دیتاست شماره ۱

نوت‌بوک این دیتاست با نام A02_Task1.ipynb ارسال می‌گردد.

بخش ۱ و ۲

برای سادگی کار ۴ کلاس مسئله را همانطور که گفته شده از اینجا به بعد به ۲ کلاس کاهش دادم، چون اگر بررسی‌های مربوط به AUC را روی ۲ کلاس و بقیه را روی ۴ کلاس انجام دهیم، بررسی عادلانه‌ای نخواهد بود. اما تمامی محاسبات و آموزش‌ها را می‌توان به حالت ۴ کلاسه نیز تعمیم داد. نکته دیگری که وجود دارد این است که داده تست در دست، فاقد ویژگی **price_range** است پس نمی‌توان از آن برای بررسی خطای تست استفاده کرد. در بسیاری از موارد داده‌های آموزش را با **train_test_split** به دو دسته آموزش و تست تقسیم می‌کنم و در برخی موارد نیز از متد wrapper استفاده می‌شود، این متد یک مدل، ویژگی‌ها و متغیر هدف را دریافت کرده و میانگین خطای Cross-Validation را روی آن گزارش می‌کند.

در این بخش روش انتخاب ویژگی Forward Selection را پیاده‌سازی می‌کنم. فرض کنید p ویژگی داریم، این روش به این صورت است که ابتدا با یک مدل بدون ویژگی یا M_0 آغاز می‌کنیم که فقط شامل عرض از مبدا (intercept) است و در واقع انگار همیشه میانگین مقدار **price_range** را پیش‌بینی می‌کند. معیار AUC برای دسته‌بندی که قابلیت ایجاد تمایز در دو کلاس را ندارد برابر ۰.۵ است (اگر این مقدار کمتر از ۰.۵ باشد دسته‌بندی غلطی انجام می‌دهد). در مرحله اول از Forward Selection میخواهیم از M_0 به M_1 گذر کنیم. ابتدا در هر مرحله یک ویژگی را به مدل اضافه می‌کنیم. ویژگی‌ای که بیشترین مقدار AUC را نتیجه می‌دهد ثابت نگه می‌داریم. در مرحله بعد p-1 ویژگی بعدی را اضافه کرده و AUC را می‌سنجیم و دومین ویژگی را فیکس می‌کنیم و به همین ترتیب پیش می‌رویم. ممکن است در مرحله‌ای با اضافه کردن همه ویژگی‌های باقیمانده، نتیجه یا همان معیار AUC بهتر نشود. همانطور که می‌بینیم ۲۰ ویژگی داریم و این روش ۷ ویژگی را نتیجه داده است، یعنی از مرحله ۸ به بعد اضافه کردن ویژگی‌ها به مدل عملکرد آن را بهتر نمی‌کند. لازم به ذکر است که این روش حریصانه بوده و تمام حالات برای انتخاب ویژگی‌ها را در نظر نمی‌گیرد، در نتیجه لزوماً بهترین زیر مجموعه از ویژگی‌ها را به دست نمی‌دهد. پیاده‌سازی این الگوریتم در متد **forward_selection** انجام شده است.

متد **fit_calculate** داده‌های متناظر با تعدادی ویژگی متغیر تصمیم (Target) متناظرشان دریافت کرده و آن‌ها را به دو مجموعه Train-Test تقسیم کرده، یک مدل رگرسیون لجستیک روی بخش Train آموزش داده و معیار AUC را برای بخش Test به عنوان خروجی برمی‌گرداند. **max_iter** برابر ۲۰۰۰ در نظر گرفته می‌شود زیرا در بعضی موارد مشاهده می‌شود که الگوریتم نمی‌تواند به همگرایی برسد.

حال با ویژگی‌های انتخاب شده با این روش، یک مدل رگرسیون لجستیک آموزش می‌دهیم و نتایج آن را روی داده‌های تست می‌سنجم. نتیجه به قرار زیر است:

	precision	recall	f1-score	support
0	0.9907	0.9725	0.9815	109
1	0.9677	0.9890	0.9783	91
accuracy			0.9800	200
macro avg	0.9792	0.9807	0.9799	200
weighted avg	0.9802	0.9800	0.9800	200

بخش ۲ و ۳

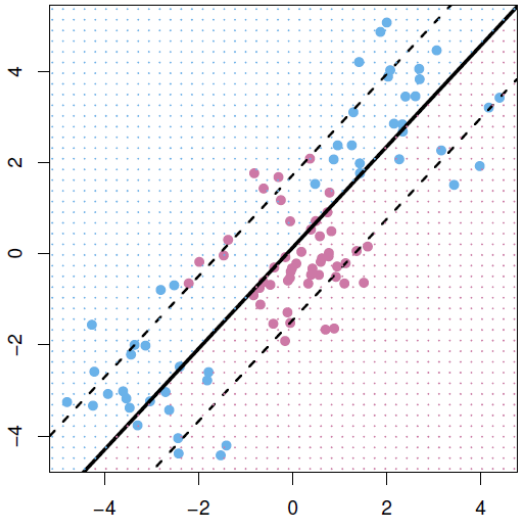
روی داده‌ها روش PCA را اعمال کرده و به تعداد ویژگی‌های انتخاب شده در مرحله Forward Selection از مولفه‌های اصلی برای آموزش یک مدل رگرسیون لجستیک استفاده می‌کنم. نتایج را می‌بینیم:

نکته‌ای که در اینجا وجود دارد این است که اولا قبل از اعمال PCA باید روی داده‌ها نرمال‌سازی صورت گیرد چون این متد به مقیاس داده‌ها حساس است. مساله دوم این است که برای این که نشتی اطلاعات (Data Leakage) از داده‌های تست به مدل وجود نداشته باشد، باید PCA را فقط روی داده‌های آموزش fit کنیم. با توجه به تمامی معیارها، با افت عملکرد رو به رو شدیم. دقت کنید که تنها چیزی که PCA تضمین می‌کند این است که داده‌ها را به فضایی جدید می‌برد به طوری که در این فضا ویژگی‌ها بر اساس واریانس به صورت نزولی مرتب شده‌اند، واریانس به مثابه میزان اطلاعات هر ویژگی است، مولفه‌ای که بیشترین واریانس را دارد بیشترین اطلاعات را در بر می‌گیرد. همچنین برای محاسبه PCA باید شرایطی برقرار باشند، مانند این که ویژگی‌ها از هم مستقل باشند و همبستگی نداشته باشند، که از این مورد در اینجا اطمینان حاصل نکردیم. همچنین متغیر Y را در PCA لحاظ نکردیم و نمی‌توانیم لحاظ کنیم چون نشت اطلاعات صورت می‌گیرد.

