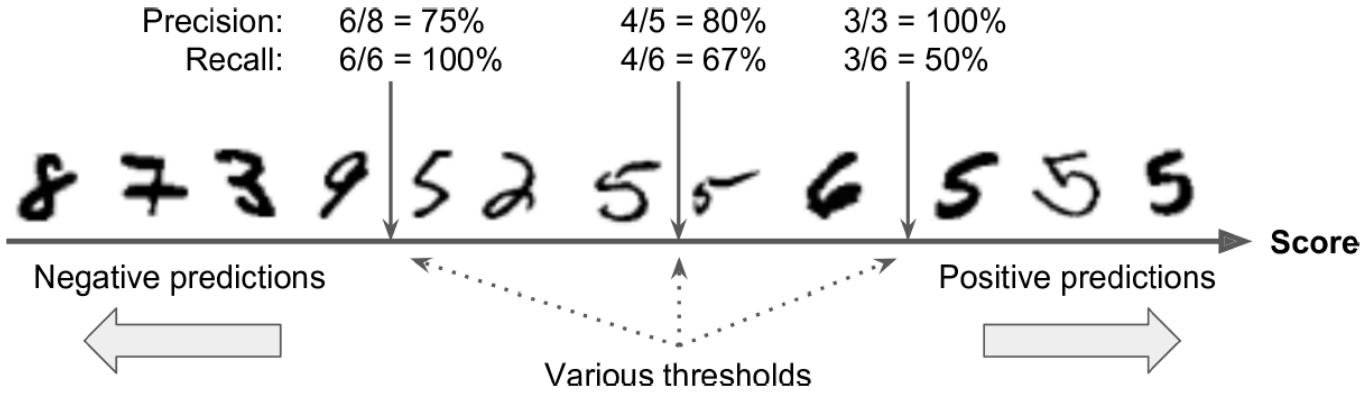
نویسنده : عرفان جناب

بیشتر دقت آن پایین می آید و بلعکس اما در اینجور مواقع دقت را فدای تعداد می کنیم زیرا کوچک ترین عملی می تواند دزدی باشد و ما باید آن را شناسایی کنیم.

ما بایستی تلاش خودمان را بکنیم تا بتوانیم بالاترین معیار عملکرد precision (بالاترین درصد) را بدست بیاوریم. قبل از آن بایستی به برسی چندین عکس بپردازیم:



این عکس نشان دهنده بارز روش جدا کردن دو کلاس است. خطی که صاف است خطی به نام hyperplane است که معیار عملکرد جداسازی ۲ یا چند کلاس از یکدیگر است خطوط خط چین خطوط threshold نام دارند که با تغییر در فاصله آنها نسبت به hyperplane می توانیم منجر به تغییر میزان precision و recall شویم، با این حال وظیفه ما این است که در اینجا با تغییر در میزان threshold به بالاترین درصد از دقت یا همان precision برسیم. در این نمودار بایستی بدانیم که فاصله نقاط تا خط hyperplane اگر نسبت به خط چین بالای آن کمتر باشد در دسته نقاط قرمز و اگر فاصله آن بیشتر باشد در دسته نقاط آبی قرار می گیرد، اگر ما نقاط آبی را ۵ در نظر بگیریم بایستی با تغییر در فاصله خط چین بالایی به دقت بالاتری برسیم. باید بدانیم که هرچه فاصله خطوط threshold نسبت به خط hyperplane بیشتر شود مرز اطمینان نیز بیشتر می شود و دقت بالاتر می رود هرچند تعدادی نقاط یا دیتایی را از دست داده (یعنی وارد کلاس نقاط قرمز می شود) باعث کاهش recall می شود اما هدف اصلی ما افزایش precision است.



این عکس نمایانگر پروژه فعلی ما است. اگر در مرحله اول خط threshold را خط وسطی در نظر بگیریم در سمت راست محدوده اعدادی است که ۵ تشخیص داده و سمت چپ بلعکس آن. به این ترتیب همانطور که می بینید در سمت راست ۵ دیتا را ۵ تشخیص داده که تنها ۴ تای آنها دست بوده است بنابراین دقت در این نقطه threshold برابر با ٪۸۰ است از آن طرف در این عکس به طور کلی ۶ دیتای ۵ داریم که در این نقطه از threshold ۴ تای آنها ۵ شناخته شده بنابراین recall برابر با ٪۶۷ شده است. دیگر نقاط threshold نیز با تغییرشان باعث تغییر در میزان precision و recall شده اند.

اما اگر بخواهیم بحث کدینگ این قسمت را پیش ببریم باید بدانیم که در کلاس SGDClassifier تابعی به نام decision_function وجود دارد که با دادن دیتا به آن فاصله (score) آن نقطه نسبت به خط hyperplane بدست می آورد و این به ما کمک می کند تا با استفاده از این فاصله ها بتوانیم بهتر

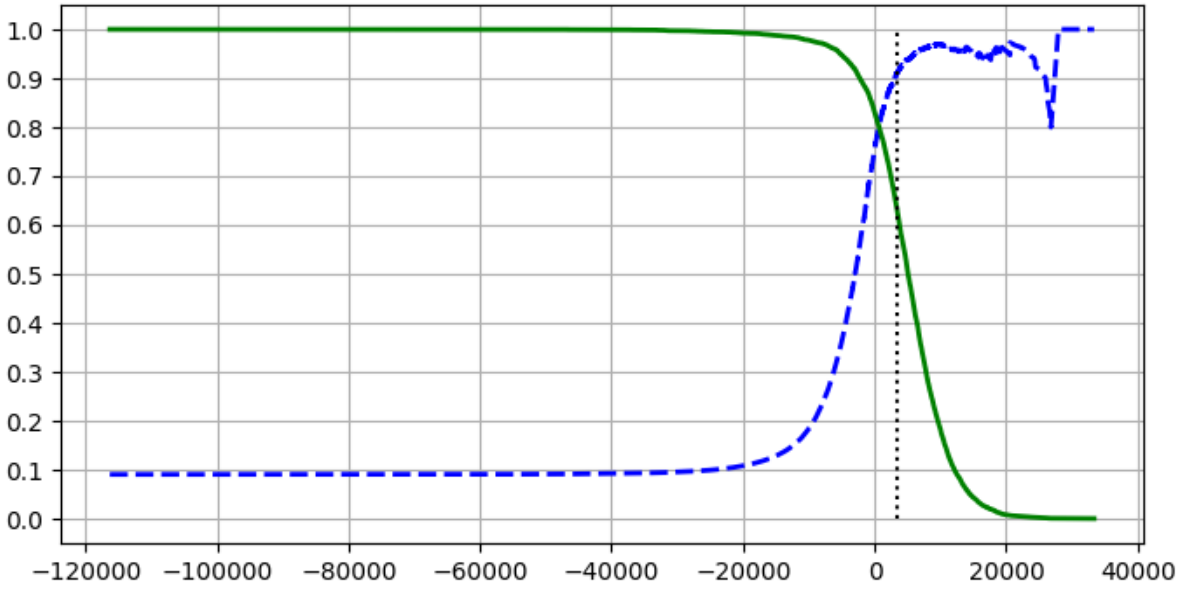
threshold را تغییر دهیم. اما سوالی که در این وسط پیش میاید این است که ما که نمی توانیم به ترتیب ۶۰۰۰۰ تا دیتا را داده و مقدار precision و recall آنها را اندازه گیری کنیم پس تکلیف چیست؟ باید بدانیم که کلاس cross_val_predict این کار را برای ما راحت کرده و در متد scoring ما با دادن المان decision_function می توانیم تمام فاصله ها یا scors ها را به دست بیاوریم و در یک متغیر بریزیم.

حال که ما یک لیست بزرگ از فاصله ها داریم بایستی به محاسبه threshold تک تک آنها بپردازیم تا با داشتن ۳ المان precision، recall و threshold بهترین عملکرد را برسی و استخراج کنیم. برای این کار کلاس precision_recall_curve از تابع sklearn.metris به ما کمک می کند و باعث می شود ما با داشتن آن ۳ المان بهترین دقت را پیدا کنیم، اما قبل از آن این کلاس دقیقا چه کاری انجام می دهد؟ این کلاس با استفاده از فاصله دیتا ها مقادیر threshold آنها به همراه precision و recall اندازه گیری می کند.

حال با داشتن این اطلاعات تنها راه برسی دقیق رسم نمودار است که از طریق سورس کد زیر امکان پذیر است:

```
plt.figure(figsize=(8, 4))
plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
plt.vlines(threshold, 0, 1.0, "k", "dotted", label="threshold")
plt.yticks(np.arange(0, 1.1, 0.05))
plt.grid()
plt.show()
```

دو خط یکی خط چین آبی نسبت precision ها به threshold ها را نشان می دهد و دیگری خط صاف سبز که نسبت recall ها به threshold ها را نشان می دهد. خط چهارم که خط چین سیاه عمودی است برای مقایسه کردن عملکرد های متفاوت است.