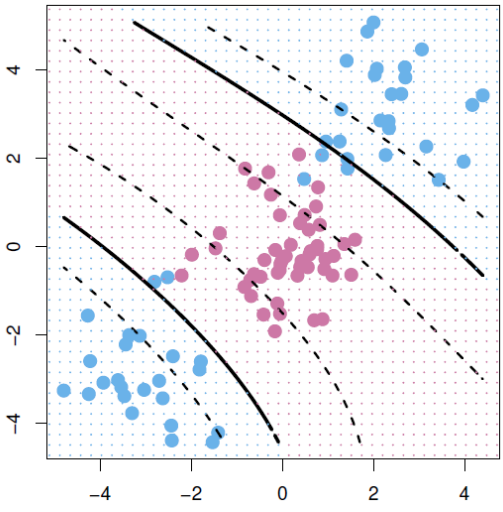
	precision	recall	f1-score	support
0	0.5701	0.6354	0.6010	96
1	0.6237	0.5577	0.5888	104
accuracy			0.5950	200
macro avg	0.5969	0.5966	0.5949	200
weighted avg	0.5979	0.5950	0.5947	200

بخش ۵

می‌دانیم الگوریتم SVM در مسائلی که داده‌ها به طور خطی جدایی‌پذیرند، سعی در یافتن یک ابرصفحه دارد با این ویژگی که بتواند داده‌ها را به دو بخش تقسیم کند. در برخی از مسائل داده‌ها به طور خطی در فضا قابل جدا کردن نیستند، مثلا در شکل زیر با یک خط نمیتوانیم بین دو کلاس تمایز ایجاد کنیم:



یک راه حل استفاده از روش Feature Expansion است در آن با به توان رساندن یا ضرب کردن ویژگی‌های موجود در هم، ویژگی‌های جدیدی با ابعاد بالاتر ایجاد می‌کنیم. فضای ویژگی جدید قابلیت مدل کردن مسائل غیرخطی را خواهد داشت. مثلا با این روش می‌توانیم مرزهای مسئله قبلی را به دست بیاوریم:



یکی از بدی‌های این روش این است که با رفتن به ابعاد بالاتر با محاسبات پیچیده‌تری مربوط به ویژگی‌های چندجمله‌ای جدید ایجاد شده روبرو می‌شویم. اینجا ایده استفاده از کرنل‌ها مطرح می‌شود. در واقع کرنل‌ها

یک میانبر هستند که به عنوان پارامتری جدید به SVM اضافه شده و به ما امکان مدل کردن ابعاد بالاتر بدون درگیر شدن با محاسبات پیچیده و ایجاد ویژگی جدید را می‌دهند. در واقع برخی کرنل‌ها حتی قابلیت مدل کردن فضایی با ابعاد بی‌نهایت را هم دارا می‌باشند. ما نیاز به انجام هیچ‌گونه محاسبات پیچیده‌ای نداریم، تنها باید داده را فراهم کرده و کرنل مناسبی را انتخاب کنیم. برخی از کرنل‌های معروف:

کرنل خطی که به صورت ضرب داخلی تعریف می‌شود:

$$K(x_i, x_{i'}) = < x_i, x_{i'} >$$

کرنل سیگموید

$$K(x_i, x_{i'}) = \tanh\left(\alpha x_i^T x_{i'}\right)$$

کرنل شعاعی

$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right)$$

کرنل Radial Basis Function که به طور پیش‌فرض در ماژول SVM بسته sklearn استفاده می‌شود:

$$K(x_i, x_{i'}) = \exp\left(-\frac{-\gamma ||x_i - x_{i'}||^2}{2\sigma^2}\right)$$

این کرنل به دلیل شباهت با توزیع گاوسی جزو پراستفاده‌ترین و تعمیم‌یافته‌ترین کرنل‌هاست و می‌تواند فضایی با بعد بی‌نهایت را مدل کند.

به طور کلی برای استفاده از کرنل‌ها نمی‌توان نظر کلی داد و به طور مثال نمی‌توان گفت که همیشه از کرنل RBF باید استفاده کنیم، چون این کرنل قطعا از کرنل خطی پیچیده‌تر است و باید وقتی از این پیچیدگی استفاده شود که واقعا به آن نیاز داریم. بنظرم بهتر است در مسائلی که از ابعاد و رفتار واقعی داده‌ها خبر نداریم، ابتدا از کرنل خطی شروع کنیم و سپس در صورت نیاز از کرنل‌های پیچیده‌تر استفاده کنیم.

بخش ۶

در این بخش باید روی داده‌ها روش SVM را اجرا کنم. در ابتدا از svc بسته sklearn با کرنل خطی و تنظیمات پیش‌فرض استفاده میکنم. دقت کنید که قبل از اعمال svm بهتر است نرمالسازی داده‌ها صورت گیرد. همچنین در این مرحله می‌توانستیم تنها از ۵ ویژگی انتخاب شده از مرحله Forward Selection استفاده کنیم، اما من مدل‌ها را برای این بخش بخش‌های بعدی روی کل ویژگی‌ها آموزش دادم. نتیجه SVM با کرنل خطی (سایر تنظیمات پیش‌فرض در نظر گرفته شده‌اند) به شرح زیر است:

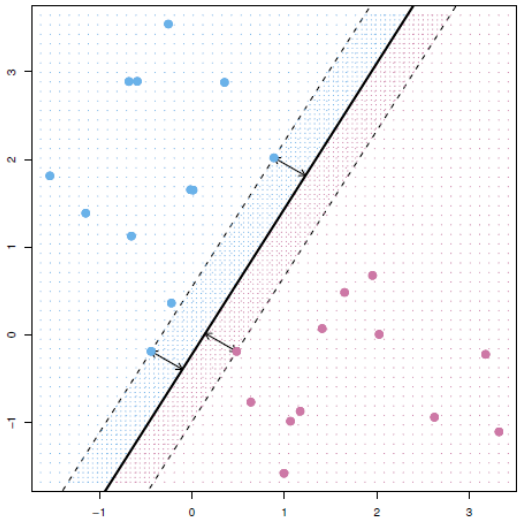
	precision	recall	f1-score	support
0	0.9694	0.9896	0.9794	96
1	0.9902	0.9712	0.9806	104
accuracy			0.9800	200
macro avg	0.9798	0.9804	0.9800	200
weighted avg	0.9802	0.9800	0.9800	200