و اما چند نکته راجب به نمودار:

precision و recall همیشه عکس هم حرکت می‌کنن: وقتی یکی زیاد می شود ، دیگری یکی کم می شود.

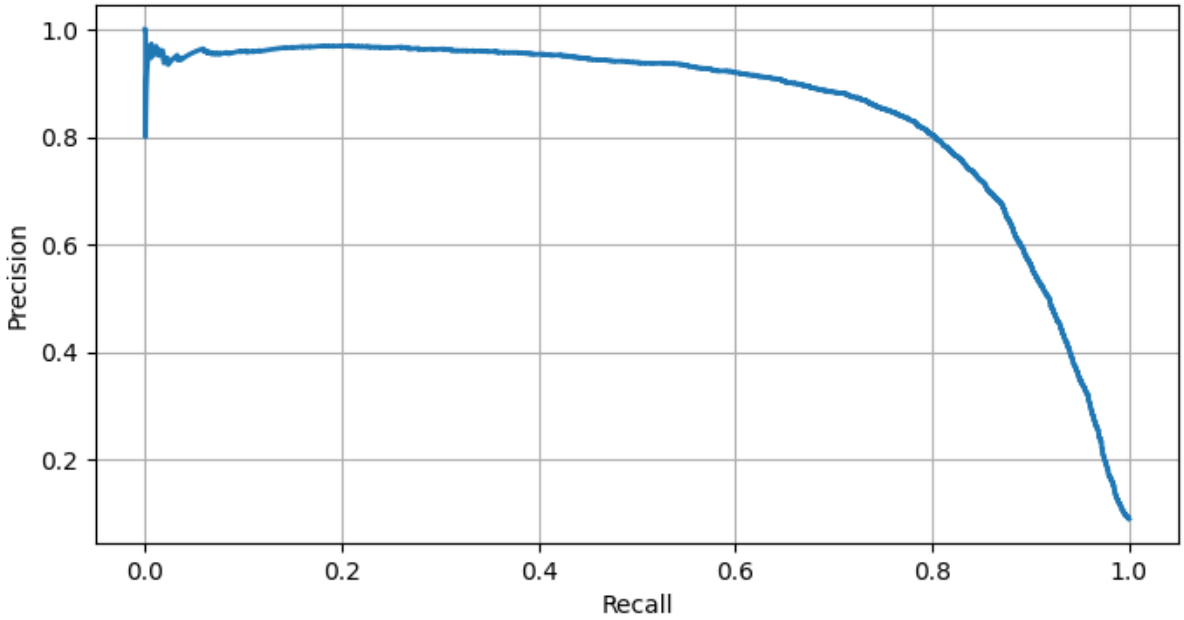
این همون trade-off معروف در مدل‌های دودویی هست.

حوالی threshold = 0 ، precision و recall به هم نزدیک می شوند و هر دو حدود ۰.۸–۰.۹ هستند.

این نقطه معمولاً بهترین تعادل بین precision و recall است.(trade-off point)

بعد از مقدار threshold = 15000 فرایند صعودی precision به مشکل می خورد و این نشان دهده وجود نویز در دیتا ها می باشد که می توان به آن رسیدگی کرد.

نویسنده : عرفان جناب

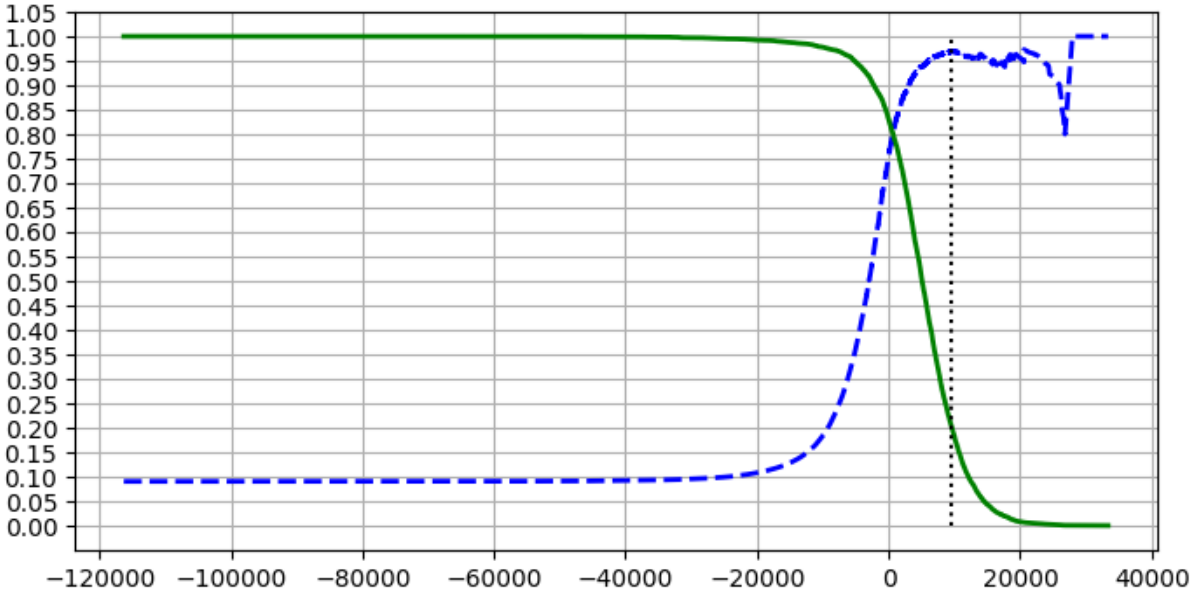


این نمودار، نمودار نسبت recall به precision است.

در این نمودار از یک عددی به بعد precision فرایند نمودار نزولی شدید را پیش می گیرد که علت آن بر اثر افزایش threshold و سخت گیری بیشتر در تشخیص اعداد است.

با چشم غیر مسلح هم تا حدودی می توان بهترین معیار دقت یا همان precision را یافت اما برای اینکه دقیق تر باشیم می توانیم با استفاده از لیستی از precision ها که از کلاس precision_recall_curve به دست آورده ایم با دادن مقداری از دقت که میخواهیم مثلا بالاترین دقت یعنی ۹۷ درصد با استفاده از متد argmax نمونه ای که دقت آن برابر با ۹۷ درصد است بر می گرداند. اما متد argmax چطور عمل می کند؟ این متد با مقدار ورودی که ما به آن می دهیم در یک لیست از نمونه شماره ۰ تا اولین نمونه ای که با مقدار ورودی باشد برسی می کند سپس متوقف می شود و شماره index آن نمونه را در داخل یک متغیر می ریزد.

حال که شماره نمونه آن را پیدا کرده ایم کافیست آن شماره نمونه را در لیست threshold پیدا کرده مقدار threshold آن نمونه را بدست بیاوریم.

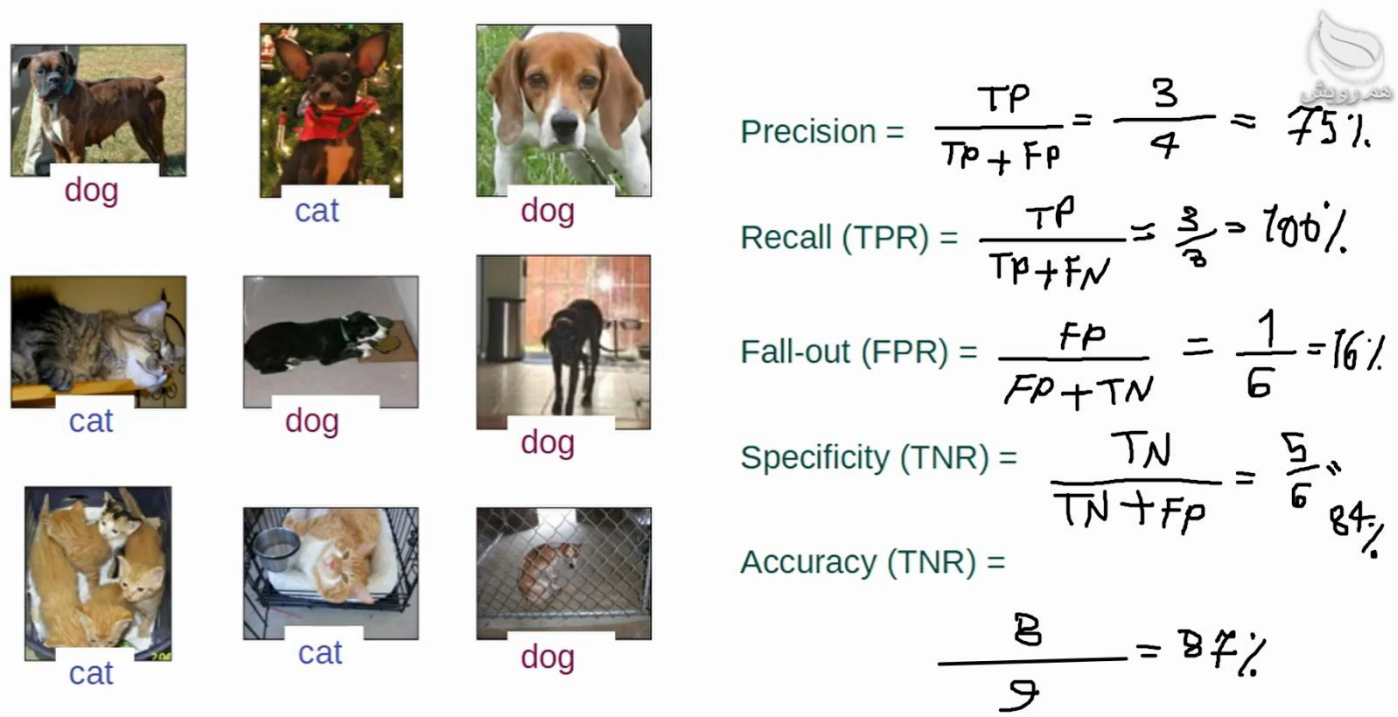


در ادامه با قرار دادن شماره index آن threshold مقادیر precision و recall که در نظر داشتیم به دست میاوریم.

اما به طور کلی در ماتریس درهم ریختگی به غیر از دو المان precision و recall المان های دیگری نیز وجود دارند که می توانند به ما در بهبود و برسی بهتر عملکرد مدل کمک کنند. این دو المان FPR و TNR هستند. نکته مهم: TPR همان recall است و توضیحات آن قبلا داده شده.

نویسنده : عرفان جناب

FPR یا همان fall_out به زبان خودمانی به معنای برسی میزان خطای مدل در تشخیص دیتا های غیر ۵ یا none_5 است. این المان برعکس precision است در precision هدف برسی دقت مدل و میزان درستی آن در تشخیص دیتاهای ۵ بود و این المان در تشخیص دیتا های غیر ۵ است. فرمول محاسبه آن برابر است با FP تقسیم بر جمع FP با TN است. المان دیگر یعنی TNR به معنای برسی میزان دقت و درستی مدل در تشخیص دیتا های غیر ۵ است و فرمول آن برابر است با TN تقسیم بر جمع TN با FP است. نکته قابل توجه این است که هم TNR و هم FPR از رابطه TNR = FPR – ۱ و بلعکس به دست می آیند. برای فهم بهتر این مفاهیم می توانیم به حل سوالات شکل کلاسیفایر زیر بپردازیم:



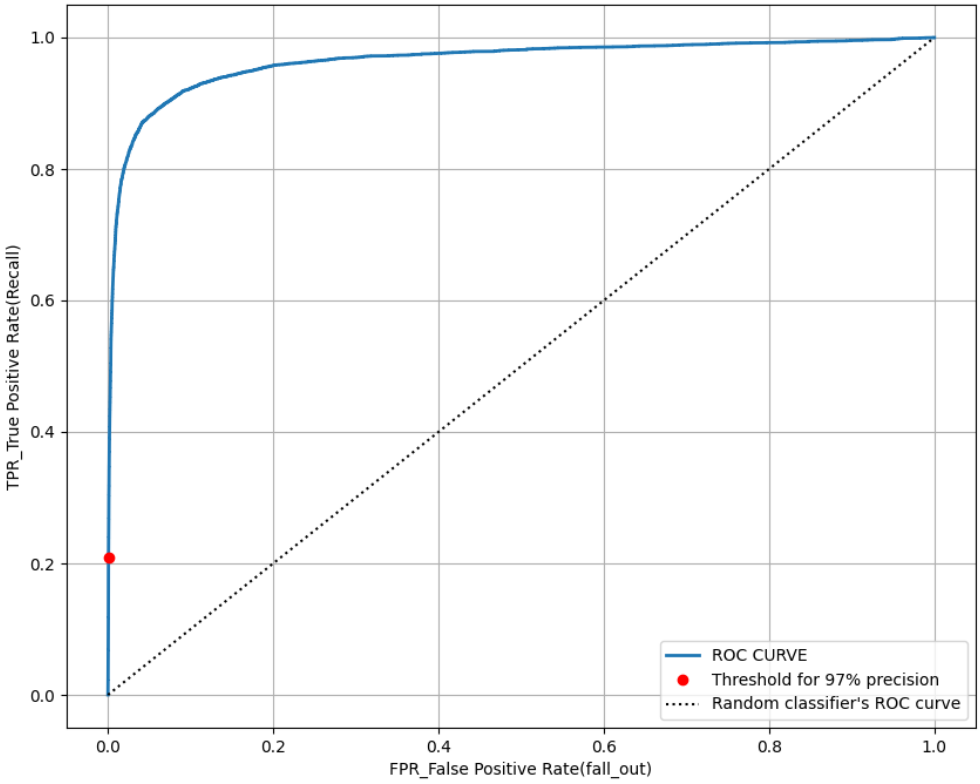
		Predicted		
		Positive	Negative	
Actual	Positive	True Positive (TP)	False Negative (FN)	**True Positive Rate, Recall** $\frac{TP}{TP+FN}$
	Negative	False Positive (FP)	True Negative (TN)	**False Positive Rate** $\frac{FP}{FP+TN}$ 1 – TNR
		Precision $\frac{TP}{TP+FP}$		**Accuracy** $\frac{TP+TN}{TP+TN+FP+FN}$

در ادامه ما فعلا کاری با المان TNR نداریم زیرا ما با داشتن یکی از آنها می توانیم دیگری را حساب کنیم بنابراین ما بایستی یک معیار عملکرد جدید با استفاده از recall و FPR به دست بیاوریم. برای اینکه بتوانیم این مقادیر از المان های جدید را بدست بیاوریم باید با استفاده از تابع roc_curve و دو متغیر y_train_5 و scors هایی که از تابع cross_val_predict بدست آوردیم به این تابع بدهیم. خروجی های آن برابر با یک لیست از threshold های مختلف با خروجی های FPR و recall(TPR) متناسب با آن است. خب در ادامه شاید برایتان سوال باشد در بین این لیست چطور بهترین المان ها را پیدا کنیم؟ برای این کار کافیست با استفاده از متد argmax() که قبلا هم از آن استفاده کرده ایم در بین لیست threshold جدید مقدار threshold قبلی را که با آن به precision ٪۹۷ رسیدیم را پیدا کنیم و پس از آن با آن مقدار threshold (که متناسب با دو لیست recall(TPR) و FPR است) مقدار FPR و recall (در precision ٪۹۷) را به دست بیاوریم.

نکته اساسی: ما قبلا مقدار recall (در precision ٪۹۷) را به دست آورده ایم و این مقدار از recall جدید با آن هیچ تفاوتی ندارد.

نویسنده : عرفان جناب

همانطور که می دانید تنها راه برسی دقیق و حرفه ای رسم نمودار است. بنابراین کاری که لازم از انجام دهیم نمودار نسبت FPR به recall را رسم کنیم.



خب همانطور که می دانید نکاتی در درون این شکل نهفته است که به برسی آن ها می پردازیم:

- در این شکل با افزایش FPR مقدار recall نیز افزایش می یابد.(رابطه مستقیم دارند)
- هرچه قوس خط آبی (ROC CURVE) به سمت گوشه سمت چپ نزدیک تر باشد مدل بهتر است.
- اگر خط آبی به صورت قائم باشد ما بهترین مدل را خواهیم داشت.
- خط نقطه چین یک خط از پیش بینی مدل است که با افزایش مقدار FPR مقدار recall همان اندازه به صورت خطی افزایش می یابد که این موضوع خوب نمی باشد.
- نقطه قرمز محل نشان دادن مقدار FPR و recall با threshold مشخص شده در نقطه precision ٪۹۷ است.
- مقدار FPR تقریبا برابر با صفر و چیزی حدود ۰.۰۶ است. این به معنای خطای کم در تشخیص داده های غیر ۵ است.

با نکات مهمی درباره شکل آشنا شدیم اما بیایید به برسی بیشتر آنها پردازیم. به طور کلی مساحت کل این نمودار برابر با مساحت مربع و برابر با ۱ است. بنابراین هر چه مساحت زیر خط آبی بیشتر باشد و قوس آن به سمت گوشه (چپ) نزدیک تر باشد مدل بهتر است. اما چرا؟ در ادامه به دلیل آن می پردازیم.

به طور کلی به برسی المان های مهم در مورد SGDClassifier پرداختیم اما چون در پروژه های هوش مصنوعی نباید تنها به یک مدل اطمینان کرد در اینجا بایستی به سراغ تست و برسی مدل های آموزشی دیگر برویم، یکی از این مدل ها RandomForestClassifier است که از کتابخانه ensemble قابل دسترسی است.(از خانواده مدل های ensemble می باشد) این مدل به صورت جنگل تصادفی به یادگیری دیتا و پیش بینی آنها می پردازد.