بخش ۷

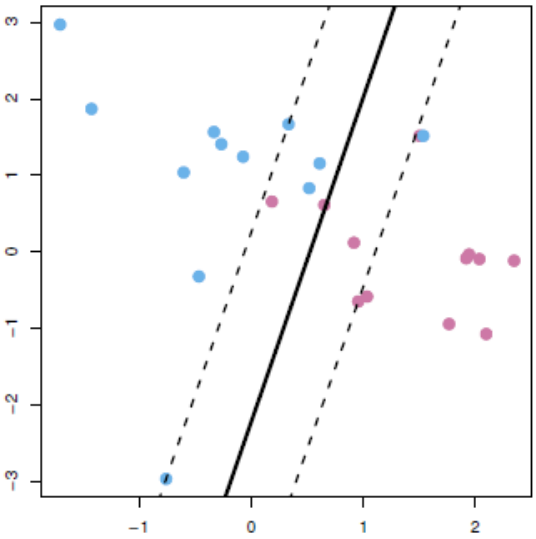
در بخش ۵ چند کرنل معروف svm را معرفی کردم. در این بخش پنج مدل متفاوت با این کرنل‌ها ساخته و عملکرد هر یک را بررسی می‌کنم. دقت کنیم که چون روی داده‌ها نرمالسازی انجام شده، واریانس آنها برابر ۱ می‌شود و در نتیجه طبق مستندات، دو گزینه scale و auto برای گاما یا ضرایب کرنل معادل می‌شوند. در اینجا کرنل‌هایی چون poly، sigmoid و rbf را بررسی می‌کنم. همچنین برای کرنل poly مقادیر ۲، ۳ و ۵ برای درجه چند جمله‌ای در نظر می‌گیرم. متد evaluate_svm_setting یک تنظیمات برای SVM و همچنین داده X و y را دریافت کرده، یک مدل svm با تنظیمات داده شده را با wrapper و Cross-Validation آموزش داده و دقت میانگین را گزارش می‌کند. به نظر می‌رسد یک مدل با کرنل خطی و پس از آن یک مدل با کرنل‌های rbf یا sigmoid انتخاب‌های مناسبی باشند. اگر مدل‌ها را به جای کل ویژگی‌ها روی ویژگی‌های به‌دست آمده از بخش ۱ آموزش دهیم، باز هم بهترین مدل، مدلی با کرنل خطی خواهد بود. از آنجایی که با تسک دسته‌بندی دوکلاسه متوازن (تعداد ۰ و ۱ تقریبا با هم برابر است) روبرو هستیم، accuracy معیار خوب و معتبری برای سنجش و مقایسه مدل‌های متفاوت است.

بخش ۸

هنگامی که روش svm را اجرا می‌کنیم، همانطور که گفته شد هدف یافتن یک ابرصفحه جداکننده است. کیفیت این ابرصفحه با میزان margin سنجیده می‌شود. یعنی هر چه حاشیه بیشتری بتوانیم ایجاد کنیم، با قطعیت بیشتری می‌توانیم داده‌ها را دسته‌بندی کنیم:



در این شرایط هدف ما این است که هیچ داده‌ای در حاشیه قرار نگیرد. در این حالت که سخت‌گیرانه‌تر عمل کرده‌ایم اصطلاحاً hard margin داریم. شرایطی وجود دارند که امکان شکل دادن به حاشیه‌ای بدون اینکه داده‌ای درون آن بیفتد وجود ندارد، در این شرایط ما یک میزان تحمل (C) برای خطا در نظر می‌گیریم و به این میزان اجازه وجود داده در حاشیه را می‌دهیم. در این شرایط یک soft margin داریم:



کران بالایی که برای خطا در نظر می‌گیریم (میزان تحمل یا budget) را با C نشان می‌دهیم و در واقع یک هایپرپارامتر شبکه است که موازنه بین بایاس و واریانس را برقرار می‌کند. هر چه میزان C کوچکتر باشد یعنی حاشیه ما تنگ‌تر بوده و تحمل کمتری برای خطا دارد (بایاس کم واریانس زیاد) و هر چه این مقدار بزرگتر باشد یعنی حاشیه پهن‌تر و تحمل بیشتری برای خطا داریم (بایاس زیاد و واریانس کم). حال در این بخش مقادیر مختلف برای C را بررسی می‌کنم. از مراحل قبل به یاد داریم کرنل خطی بهترین عملکرد را داشت، پس آن را ثابت نگه می‌داریم. به نظر می‌رسد هر چه مقادیر C به ۰ نزدیک‌تر باشد و سخت‌گیرانه‌تر درباره خطاها برخورد کنیم، نتیجه بدتری روی تست خواهیم داشت (که با توجه افزایش واریانس طبیعی است). بهترین مقادیر را وقتی خواهیم داشت که C=100 یا C=1000 باشد، یعنی هر چه حاشیه softتر باشد.

حال در آخر یک GridSearchCV با تمام پارامترها و تنظیمات بررسی شده تعریف می‌کنم. این متد تعدادی تنظیمات برای یک مدل دریافت کرده، مدل را به ازای تمام ترکیبات این پارامترها آموزش داده و بهترین پارامترها را می‌یابد. همچنین به طور خودکار داده‌ها را به ۵ قسمت تقسیم کرده و K-Fold Cross-Validation (به طور پیش فرض K=5) انجام می‌دهد پس در نتیجه واریانس خطا کاهش یافته و خطایی که برای هر مدل و مجموعه پارامتر به دست می‌آورد قابل اتکاست. در انتها می‌بینیم که در بین ۳۲۰ جایگشت مختلف از پارامترها، بهترین مدل دارای کرنل خطی و C=1000 است. نکته قابل توجه در اینجا این است که اگر کرنل poly نباشد مقدار degree برای آن در نظر گرفته نمی‌شود، حتی اگر به مدل ورودی داده شود.

بخش ۹ و ۱۰

برای این بخش یک متد train_svm نوشته شده که با دریافت داده‌های X و y آنها را نرمالسازی کرده و روی یک مدل svm با کرنل خطی و مقدار C=1000 (بدست آمده از GridSearch) با wrapper آموزش داده و نتایج را چاپ می‌کند. اول از همه یک مدل baseline را بدون انجام هیچگونه مهندسی ویژگی آموزش می‌دهم. دقت این مدل برابر ۹۸.۸۵٪ است. دقت داریم که این دقت بر اساس Cross-Validation بدست آمده و تقریب خوبی روی داده تست را تخمین می‌زند. هر مهندسی ویژگی که کمتر از این دقت باشد، صلاح نیست که از آن استفاده شود.

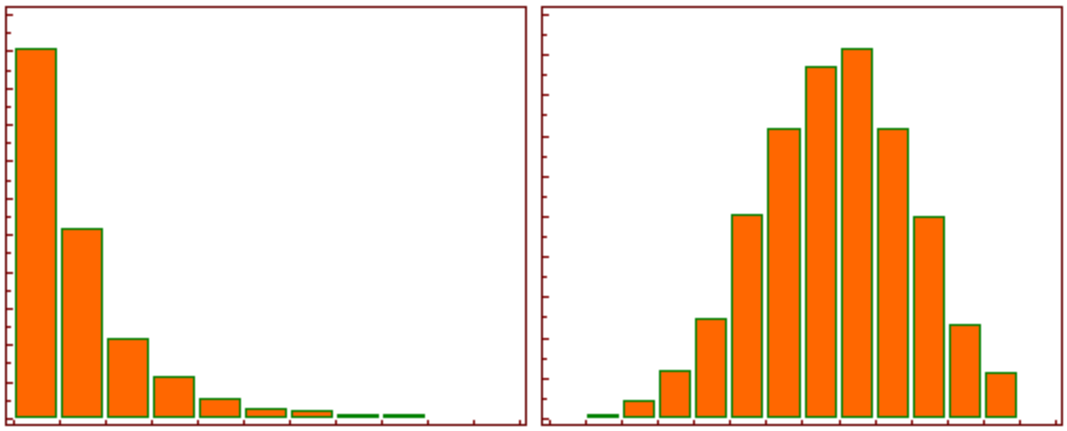
در قسمت آ خواسته شده که روی ویژگی battery_power از روش binning استفاده شود. چهار اندازه مختلف در نظر گرفته شده که فواصل در یکی از آنها برابر نیست. دوتا از نمونه‌ها حاوی ۵ bin (یکی با فواصل نابرابر) و دوتای دیگر با ۳ و ۱۰ bin هستند. پس از بررسی به این نتیجه می‌رسم که اگر اندازه bin برابر ۱۰ و فواصل برابر باشد دقت از حالت‌های دیگر بیشتر خواهد بود.

در قسمت ب خواسته شده تا One-Hot Encoding انجام دهیم. با بررسی اولیه ویژگی‌ها به این نتیجه می‌رسم که تمامی ویژگی‌ها عددی هستند. پس نیازی به انجام اینکار نیست. اگر یک ویژگی عددی و int باشد (LabelEncoding) ولی ترتیب ارزشی خاصی بین اعداد برقرار نباشد باید One-Hot انجام دهیم. به طور مثال اگر ویژگی‌ای تحت عنوان سیستم عامل داشته باشیم و برای ios مقدار ۱ و برای android مقدار ۲ و برای سایر سیستم‌عامل‌ها عدد ۳ در نظر گرفته باشیم. با این که مقادیر غیرعددی نیستند اما اینطور القا می‌کنند که گویی اندروید چون با عدد ۲ که بزرگتر است مشخص شده پس بر ios برتری دارد که لزوما اینطور نیست. در مورد سایر ویژگی‌های int و عددی مانند fc که نشان دهنده مگاپیکسل دوربین جلو است، نیازی به One-Hot نداریم چون ترتیب مهم است و هر چه این مقدار مگاپیکسل بیشتر باشد بهتر است.

در ضمن ویژگی‌های بولین (boolean) مثلا **dual_sim** که در صورت دوسیم‌کارت بودن گوشی ۱ و در غیر اینصورت مقدار ۰ می‌گیرند، در واقع One-Hot شده هستند (انگار get_dummies را با drop_first=True صدا زده‌ایم که به جای دو ویژگی **is_dual** و **is_not_dual** فقط یکی را نگه دارد چون صفر بودن یکی به معنی یک بودن دیگری است). روی چنین ویژگی‌هایی اگر دوباره One-Hot اعمال کنیم کار بیهوده‌ای کردیم چون برخی ویژگی‌ها زیادتر کرده‌ایم بدون اینکه اطلاعات اضافی به داده‌ها اضافه کرده باشیم.

بخش ج

در بحث مهندسی ویژگی، گاهی با مقادیر عددی مثبتی روبرو هستیم که یا از توزیع نرمال پیروی نمی‌کنند و یا اگر پیروی می‌کنند دچار چولگی هستند. مانند تصویر زیر:



در این شرایط تبدیلات مختلفی می‌تواند مورد استفاده قرار گیرد که یکی از آن‌ها تبدیلات نمایی است. در تصویر بالا تبدیل نمایی اعمال شده است. این تبدیل درواقع مقادیر بسیار بزرگ را در وسط گردآورده و مقادیر کوچک را در گوشه‌ها قرار می‌دهد. علاوه بر تبدیلات نمایی، از تبدیلاتی مانند square root transform نیز استفاده می‌شود. یک تبدیل معروف دیگر Box-Cox Transform نام دارد. این تبدیل از

روش‌های قبلی مانند تبدیل نمایی کمی پیشرفته تر بوده و قابلیت این را دارد که یک ویژگی را به توزیع نرمال نزدیک کند. فرمول آن به قرار زیر است:

$$w_t = \begin{cases} \log(y_t) & \text{if } \lambda = 0; \\ (y_t^\lambda - 1)/\lambda & \text{otherwise.} \end{cases}$$

پس از انجام تبدیل نمایی و Box-Cox متوجه می‌شویم که دقت دومی بیشتر بوده (به ترتیب ۹۶.۷۵٪ و ۹۸.۳۰٪) و دقت هر دو از حالت baseline کمتر است.