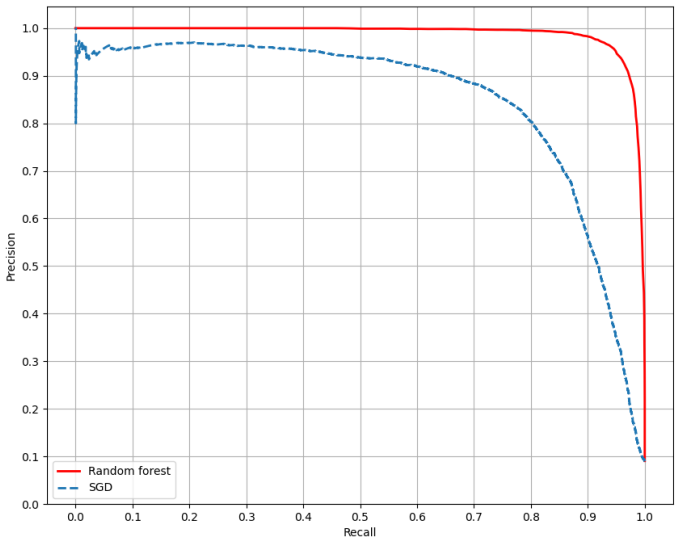
اگر که این مدل را به تابع cross_val_predict بدهیم در روش محاسبه و پیش بینی دو حالت متد دارد یکی predict و دیگری predict_proba است. اما این دو چه تفاوتی با یکدیگر دارند؟

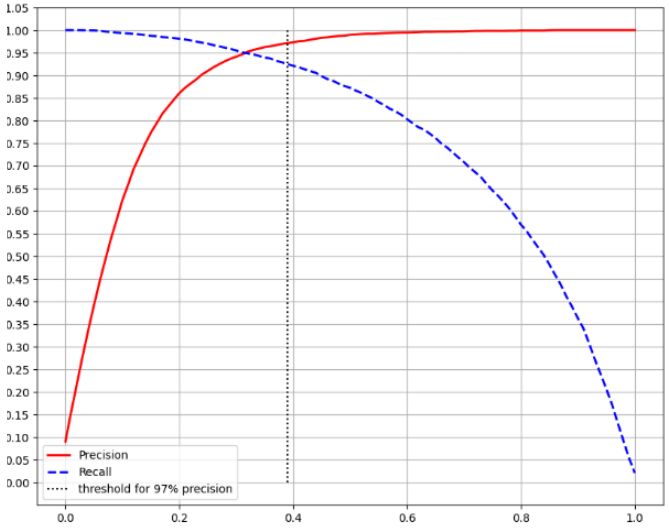
در predict مدل نظر نهایی و قطعی خودش را اعلام (خروجی) می دهد اما در predict_proba مدل به صورت یک لیست دو ستونه احتمال های موجود را خروجی می دهد. به مثال زیر می پردازیم:

```
array([[0.11, 0.89],
       [0.99, 0.01]])
```

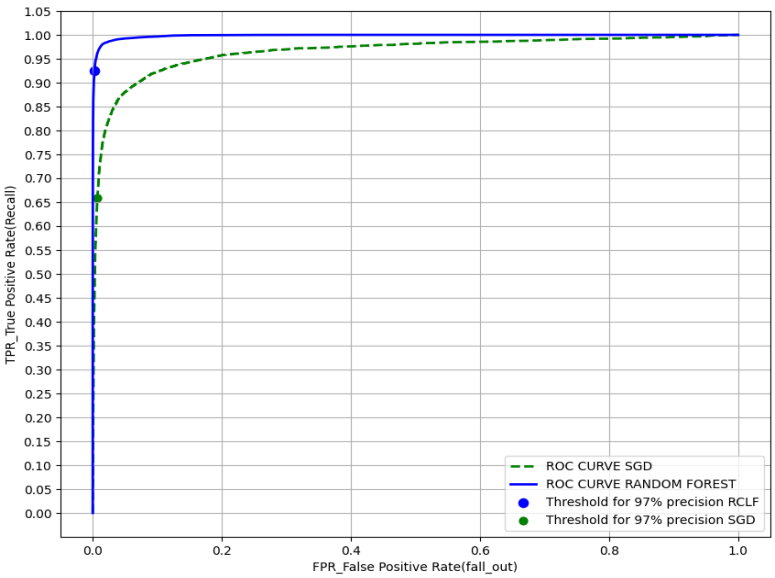
ما پس از پرینت دو مقدار اول مقدار خروجی cross_val_predict با متد predict_proba به این لیست از دو ستون مواجه می شویم. در ردیف اول دو ستون وجود دارد که ستون اول (سمت چپ) درصد پیش بینی ۵ نبودن اون دیتا را نشان می دهد و عدد ستون دوم (سمت راست) درصد پیش بینی ۵ بودن آن دیتا را نشان می دهد بنابراین به طور کلی در دیتای اولی احتمال آنکه عدد ۵ باشد حدود ٪۹۰ است. اما اگر متد بر روی predict باشد بدون دادن احتمالات تنها با دادن لیستی از اعداد ۰ و ۱ به ما می گویید که ۱ یعنی آن عدد ۵ است و ۰ یعنی آن عدد ۵ نیست.

بیایید برای برسی دقیق تر این مدل به سراغ اعداد و ارقام و نمودار های مرتبط برویم:





نمودار سمت چپ نمودار نسبت recall به precision است یعنی در واقع نمودار هر دو مدل SGD و Random در اینجا رسم شده است. همانطور که قبلا گفتیم و الان هم در شکل مشخص است نمودار قرمز (random) قوس آن به سمت گوشه (راست) نزدیک تر است و مساحت زیر خط آن بیشتر است و به ۱ نزدیک تر است. این کار باعث می شود با افزایش precision، recall نیز به اندازه زیادی افزایش یابد یعنی ما در این مدل علاوه بر اینکه از precision بالایی برخوردار هستیم از recall بالایی نیز برخوردار هستیم. بیایید با اعداد کار کنیم: در همین نمودار اگر ما بخواهیم مقدار precision ٪۹۷ داشته باشیم در مدل SGD مقدار recall چیزی حدود ٪۶۵ می شود اما در مدل random این مقدار به چیزی حدود ٪۹۲ می شود همچنین در نمودار سمت راست نیز این موضوع را می توان مطرح کرد زیرا نمودار سمت راست خط نمودار دو المان رسم شده و دو نکته مهمی که می توان از آن استخراج کرد این است که در یک نقطه از threshold مقدار این دو المان برابر با هم و مساوی با ٪۹۵ می شود و نکته بعدی این که در این نمودار نیز در نقطه ای با precision ٪۹۷ مقدار recall برابر با ٪۹۲ می باشد.



این نمودار این نشان دهنده نسبت FPR به TPR(recall) است که البته همانند نمودار قبلی نمودار هر دو مدل در اینجا برای برسی رسم شده اند. در این نمودار همانطور که می بینید خط آبی دارای مساحت بیشتری در زیر نمودار خود است و قوس به آن به گوشه (چپ) نزدیک تر است.

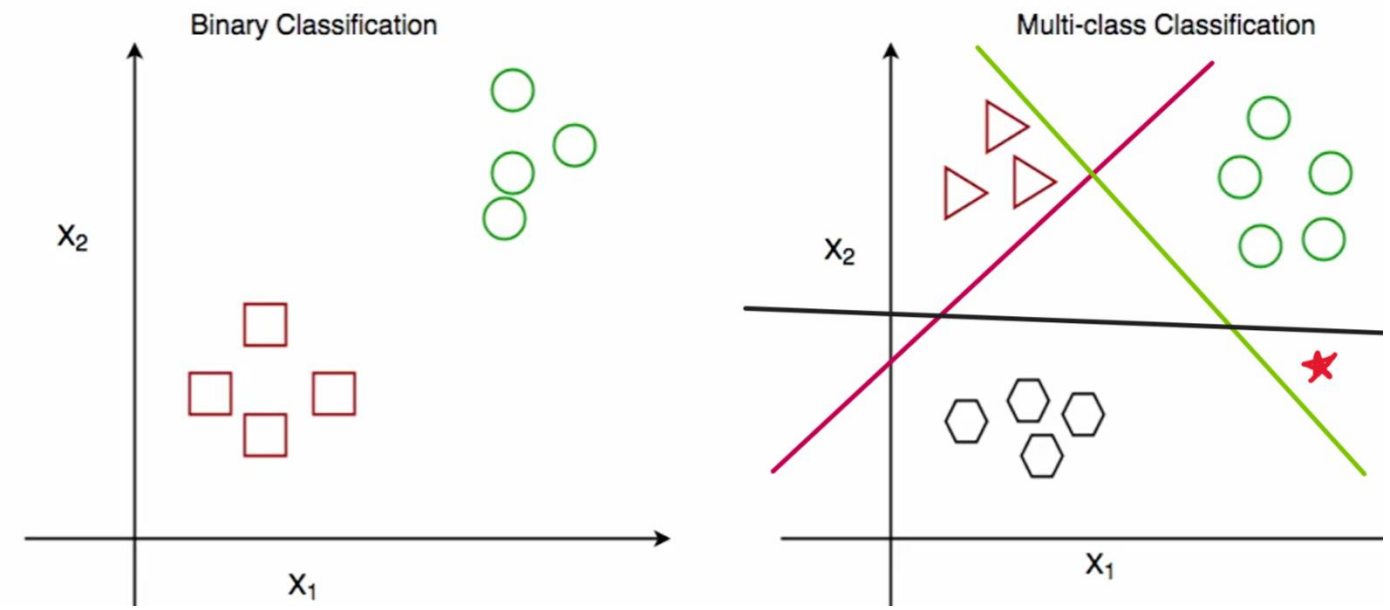
نویسنده : عرفان جناب

بنابراین این نکته را برای ما بازگو می کند که در مدل random با افزایش مقدار recall مقدار FPR نیز مقداری بسیار کم باقی می ماند در صورتی که در مدل دیگر یعنی SGD با افزایش مقدار recall مقدار FPR نیز افزایش می یافت.