در قسمت د ویژگی مساحت و حجم را اضافه کردم. منظور از مساحت صرفا مساحت صفحه نمایش در نظر گرفته شده است. سه حالت در اینجا پیش می‌آیند، مدلی با ویژگی مساحت، مدلی با ویژگی حجم و مدلی با هر دو ویژگی. (دقت‌ها به ترتیب ۹۷.۵۰٪، ۹۸.۷۰٪ و ۹۷.۳۵٪) به نظر وجود حجم به تنهایی بهتر از دو حالت دیگر است. لازم به ذکر است در این مرحله پس از محاسبه یک ویژگی جدید، ویژگی‌هایی که در محاسبه آن استفاده شدند را حذف کردم.

baseline بدتر است. در مجموع تمام مهندسی‌های ویژگی انجام شده در این بخش دقت کمتری نسبت به مدل baseline داشتند.

بخش ۱۱

ID3 که از قدیمی‌ترین الگوریتم‌های ساخت درخت تصمیم است، در یک رویکرد حریصانه بالا به پایین، در هر مرحله سعی می‌کند یک ویژگی را انتخاب کند به طوری که با شکست داده‌ها بر اساس آن ویژگی، بیشترین میزان Information Gain (کمترین مقدار Entropy) حاصل شود. در واقع این الگوریتم ویژگی‌هایی که اطلاعات زیادتری (Gain بیشتری) را در بر می‌گیرند در سطوح بالاتر درخت قرار می‌دهد. یکی از محدودیت‌های این الگوریتم این است که ذاتاً برای ویژگی‌های پیوسته معرفی نشده است.

الگوریتم C4.5 چندسال پس از ID3 توسط سازنده همان معرفی شد تا برخی از محدودیت‌های ID3 را کاهش دهد. اول این که این الگوریتم قابلیت درک الگوریتم‌های گسسته و پیوسته را دارد و فقط به الگوریتم‌های گسسته محدود نمی‌شود. همچنین این الگوریتم با داده‌های ناموجود (Missing Values) نیز مشکلی ندارد و با وجود آن‌ها نیز می‌تواند درخت تصمیم بسازد. همچنین ایده دیگری که باعث بهبود عملکرد می‌شود، عملیات هرس کردن (Pruning که در بخش ۱۴ درباره آن صحبت می‌کنم) برای کاهش واریانس و افزایش قابلیت مدل در تعمیم (Generalization) است. تفاوت پایانی این الگوریتم با ID3 این است که می‌تواند وزن‌های متفاوتی برای ویژگی‌های مختلف قائل شود.

معیار بعدی یعنی CART که پس از C4.5 معرفی شده است، بسیار شبیه آن است با این تفاوت که قابلیت پشتیبانی از متغیر هدف (Target Variable) کمی یا رگرسیون را نیز دارد و این که CART جداسازی را با روشی عددی و بازگشتی بدون محاسبه Rule Setها (برخلاف C4.5) انجام می‌دهد.

بخش ۱۲ و ۱۳

در این بخش چند درخت تصمیم با عمق‌های مختلف می‌سازم. در ابتدا یک درخت تصمیم بدون تغییر پارامترها (با پارامترهای پیش‌فرض) می‌سازم که محدودیتی برای عمق ندارد و همچنین از معیار Gini استفاده می‌کند. سپس به ترتیب درخت‌هایی با حداکثر عمق ۳، ۵، ۱۰، ۵۰ و ۲۰۰ می‌سازم. اگر عمق خیلی کم باشد مثلا ۳، به نظر Underfit رخ داده است. همچنین در مدل پیش‌فرض چون محدودیت عمقی وجود ندارد، دقت از حالات دیگر (بجز عمق ۳) کمتر است. با افزایش عمق تا ۱۰ و ۵۰ دقت افزایش می‌یابد اما اگر خیلی عمق افزایش یابد مثلا ۲۰۰، دقت کاهش می‌یابد. در گزارش دقت برای کاهش واریانس، از روش K-Fold Cross-Validation با K=5 استفاده شده است. همچنین تاثیر تعداد نمونه‌های موجود در هر گره نیز بررسی شده است. در پایان یک **GridSearchCV** با مجموعه‌ای از پارامترها اجرا شده و بهترین پارامترهای به دست آمده عبارتند از: max_depth=50 و min_samples_leaf=4

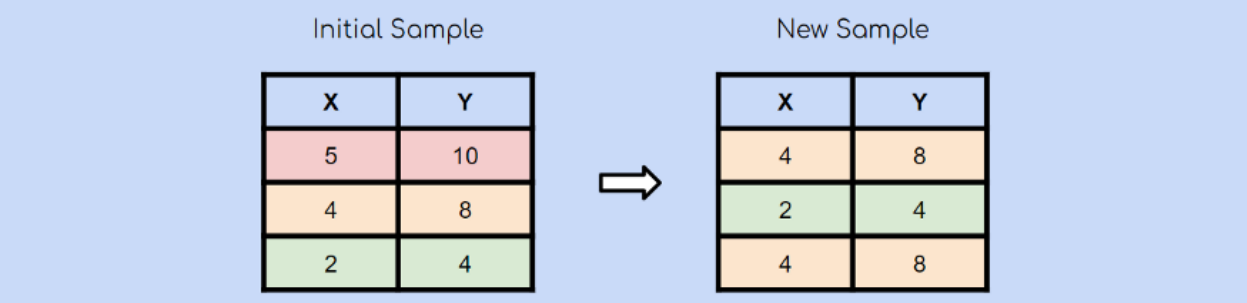
بخش ۱۴

یکی از پرسش‌هایی که در ساخت درخت‌های تصمیم مطرح می‌شود این است که اندازه یا عمق بهینه یک درخت تصمیم باید چقدر باشد؟ اگر عمق خیلی کم باشد بایاس افزایش پیدا می‌کند و دچار Overfitting می‌شویم و اگر درخت خیلی عمیق باشد نیز واریانس افزایش یافته و دچار Overfitting می‌شویم و مدل قدرت تعمیم به نمونه‌های جدید را از دست می‌دهد. در این مواقع از هرس کردن درخت بهره می‌جوییم. هرس پسین (Post-Pruning) یعنی وقتی که که درخت را تا جایی که می‌شود رشد دهیم و سپس از هرس کردن استفاده کنیم تا عمق و برگ‌ها را کاهش دهیم. هرس پیشین وقتی اتفاق می‌افتد که مدل درخت تصمیم را در رشد دادن برگها و افزایش عمق محدود کنیم. یکی از معروف‌ترین روش‌های هرس پسین که در کتاب An Intro to Statistical Learning نیز معرفی شده، روش Cost-complexity pruning است. در این روش یک مقدار خطا برای درخت (همان RSS) را محاسبه کرده و با مفهومی به نام جریمه پیچیدگی (Tree Complexity Penalty) جمع می‌کنیم که تابعی از تعداد برگ‌ها در یک زیردرخت است. در اینجا یک هایپرپارامتر α در این جریمه ضرب می‌شود که با استفاده از Cross-Validation یک مقدار مناسب برای آن می‌یابیم. این پارامتر مشخص می‌کند که چه مقدار از درخت باید هرس شود و در واقع اگر برابر ۰ باشد هیچ هرسی انجام نمی‌شود:

$$TreeScore = RSS + aT$$

بخش ۱۵

Bootstrapping یک روش نمونه‌گیری مجدد (Resampling) است که در آن از نمونه‌گیری تصادفی با جایگذاری استفاده می‌شود. چون نمونه‌گیری با جایگذاری انجام می‌شود، امکان این وجود دارد که یک مشاهده بیش از یک با انتخاب شود (در برخی موارد ممکن است اصلاً نمونه‌ای انتخاب نشود).



یکی از اهمیت‌های بوت‌استرپ این است که در بسیاری از الگوریتم‌های یادگیری ماشین که مبتنی بر کیسه‌گذاری (Bagging) هستند دیده می‌شود. مثلا AdaBoost و XGBoost

استفاده دیگر این روش در تخمین پارامترهای یک جامعه آماری مانند میانگین و خطای استاندارد است. در بسیاری از مواقع ما نمونه‌ای که داریم بزرگ نیست و در تخمین این موارد به مشکل می‌خوریم که می‌توانیم از بوت‌استرپ استفاده کنیم.

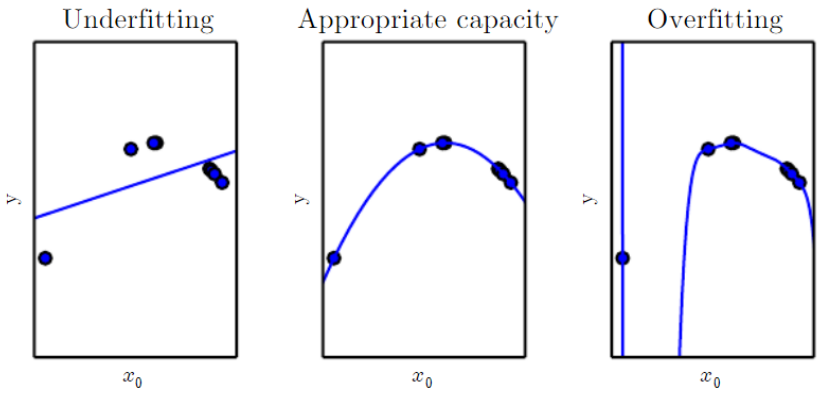
Cross-Validation انتخاب با جایگذاری انجام می‌دهیم و نمونه دادگان را به مجموعه‌های کوچکتر تقسیم می‌کنیم در صورتی که در بوت‌استرپ می‌توانیم چندین مجموعه داده به اندازه مجموعه داده اصلی تولید کنیم.. به طور کلی Cross-Validation برای ارزیابی و اعتبارسنجی مدل بهتر است، در مقابل بوت‌استرپ می‌تواند در تخمین پارامترهای آماری و همچنین ساخت مدل‌های جمعی (Ensemble Models) می‌تواند موثرتر واقع شود.

بخش ۱۶

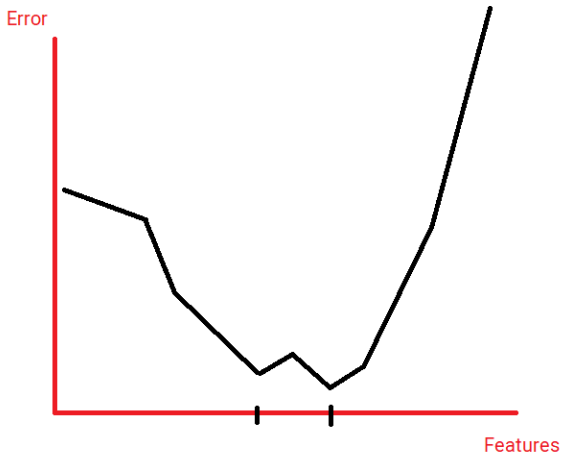
در این روش (که Nested Cross-Validation نیز نامیده می‌شود) یک Cross-Validation با K=2، پنج بار تکرار می‌شود. یعنی یکبار داده‌ها را به دو قسمت تقسیم کرده، یکی را برای آموزش و یکی را برای تست استفاده می‌کنیم و این عمل را ۴ بار دیگر نیز انجام می‌دهیم. ایده‌ای که اینجا مطرح می‌شود این است که ما دو تسک Hyperparameter Tuning و Model Selection را در نظر می‌گیریم. در ابتدا ممکن است این دو مفهوم تقریبا یکسان قلمداد شوند اما تفاوت ظریفی دارند. در Model Selection ما از روش‌هایی مانند AIC، BIC و ... برای یافتن تعداد ویژگی‌های مناسب و انتخاب یک مدل خوب بهره می‌بریم. اما در Hyperparameter Tuning ما یک مدل انتخاب کرده‌ایم و حال می‌خواهیم بهترین تنظیمات برای آن را پیدا کنیم. گاهی پیش می‌آید که می‌خواهیم هر دو تسک را با هم انجام دهیم. اگر فقط یکبار Cros-Validation انجام دهیم، انتخاب هایپرپارامترهایی که بهترین نتیجه را می‌دهند، مدل را به دیتاست بایاس می‌کند و باعث می‌شود یک تخمین خوش‌بینانه از خطا داشته باشیم. در چنین شرایطی از روش‌هایی مثل 5x2 Cross-Validation استفاده می‌شود.