یادآوری: recall به معنای این است که مدل از بین مقادیری که واقعا ۵ بودند چقدر آنها را ۵ تشخیص داده یا به عبارت برعکس چقدر از دیتاهای ۵ را از دست داده است. FPR نیز به معنای این است که مدل در تشخیص دیتای های غیر ۵ چه مقدار (اشتباه) کرده است.

ما تا به اینجای کار از طریق روش باینری به جدا کردن کلاس ها پرداختیم که همانطور که دیدیم روشی جذاب بود، اما روش دیگری نیز به نام multiclass classification وجود دارد که بعدا به آن خواهیم پرداخت. در همین روش باینری که ما با آن کار کرده ایم نیز به دو روش تقسیم شده و مجزا می شوند که به برسی کامل آنها خواهیم پرداخت:

روش اول OvA است که مخفف روش one_vs_all است (روشی که ما تا به اینجای کار پیش گرفته ایم) در این روش کاری که ما انجام می دهیم این است که هر کلاس را در برابر تمام کلاس های موجود قرار می دهیم و آنها را برسی می کنیم. اما این به چه معناست؟ این به این معناست که ما با ساختن یک کلاس یا متغیر جدید یک کلاس از کلاس هایی را که داریم مثال عدد ۵ در برابر تمام کلاس ها قرار می دهیم و آنهایی که ۵ پیش بینی یا تشخیص داده شدند را در آن متغیر می ریزیم. در این روش اگر ما ۱۰ کلاس داشته باشیم ۱۰ متغیر یا کلاس جدید تشکیل خواهیم داد. اما اگر بخواهیم عمیق تر به این موضوع بپردازیم بایستی به سراغ شکل برویم:



این شکل گویای توضیحات زیادی است. در شکل سمت راست همانطور که مشاهده می کنید خطوطی متناسب با کلاس های مربوطه برای جدا کردن آنها از یکدیگر کشیده شده است. خط سبز برای جدا کردن دایره های سبز از دیگر کلاس هاست و خط مشکی و بنفش نیز به همین ترتیب با این حال اگر یک دیتای جدید همانند ستاره قرمز را بخواهیم به یکی از این کلاس ها اضافه کنیم تکلیف چیست؟ در اینجا دو مرحله را پیش می گیریم. مرحله اول بایستی ببینیم که دیتای ما در زیر خط یا مرز کدام کلاس ها قرار دارد و همانطور که می بینیم در مرز بین کلاس های سبز و مشکی قرار دارد و کلاس بنفش به طور کل حذف می شود. در ادامه یعنی مرحله ۲ بایستی ببینیم که فاصله (score) آن ستاره (دیتا) نسبت به کدام خط دورتر است زیرا هرچه دورتر باشد ضریب اطمینان آن بیشتر است. در روش OvA نیز به همین شکل است چه در مرحله جداسازی دیتا و چه در مرحله پیش بینی دیتای جدید برای یافتن بهترین کلاس از فاصله آن یا score استفاده می شود و بیشترین فاصله ای را که از کلاسی داشته باشد در دسته آن کلاس در نظر گرفته می شود.

و اما روش دوم یعنی OvO که مخفف one_vs_one است روشی است که هر کلاس را به صورت جداگانه در برابر کلاس دیگر قرار می دهد و برای آن یک کلاس جدید به وجود می آورد. این به این معناست که ما اگر چهار کلاس داشته باشیم به نام های A,B,C,D در هر مرحله A به صورت جدا و مقابل هم در برابر هر ۳ کلاس دیگر یعنی B,C,D قرار می گیرد. و برای محاسبه تعداد کلاس ایجاد شده در آخر می توانیم از فرمول k*(k-1)/2 استفاده کنیم که در اینجا k تعداد کل کلاس های اولیه است. در اینجا 4*(4-1)/2 برابر با ۶ کلاس می شود. در این روش با ورود یک دیتای جدید آن دیتا در بین دو کلاس قرار می گیرد مثال بین دو کلاس A و B و پس از آن با انجام محاسبات از نظر ریاضیات آن دیتا به کلاسی شبیه تر است که فاصله آن تا خط hyperplane (خطوطی رنگی که در شکل می بینید) دور تر باشد حال چرا دورتر؟ زیرا این کار باعث افزایش ضریب اطمینان ما نسبت به آن دیتا می شود.

و اما معایب و فواید این دو کلاس:

OvA : فواید ← تعداد مدل‌ها کمه (فقط به اندازه‌ی تعداد کلاس‌ها ← k).

ساده‌تر و سریع‌تر آموزش داده میشه.

برای دیتاست‌هایی با کلاس‌های زیاد مناسب‌تره.

OvA : فواید ← داده‌ها نامتوازن میشن (یک کلاس در برابر همه‌ی کلاس‌های دیگه).

ممکنه توی مرزهای نزدیک بین کلاس‌ها دقت پایین بیاد.

احتمال هم‌پوشانی پیش‌بینی‌ها (چند مدل باهم یک نمونه رو انتخاب کنن).

OvO : فواید ← هر مدل فقط روی دو کلاس آموزش می‌بینه ⟵ دقت مرزی بهتر.

داده‌ی هر مدل کوچک‌تره ⟵ آموزش روی هر مدل سریع‌تره.

معمولاً دقت بالاتری در مسائل با کلاس‌های نزدیک به هم داره.

OvO : معایب ← تعداد مدل‌ها زیاده ⟵ k(k-1)/2 (برای ۱۰ کلاس = ۴۵ مدل).

ترکیب نتایج و رأی‌گیری پیچیده‌تره.

مدیریت منابع (حافظه و زمان) برای کلاس‌های زیاد سخت میشه.

در ادامه برای اینکه بتوانیم به سراغ برسی روش های محاسبه OvO و OvR برویم بایستی از یکسری کلاس ها استفاده کنیم و در آخر هر کدام معیار عملکرد ها را نیز برسی کنیم. اولین کلاسی که به سراغ آن می رویم کلاس SVC از sklearn.svm است. این کلاس به صورت پیش فرض به روش OvO به تحلیل دیتا می پردازد. ما با فراخوانی این کلاس و fit کردن آن بر روی x_train و y_train (نه y_train_5) می توانیم به اطلاعات خاصی دست پیدا کنیم. اولین مورد آن متد decision_function است که ما با دادن دیتا جدید (دیتای نمونه برای تست) می توانیم به یک لیست ۱۰ تایی از نمرات (score یا همان فاصله شان نسبت به دورترین کلاس) دسترسی داشته باشیم.

[3.79297828, 0.72949369, 6.06184129, 8.29800527, -0.29383983, 9.30157597, 1.74723215, 2.77365456, 7.20601456, 4.82245092]

این لیست خروجی است که از عدد سمت چپ لیست که برابر کلاس ۰ است شروع می شود و تا اخرین عدد لیست که برابر کلاس ۱۰ است ادامه دارد. این نمراتی که در اینجا مشاهده می کنید از طریق یکسری فرمول هایی بدست آمده اند. اما قبل از هر چیز بیایید به برسی عمیق تر این لیست بپردازیم.

- در این لیست عددی (کلاسی) به عنوان پیش بینی انتخاب می شود که بزرگ ترین عدد (فاصله تا hyperplane) را داد.
- اعداد منفی نشان دهنده کمترین شباهت یا نزدیکی آن عدد ورودی نسبت به آن کلاس را دارد.
- گمان نکنید که چون این لیست ده تایی است از طریق روش OvR استفاده شده است بلکه از همان روش OvO است و ما score ها را خروجی گرفته ایم

و اما روش محاسبه، اگر متد decision_function_shape را برابر با ovo قرار دهیم و سپس متد قبلی را خروجی بگیریم با یک لیست ۴۵ تایی مواجه می شویم (پشت پرده محاسبات اعداد لیست ۱۰ تایی) .

[0.11 , -0.21 , -0.97 , 0.5 , -1.0 , 0.18 , 0.08 , -0.31 , -0.03 , ...]

این لیست ده تای اول آن لیست ۴۵ تایی است یعنی مقایسه عدد ۰ با تمام اعداد ۱ تا ۹ به صورت مجزا می باشد. Score هایی که در اینجا ملاحظه می کنید برخی منفی و برخی مثبت هستند. این بدان معناست که دیتای ورود به کدام یک از دو کلاس در حال مقایسه شباهت بیشتری داشته و نزدیک تر بوده است.

کلاس ورودی همانطور به می دانید برابر عدد ۵ بود بنابراین در این لیست عدد ۵ ابتدا میان دو کلاس صفر و یک قرار می گیرد و به هر کدام که شباهت بیشتری داشته باشد نمره آن به سمت آن می رود اگر عدد مثبت باشد یعنی آن عدد به صفر (عدد یا کلاس مرجع در حال مقایسه) نزدیک تر است در اینجا نزدیک تر به معنای فاصله نیست بلکه به معنای میزان شباهت است و در آخر عدد خروجی آن یعنی ۰.۱۱ در لیست قرار می گیرد این عدد گویای آن است که عدد ۵ طبق پیش بینی مدل به عدد ۰ نزدیک تر است. اما یادتان باشد این همه ماجرا نیست لذا بایستی برای داشتن یک عدد مرجع قابل اعتماد به سراغ فرمول محاسبه آنها برویم. اما فرمول چیست؟

فرمول محاسبه شباهت یک دیتا به یک کلاس برابر است با:

جمع تمام اعداد موجود در لیست مربوط(با درنظر گرفتن مثبت و منفی بودن آنها) تقسیم بر margin.

و اما این margin چیست؟ به طور کلی اگر قرار باشد که ما تنها تعداد اعداد مثبت هر کلاس را در نظر بگیریم احتمال آنکه دو کلاس اعداشان شبیه به هم باشد وجود دارد لذا کاری که ما می توانیم انجام دهیم این است که با محاسبه فرمول margin و کم کردن آن از تعداد اعداد مثبت به یک عدد مناسبی برسیم.

و اما محاسبه margin:

X = جمع تمام اعداد موجود در لیست مربوط(با درنظر گرفتن مثبت و منفی بودن آنها)

Y = (|X| + 1) * 3

Margin : X / Y

و در آخر برای محاسبه میزان شباهت تعداد شباهت های مثبت را منهای margin می کنیم

-0.11 + -0.21 + -0.97 + 0.5 + -1.0 + 0.18 + 0.08 + -0.31 + -0.03 = -1.638 = X

(|-1.638| + 1) * 3 = 7.94 = Y

1.638 / 7.94 = 0.26 = margin

Score = X – margin = 3.79

شاید در ذهنتان سوال پیش بیاید که محاسبه margin آیا تنها برای این است که score یکسان به وجود نیاید؟ باید در جواب بگوییم خیر. دلیل اصلی دیگر آن برای افزایش ضریب اطمینان شباهت دیتا است. بگذارید مبحث را با یک مثال جمعبندی کنیم:

فرض کنید دو تیم فوتبال زمین را به دو نیمه تقسیم کرده‌اند و خط وسط زمین نقش ابرصفحه (Hyperplane) را دارد که مرز بین دو کلاس محسوب می‌شود. توپ همان داده‌ی ورودی است که باید تشخیص دهیم به کدام نیمه تعلق دارد.

حالتی را در نظر بگیرید که چند داور) مانند سناریوی One-vs-One در (SVM هر کدام از زاویه‌ی خود تصمیم می‌گیرند توپ در نیمه‌ی کدام تیم قرار دارد. ممکن است تعداد آراء مساوی شود؛ مثلاً دو داور بگویند توپ در نیمه‌ی تیم A است و دو داور دیگر بگویند توپ در نیمه‌ی تیم .B در این حالت سیستم دچار تساوی (Tie) می‌شود.

برای رفع این مشکل از Margin-Based Tie Breaking استفاده می‌کنیم. به این معنا که داور اصلی به جای شمارش صرف آراء، به فاصله‌ی توپ از خط وسط (فاصله از ابرصفحه) توجه می‌کند. تیمی برنده اعلام می‌شود که توپ در نیمه‌ی آن فاصله‌ی بیشتری از خط وسط دارد؛ یعنی داده با حاشیه (Margin) بالاتری در آن کلاس قرار گرفته است.

به بیان رسمی‌تر: در شرایط تساوی، کلاس نهایی با توجه به بیشینه‌ی مقدار تصمیم (Decision Function Value) یا همان فاصله‌ی توپ از ابرصفحه انتخاب می‌شود. این رویکرد تضمین می‌کند که پیش‌بینی نهایی نه صرفاً بر اساس رأی‌گیری ساده، بلکه بر پایه‌ی اطمینان مدل (Margin) اتخاذ گردد.

همانطور که دیدید SVC به صورت پیش فرض در حالت OvR قرار دارد. اما آیا راهی دارد که بتوان از طریق همین کلاس به صورت OvR آنالیز کرد؟ به صورت مستقیم خیر اما با استفاده از یک کلاس دیگر به همراه این کلاس می توان این کار را انجام داد. کلاس OneVsOneClassifier در کتابخانه sklearn.multiclass می تواند این را برای ما فراهم سازد ما با قرار دادن کلاس SVC در این کلاس می توانیم به صورت OvO این کار را انجام دهیم.

با انجام این روش ما یک لیست ۴۵ تایی score خواهیم داشت که نشان دهنده مقایسه و برسی هر کلاس با باقی کلاس ها است و به صورت کلی کلاسی که نمره بالاتری داشته باشد می تواند به عنوان پیش بینی برنده انتخاب شود.

اما اگر بخواهیم کمی کار را راحت کنیم می توانیم می توانیم از کلاس کاربردی SGDClassifier که قبلا هم از آن استفاده کرده ایم استفاده کنیم زیرا این کلاس به صورت پیش فرض به روش OvR به برسی دیتا می پردازد. نکته: تا قبل از آن هم ما با دادن دو متغیر y_train_5 و x_train از این مدل خواسته بودیم تا کلاس ۵ را با باقی کلاس ها برسی کند اما الان ما با دادن y_train به جای y_train_5 به برسی تمامی کلاس ها خواهیم پرداخت.

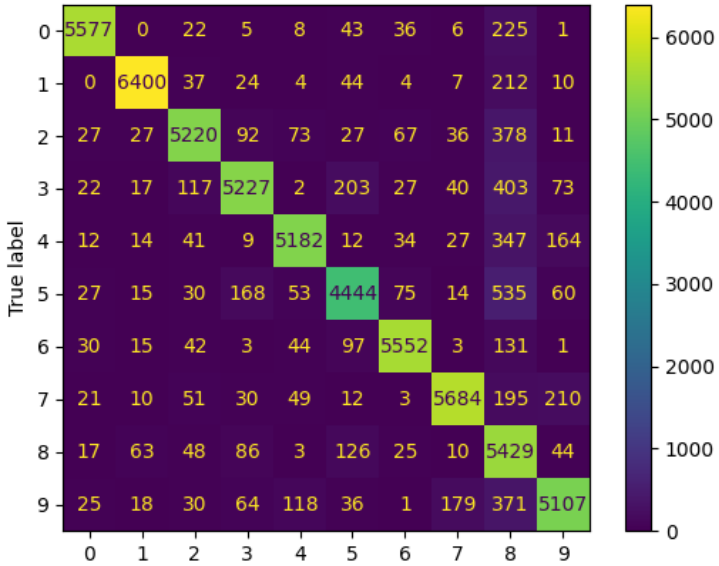
به یاد داریم که قبلا در استفاده از این کلاس می توانستیم با استفاده از تابع cross_val_score به برسی معیار عملکرد این مدل بپردازیم. در اینجا هم ما با دادن مدل و x_train و y_train به آن تابع می توانیم به معیار آن دستیابی کنیم. این معیار عملکرد در مدل ما برابر با ٪۸۵ است همانطور که می دانید این معیار عملکرد در هنگام دادن کلاس ۵ برای برسی با همه کلاس ها برابر ٪۹۵ بود اما در اینجا این عدد به چیزی حدود ٪۸۵ رسیده است. اما چرا؟ این یعنی ما در مدل خود پسرفتی داشته ایم؟ در جواب کوتاه بایستی بگوییم خیر. اما برای درک بهتر این موضوع بایستی به قبل تر برگردیم همانطور که قبلا هم ذکر شد دیتای ما skewness است یعنی نامتوازن بوده و تعداد دیتا ها با هم برابر نیستند این کار باعث می شود تا در برسی یک کلاس با باقی کلاس ها (علل خصوص کلاسی که دیتای کمتری دارد) نمره بالاتری کسب کند اما در برسی همه کلاس ها با یکدیگر (OvR) یک نتیجه کلی می تواند رضایت بخش باشد و آن عدد برابر با ٪۸۵ است.