بخش ۱۷

در بسیاری از مسائلی که در کتاب‌ها و مثال‌های تئوری می‌بینیم، داده‌ها به طور ذاتی مربوط به یک بعد خاصی هستند که برایمان مشخص است. تصویر زیر از کتاب Deep Learning (Ian Goodfellow et. al) را در نظر بگیرید:



در اینجا فرض کرده‌ایم که یک مدل خطی ناکافی (بایاس زیاد) بوده و یک مدل درجه ۹ نیز بیش از حد کافیست (واریانس زیاد) و فرض شده که داده‌ها به طور ذاتی از یک چندجمله‌ای درجه ۲ پیروی می‌کنند. اما بیایید فرض کنیم در دنیای واقعی هستیم، اولا شاید تنها یک زیرنمونه از داده‌ها را در دست داشته باشیم و ویژگی کل جامعه مانند نمونه نباشد، دوماً ممکن است تابع اصلا از چندجمله‌ای درجه ۲ (و یا کلا هیچ چندجمله‌ای) پیروی نکند، این فقط حدس ما بوده است و از ویژگی ذاتی تابع خبر نداریم. یا در مواقعی که نمودار ما به شکل زیر باشد:



پس در نتیجه بنظر روش Elbow همیشه بهترین و بهینه‌ترین نتیجه را به ما نمی‌دهد.

دیتاست شماره ۲

در این قسمت سوالات مطرح شده روی دیتاست بیماری‌های قلبی را پاسخ می‌دهم. کد این قسمت در فایل **A02_Task2.ipynb** قرار دارد.

بخش ۱

قضیه بیز یکی از مهم‌ترین قضایای آمار و احتمالات است که به ما کمک می‌کند احتمال شرطی یک رویداد را بر اساس اطلاعاتی که از قبل داریم محاسبه کنیم. دو مفهوم فرض (Hypothesis) و شواهد (Evidence) را در نظر بگیرید. فرض کنید می‌خواهیم احتمال این که یک فرض درست باشد را به شرط داشتن یک سری شواهد (اطلاعات قبلی) محاسبه کنیم. قضیه بیز این مساله را به صورت زیر فرمول‌بندی می‌کند:

$$P(Hypothesis \mid Evidence) = P(Hypothesis) \times \frac{P(Evidence \mid Hypothesis)}{P(Evidence)}$$

دقت کنید که اگر به تمام جامعه و ویژگی‌های آماری یک جامعه دسترسی داشته باشیم، به راحتی می‌توانیم احتمال یاد شده را به طور مستقیم محاسبه کنیم اما در بیشتر اوقات به این اطلاعات دسترسی نداریم، قضیه بیز به ما کمک می‌کند که این مقدار را بر اساس مقادیر دیگری محاسبه کنیم.

$$\underbrace{P(A|B)}_{\text{posterior}} = \underbrace{P(A)}_{\text{prior}} \times \frac{\overbrace{P(B|A)}^{\text{likelihood}}}{\underbrace{P(B)}_{\text{marginal}}}$$

مقدار سمت چپ معادله بالا یعنی احتمال صحیح بودن یک فرض در صورت وجود مدرک، احتمال موخر یا به‌روز شده (Posterior, Updated) نامیده می‌شود. احتمال پیشین (Prior) یعنی باوری که قبل از مشاهده شواهد داشته‌ایم. این احتمال در سمت راست معادله در یک کسر ضرب می‌شود که مخرج آن راستی‌نمایی (Likelihood) بوده که احتمال وجود شواهد به شرط درست بودن فرض است (این مقدار با احتمال پسین تفاوت دارد). مخرج این کسر احتمال حاشیه‌ای (Marginal Probability) شواهد نامیده می‌شود که یعنی احتمال وجود شواهد خواه فرض درست باشد یا غلط. در واقع هر چه مخرج کسر کوچکتر باشد، شواهدی که داریم مستدل‌تر و متقاعدکننده‌تر هستند.

از کلید واژه Naïve Bayes یا بیز ساده استفاده کرده‌اید، به طور خیلی خلاصه تفاوت آن با قضیه بیز این است که همه ویژگی‌های ورودی را مستقل از هم در نظر می‌گیرد که در قضیه بیز این فرض ساده وجود ندارد.

در یادگیری ماشین از بیز ساده می‌توان به عنوان یک روش یادگیری بانظارت بهره جست و دسته‌بندهایی ساخت که به آن‌ها Naive Bayes Classifiers گفته می‌شود. حال بیاید انواع مختلف آن را با هم مقایسه کنیم. فرمول زیر را در نظر بگیرید که y برچسب (فرض) و x_i ویژگی‌ها (شواهد) هستند (حاصل ضرب ناشی از استقلال ویژگی‌ها از هم است):

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

به طور کلی، تفاوت سه دسته‌بند یاد شده بر اساس توزیع آماری‌ای است که برای $P(x_i|y)$ فرض می‌کنند.

در دسته‌بند بیز ساده گاوسی (Gaussian Naive Bayes)، فرض می‌شود که مقادیر پیوسته هر ویژگی از توزیع نرمال پیروی می‌کنند، یعنی تابع درست‌نمایی (مخرج در قضیه بیز) به صورت زیر است:

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

به این ترتیب اگر k دسته یا کلاس داشته باشیم می‌توانیم برای هر دسته میانگین و واریانس را محاسبه کرده و پارامترهای توزیع نرمال را برای آن‌ها برآورد کنیم.