اما اگر بیاییم به این درصد بسنده نکنیم. ما می توانیم با استفاده از کلاس StandardScaler از sklearn.preprosesing با نرمال های دیتا و بردن آنها به محدوده ای که مدل بتواند بهتر یاد یگیرد باعث افزایش درصد یادگیری یا همان accuracy شویم. با انجام این کار مشاهده می کنیم که این درصد به چیزی حدود ٪۹۱ می رسد که بسیار قابل توجه است.

با این حال می توانیم فعلا مدل SGDClassifier را جز مدل های بهتر خودمان در نظر بگیریم. با این فکر به سراغ تابع cross_val_predict برای اجرای مدل و یادگیری آن به صورت fold می رویم و نتیجه پیش بینی های آن را در داخل یک متغیر می ریزیم.

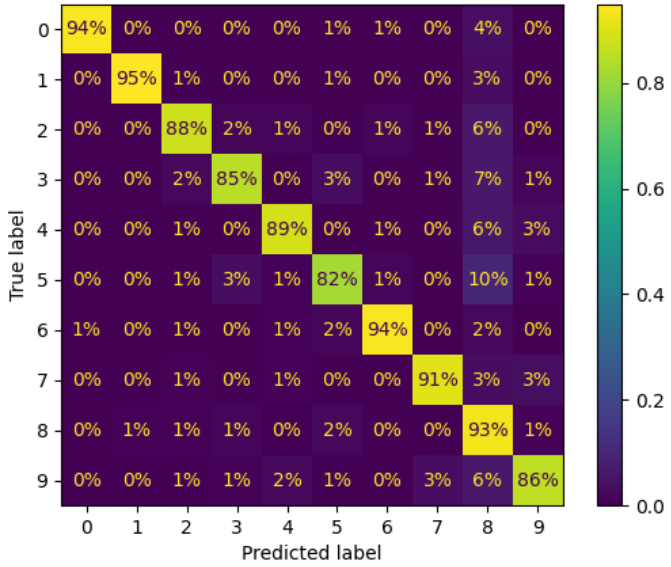
اما یک سوالی در اینجا مطرح می شود، آیا می توان به این درصد ها اعتماد کافی داشت؟ آن ۹ درصد باقی مانده بیشتر اشتباه بر روی کدام کلاس ها است؟

برای رسیدن به این پرسش ها ما بایستی از طریق ماتریس درهم ریختگی به بررسی این موضوع بپردازیم. اما بایستی به این مته توجه کنیم که تابع confusion_matrix تنها قادر به خروجی عددی ماتریسی است و ما برای درک بهتر بایستی به صورت گرافیکی به رسم ماتریس درهم ریختگی آن بپردازیم که sklearn.metrics با کلاس ConfusionMatrixDisplay این کار را برای ما راحت کرده است، لذا ما با دادن دو متغیر y_train و y_train_pred که حاصل خروجی تابع پیش بینی cross_val_predict است این مار را انجام می دهیم.



این شکل رسم شده از ماتریس درهم ریخته ما است. همانطور که می بینید ردیف ها مقدار های واقعی و مقادیر پیش بینی شده ستون ها هستند. با مشاهده تصویر می توانیم مشاهده کنیم که بیشتری مقدار درست پیش بینی شده مربوط به کلاس ۱ است که تعداد ۶۴۰۰ آنها واقعا ۱ پیش بینی شده است. همانطور که مشخص است بیشترین مقدار خطا مربوط به کلاس ۸ است به گونه ای که ۴۰۳ مورد از کلاس ۵ هشت پیش بینی شده است. با این حال برای برسی معیار عملکرد هر کدام چه کاری می شود انجام داد؟ برای این کار بایستی تعداد تمام دیتا های موجود در آن کلاس را جمع کنیم و سپس مقدار درست پیش بینی آن را تقسیم بر تعداد کل کنیم.

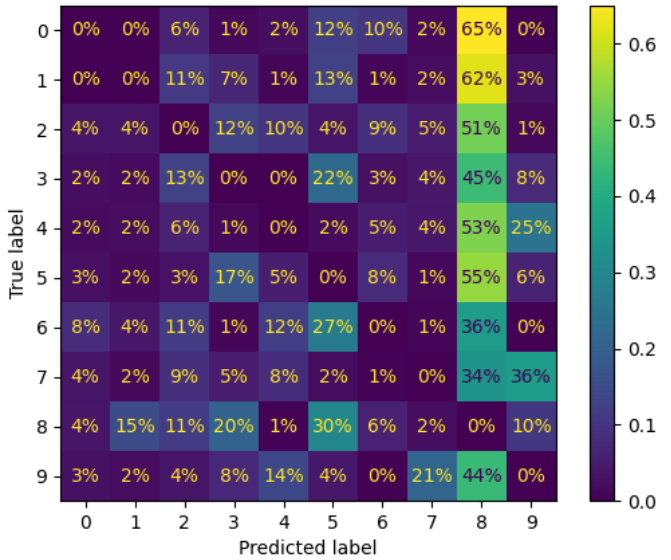
مثال: در کلاس ۵ ما ۵۴۲۱ عدد دیتا داریم و با این حال تعداد ۴۴۴۴ آن ها درست پیش بینی شده است، با یک تقسیم ساده متوجه می شویم که معیار عملکرد مدل چیزی حدود ٪۸۲ است.



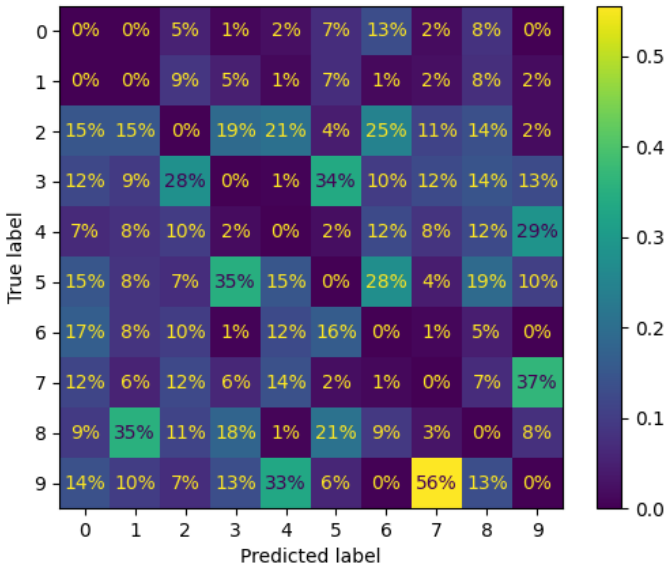
اما شایان به ذکر است که باز هم این کلاس به کمک ما آمده و خودش مقادیر معیار عملکرد را محاسبه می کند.

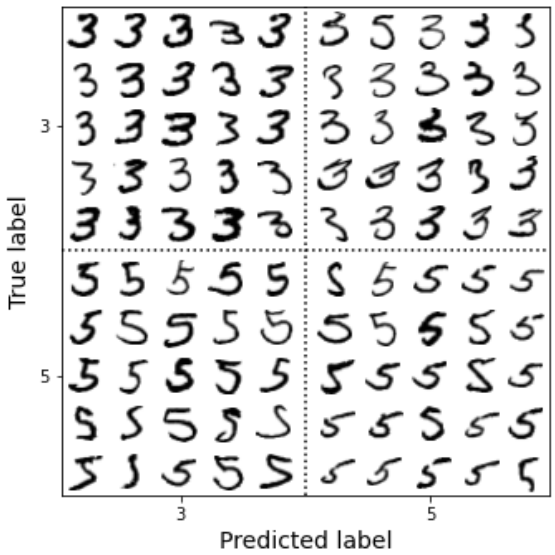
همانطور که می بینید اگر مقادیر درصد درست جمع و تقسیم بر تعدادشان کنیم به همان درصدی می رسیم که در cross_val_predict رسیدیم یعنی ٪۹۱.

در داخل متد form_prediction از کلاس ConfusionMatrixDisplay ما دارای دو متد normalize و values_format هستیم. حال این دو متد چه کاری انجام می دهند؟ در متد دومی ما مشخص می کنیم خروجی محاسبات به چه صورت باشد، درصد باشد، لگاریتم باشد و .... متد اول یعنی normalize ۳ مدل ورودی دارد اولی true است که اگر روی آن قرار دهیم مقداری که نشان می دهد را می توانیم در عکس سوم مشاهده کنیم. این مقادیر به ما می گویید که مدل چند درصد از دیتا های آن کلاس را در مرحله پیش بینی از دست داده است مثال: در کلاس ۰ مدل به اشتباه ٪۶۵ از آن کلاس را ۸ پیش بینی کرده و آنها را از دست داده است و همانطور که می بینید بیشتر خطا ها مربوط به تشخیص اشتباه کلاس ۸ است که باید به آن توجه لازم را داشت. ورودی دوم مقدار pred است که می توانیم در شکل چهارم (صفحه بعد) مشاهده کنیم. این مقدار به ما می گویید که چند درصد اشتباه کرده است. مثال: مدل ٪۵۶ از داده ها را کلاس ۷ پیش بینی کرده در صورتی که واقعا کلاس ۹ بوده اند. به طور کلی در حالت true مقدار recall و در حالت pred مقدار precision گزارش می شود.



نویسنده : عرفان جناب





همانطور که در شکل بالا سمت راست می بینید یک شکلی شبیه به ماتریس درهم ریخته است که اگر با چشم غیر مسلح نگاه کنیم در بسیاری از موارد حتی ما انسان ها ممکن است در بین تشخیص ۳ از ۵ دچار اشتباه شویم در قسمت سمت چپ پایین همانطور که مشاهده می کنید اعداد در اغلب موارد بسیار شبیه به ۳ هستند اما در واقعیت ۵ بوده اند. اما برای رفع این اشکال چه کاری می شود کرد؟ آیا راهی است که بتوان خطاهایی که مدل در پیش بینی عدد ۸ داشته است را کاهش داد؟ برای این کار ۳ روش در جلوی ما قرار دارد. روش اول این است که داده های آموزش جدیدی را اضافه کنیم که شبیه به ۸ هستند اما ۸ نیستند این کار باعث می شود تعداد زیادی عدد شبیه به ۸ با برچسب (Label) غیر ۸ داشته باشیم و مدل بتواند بهتر عدد ۸ را از دیگر کلاس ها بشناسد.

نکته: در این روش جمع آوری دیتا کاری آسان نیست و بسا می تواند در برخی پروژه ها غیر ممکن باشد.

در روش دوم ما می توانیم از روش الگوریتم نویسی استفاده کنیم، اما به چه شکل؟ ما می توانیم با دادن یک راهنما (الگوریتم خاص) به مدل به تشخیص آسان تر آن کلاس ها کمک کنیم. یکی از این الگوریتم ها می تواند تشخیص حلقه ها در اعداد باشد این حلقه ها در اعداد ۸ ۶ و ۵ کاربرد دارد زیرا به ترتیب این اعداد دارای ۲ ۱ و ۰ حلقه هستند. این کار کمک می کند تا مدل بتواند راحت تر به تشخیص کلاس ها بپردازد.

روش سوم روشی است که در پروژه ما می تواند بهتر عمل کند و آن این است که ما با استفاده از کتابخانه های OpenCv و scikit-image و pillow یک مرحله preproces انجام داده و سپس به مدل برای آموزش بدهیم. این کتابخانه ها باعث می شوند تا حلقه هایی که در روش ۲ وجود دارند پر رنگ تر و با وضوح بیشتری ظاهر شوند تا مدل بتواند فرق آن ها را راحت تر تشخیص دهد.

در ادامه به سراغ کلاس های چند برچسبی میرویم. اما این کلاس های چندبرچسبی اصلا چی هست؟ آیا ما به دیتایمان چیزی اضافه یا کم میکنیم؟

در جواب باید بگویم خیر. بلکه آموزش با استفاده از کلاس چند برچسبی به این معنی است که مدل با گرفتن چند المان بتواند دیتاها را براساس آنها تفکیک و جدا کرده و یاد بگیرد. حال این چطور ممکن است؟ برای شروع بیایید یکی از برچسب هایی که به دیتاهایمان می زنیم این باشد که دیتاهای بزرگ تر مساوی ۷ را در مجموعه y_train جدا کنیم. این کار باعث می شود مدل فرق بین دیتاهای کمتر از ۷ و بیشتر از ۷ را بشناسد و یاد بگیرد. برچسب دیگری که می توان زد این است که از میان دیتای مجموعه y_train دیتاهای فرد را بیرون کشیده و مشخص کنیم، این کار باعث می شود مدل علاوه بر اینکه فرق دیتاهای فرد را از زوج تشخیص دهد در تشخیص اعداد بزرگ از ۷ نیز رابطه ای مستقیم پیدا کند. چه رابطه ای؟ در اعداد بزرگ تر مساوی از ۷ یعنی ۷ ۸ و ۹ ما دو عدد فرد و یک عدد زوج داریم، همین عامل باعث یادگیری یک رابطه در بین دو کلاس شده و باعث افزایش f1_score می شود.

برچسب آخری که میتوان زد این است که ما از بین دیتاهای y_train دیتایی که جز دسته اعداد اول هستند (۲ و ۳ و ۵ و۷) را جدا و برچسب بزنیم.

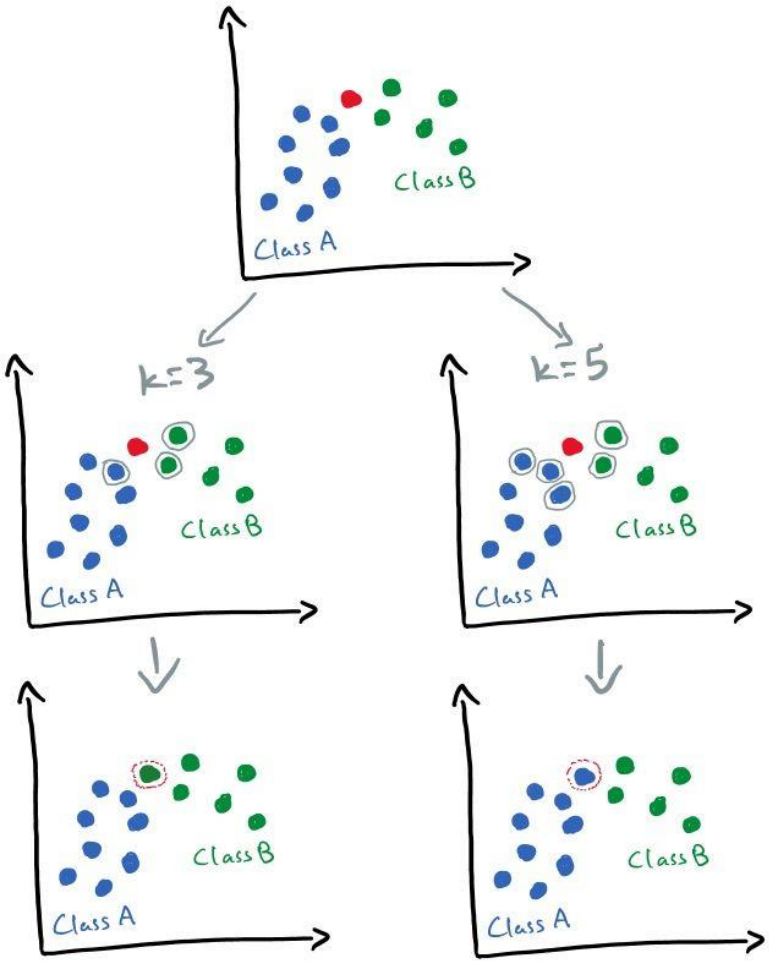
(نکته: تنها این ۳ برچسب نیستند بلکه ما میتوانیم هرگونه برچسبی که به یادگیری بهتر مدل منجر شود و بتواند روابط را بهتر درک کند میتوانیم ایجاد کنیم.)

تمام این کارها را ما با استفاده از numpy و pandas میتوانیم انجام دهیم و در پایان با استفاده از متد _c در کتابخانه numpy میتوانیم یک لیست ۳ ستونه ایجاد کنیم که شامل True و Flase ها است و در نهایت دیتاهای اصلی به همراه آن را به مدلی به اسم KneighborsClassifier کتابخانه sklearn.neighbors بدهیم.

این کلاس که معروف به نزدیک ترین همسایه هم است به این شکل عمل میکند:

همانطور که در شکل صفحه بعد میبینید این مدل با استفاده از متد ورودی به اسم n_neighbors که ما مشخص میکنیم دیتاهای آموزش را بر روی یک صفحه دو بعدی نموداری قرار داده (همه کلاس ها) و سپس با استفاده از روش نزدیک ترین همسایگی دیتای جدیدی که باید پیش بینی کند انجام میدهد. در

این روش دیتای جدید را براساس نزدیک ترین فاصله (score) به دیتای اطرافش حدس و پیش بینی میکند. در این عکس همانطور که میبینید هرچه مقدار k بیشتر باشد دقت مدل افزایش میابد زیرا میتواند کلاس های مختلف بیشتری را تحت برسی قرار دهد.



نکته اول قابل توجه این است که باید مواظب باشیم اشتباهی k را روی عدد زوجی قرار ندهیم زیرا امکان برابر شدن فاصله ها و یکسان شدن score وجود دارد.

نکته دوم قابل توجه این است که مدلی که الان ذکر کردیم نکاتی که قبل تر گفته ایم را (یادگیری رابطه بین فرد بودن اعداد بزرگ تر از ۷ و دسته اعداد فرد) را نمیتواند اجرا کند و یاد بگیرد