در دسته‌بند بیزی چندجمله‌ای (Multinomial Naive Bayes) یکی از بهترین گزینه‌ها برای یک دسته‌بند متنی بوده و در پردازش زبان طبیعی پرطرفدار است. این دسته بند برای مقادیر گسسته‌ای چون تعداد رخداد مناسب است. در این حالت برحسب توزیع احتمال چندجمله‌ای برداری از n ویژگی برای یک مشاهده به صورت $X = (x_1, x_2, \dots, x_n)$ با احتمالات $(p_1, \dots, p_2)$ درنظر گرفته می‌شود. در این حالت فرض کنید بردار X بیانگر تعداد مشاهداتی است که ویژگی خاصی را دارا می‌باشند. تابع درست‌نمایی به شکل زیر است:

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_{i=1}^{n} x_i)!}{\prod_{i=1}^{n} x_i!} \prod_{i=1}^{n} {p_{ki}}^{x_i}$$

در دسته‌بند بیزی برنولی (Bernoulli Naive Bayes) که بیشترین کاربرد را در دسته‌بندی متن‌های کوتاه دارد و شبیه گونه چندجمله‌ای است، به جای میزان رخداد یک ویژگی، وجود یا عدم وجود آن ویژگی درنظر گرفته می‌شود (متغیرهایی از جنس بولین). تابع درست‌نمایی برابر است با:

$$p(\mathbf{x} \mid C_k) = \prod_{i=1}^{n} p_{ki}^{x_i} (1 - p_{ki})^{(1-x_i)}$$

بخش ۲، ۳، ۴، ۵

در این بخش کلاس **GNB** که یک دسته‌بند بیزی گاوسی است پیاده‌سازی می‌شود. فرمول زیر را در نظر داشته باشید:

$$\mathrm{P(Y|X)} = \mathrm{P(Y)} \times \frac{\mathrm{P(X|Y)}}{\mathrm{P(X)}}$$

$\mathrm{P(Y)}$ را داشته باشیم. برای اینکار باید بدانیم درصد رخداد هر یک از کلاس‌ها در کل داده‌ها چقدر است. این کار با جدا کردن کلاس‌ها و ذخیره آن‌ها در یک دیکشنری انجام می‌شود که در متد **separate_classes** صورت گرفته است. چون از دسته‌بندی گاوسی استفاده می‌کنیم، پس یعنی فرض می‌کنیم که درست‌نمایی $\mathrm{P(X|Y)}$ از توزیع گاوسی پیروی می‌کند. پس باید قادر باشیم میانگین و انحراف معیار دسته‌ای داده را محاسبه کنیم. برای این کار یک yield می‌نویسم که در متد **summarize** وجود دارد. برای این از yield استفاده می‌کنم که با فراخوانی تابع و ورودی X، یک iterator از میانگین و انحراف معیارها بدون اینکه در حافظه‌ذخیره شوند، ایجاد شود. متد بعدی **gauss_distribution_function** است که با دریافت یک ویژگی و میانگین و انحراف معیار، توزیع گاوسی آن (در واقع صورت کسر، درست‌نمایی) را محاسبه می‌کند.

در ادامه باید مدل را آموزش دهیم. آموزش مدل در اینجا یعنی میانگین و انحراف معیار را برای هر ویژگی هرکلاس محاسبه کنیم، این به ما اجازه می‌دهد تا درست‌نمایی‌ها را محاسبه کرده و برای پیش‌بینی روی داده‌های تست به کار ببریم. همچنین احتمال پیشین نیز در این مرحله محاسبه می‌شود. این محاسبات در متد **fit** صورت می‌گیرند. این متد یک دیکشنری به نام **class_summary** برمی‌گرداند که کلیدهای آن کلاس‌ها و مقادیر آن یک دیکشنری دیگر حاوی احتمال پیشین آن کلاس و همچنین میانگین و انحراف معیار ویژگی‌های آن کلاس است. (نمونه‌هایی که آن کلاس را داشته‌اند جدا کرده و روی ویژگی‌های آن میانگین و انحراف معیار محاسبه می‌شود، یعنی مثلا حاصل برای میانگین لیستی به اندازه ویژگی‌هاست).

در فاز پیش‌بینی که در متد **predict** پیاده‌سازی شده است، ما باید احتمال پسین را به ازای کلاس‌های مختلف محاسبه کرده و کلاسی را که بیشترین احتمال را دارد به عنوان کلاس آن نمونه داده اعلام کنیم.

دقت کنید که در این مرحله هدف ما مقایسه احتمال‌های مختلف است و در این مقایسه، صورت کسر یا احتمال حاشیه‌ای برای یک نمونه در کلاس‌های مختلف برابر است، پس از محاسبه آن صرف نظر می‌کنیم و فقط مقدار زیر را برای ویژگی‌های ۱ تا n محاسبه می‌کنیم:

$$P(Y)\prod_{i=1}^{n} P(X_i|Y)$$

با مقایسه این مقدار برای کلاس‌های مختلف و ماکزیمم‌گیری، برچسب نمونه داده بدست می‌آید. در اینجا سه for داریم که در بیرونی‌ترین روی نمونه‌ها (سطرهای داده تست) حرکت کرده و کلاس هر نمونه را مشخص می‌کنیم، در وسطی روی کلاس‌ها حرکت کرده و دیکشنری class_summary که حاوی احتمال پیشین، میانگین و انحراف معیار است کمک گرفته و در for داخلی روی ویژگی‌ها حرکت کرده درست‌نمایی هر یک را محاسبه کرده و در هم ضرب می‌کنیم. خارج از دو for داخلی نیز روی مقادیر ماکزیمم گرفته و کلاس هر نمونه مشخص می‌شود.

ذکر این نکته خالی از لطف نیست که با اعمال تست Shapiro متوجه شدم که گویا ویژگی‌های X از یک توزیع نرمال پیروی نمی‌کنند.

پس از پیاده‌سازی GNB، ستون‌های خواسته شده را جدا کرده و داده‌ها را به نسبت ۸۰-۲۰ به آموزش و تست تقسیم می‌کنم. ابتدا داده‌های آموزش را روی مدل fit کرده و سپس روی داده‌های تست با predict خروجی می‌گیرم. نتایج ارزیابی مدل من به شرح زیر است:

	precision	recall	f1-score	support
0	0.6154	0.7619	0.6809	21
1	0.8571	0.7500	0.8000	40
accuracy			0.7541	61
macro avg	0.7363	0.7560	0.7404	61
weighted avg	0.7739	0.7541	0.7590	61

پس از مقایسه با GaussianNB از sklearn.naive_bayes با مورد عجیبی مواجه شدم. تمامی معیارهای ارزیابی خواسته شده، در دو مدل تا ۴ رقم اعشار با هم برابرند. این مورد اتفاقی نیست، با بیش از ۲۰ بار ریستارت کردن و اجرا کردن نوت‌بوک نیز این اتفاق می‌افتد. در واقع متد پیاده‌سازی شده دقیقا رفتاری مانند ماژول بسته sklearn از خود ارایه می‌دهد.