مؤسسه فرهنگی هنری
دیباگران تهران

یادگیری ماشین با زبان برنامه نویسی پایتون

python

طاهره اسمعیلی ابهریان
محسن علی مرادی

به نام خدا

یادگیری ماشین
با زبان برنامه نویسی
پایتون

مؤلفان:

مهندس طاهره اسمعیلی ابهریان

مهندس محسن علی مرادی

مؤسسه فرهنگی هنری دیباگران تهران

عنوان کتاب: **یادگیری ماشین با زبان برنامه نویسی پایتون**

مولفان : مهندس طاهره اسمعیلی ابهریان

مهندس محسن علی مرادی

ناشر: موسسه فرهنگی هنری دیباگران تهران

ویراستار: زهرا خانیانی

صفحه آرایی: نازنین نصیری

طراح جلد: داریوش فرسایی

نوبت چاپ: اول

تاریخ نشر: ۱۳۹۸

چاپ و صحافی: صدف

تیراژ: ۱۰۰ جلد

شابک: ۱-۲۱۵-۲۱۸-۶۲۲-۹۷۸

نشانی واحد فروش: تهران، میدان انقلاب، خ کارگر جنوبی، روبروی پاساژ مهستان، پلاک ۱۲۵۱

تلفن: ۶۶۴۱۰۰۴۶-۲۲۰۸۵۱۱۱

فروشگاههای اینترنتی دیباگران تهران :

WWW.MFTBOOK.IR

www.dibbook.ir

www.dibagarantehran.com

نشانی تلگرام:@mftbook نشانی اینستاگرام دیبا dibagaran_publishing

سرشناسه:اسمعیلی ابهریان،طاهره،۱۳۶۵-

عنوان و نام پدیدآور: یادگیری ماشین با زبان برنامه نویسی پایتون/مولفان: طاهره اسمعیلی ابهریان،محسن علی مرادی.

مشخصات نشر: تهران : دیباگران تهران :۱۳۹۸

مشخصات ظاهری:۱۸۰ص:مصور.

شابک: ۱-۲۱۵-۲۱۸-۶۲۲-۹۷۸

وضعیت فهرست نویسی: فیپا

موضوع:پایتون(زبان برنامه نویسی کامپیوتر)

موضوع: Python(computer program language)

شناسه افزوده:علی مرادی،محسن،۱۳۶۵-

رده بندی کنگره:۷۶/۷۳ QA

رده بندی دیویی:۰۰۵/۱۳۳

شماره کتابشناسی ملی: ۵۹۴۱۰۵۶

فهرست مطالب

فصل دوم

فصل سوم

فصل چهارم

فصل پنجم

فصل ششم

مقدمه ناشر

خط مشی کیفیت انتشارات مؤسسه فرهنگی هنری دیباگران تهران در عرصه کتاب هایی است که بتواند خواسته های به روز جامعه فرهنگی و علمی کشور را تا حد امکان پوشش دهد.

هر کتاب دیباگران تهران، یک فرصت جدید شغلی

حمد و سپاس ایزد منان را که با الطاف بیکران خود این توفیق را به ما ارزانی داشت تا بتوانیم در راه ارتقای دانش عمومی و فرهنگی این مرز و بوم در زمینه چاپ و نشر کتب علمی دانشگاهی، علوم پایه و به ویژه علوم کامپیوتر و انفورماتیک گام‌هایی هرچند کوچک برداشته و در انجام رسالتی که بر عهده داریم، مؤثر واقع شویم.

گستردگی علوم و توسعه روزافزون آن، شرایطی را به وجود آورده که هر روز شاهد تحولات اساسی چشمگیری در سطح جهان هستیم. این گسترش و توسعه نیاز به منابع مختلف از جمله کتاب را به عنوان قدیمی‌ترین و راحت‌ترین راه دستیابی به اطلاعات و اطلاع‌رسانی، بیش از پیش روشن می‌نماید.

در این راستا، واحد انتشارات مؤسسه فرهنگی هنری دیباگران تهران با همکاری جمعی از اساتید، مؤلفان، مترجمان، متخصصان، پژوهشگران، محققان و نیز پرسنل ورزیده و ماهر در زمینه امور نشر درصدد هستند تا با تلاش‌های مستمر خود برای رفع کمبودها و نیازهای موجود، منابعی پُربار، معتبر و با کیفیت مناسب در اختیار علاقمندان قرار دهند.

کتابی که در دست دارید با همت "**مهندسان طاهره اسمعیلی ابهریان- محسن علی مرادی** " و تلاش جمعی از همکاران انتشارات میسر گشته که شایسته است از یکایک این گرامیان تشکر و قدردانی کنیم.

کارشناسی و نظارت بر محتوا: زهره قزلباش

در خاتمه ضمن سپاسگزاری از شما دانش‌پژوه گرامی درخواست می‌نماید با مراجعه به آدرس dibagaran.mft.info (ارتباط با مشتری) فرم نظرسنجی را برای کتابی که در دست دارید تکمیل و ارسال نموده، انتشارات دیباگران تهران را که جلب رضایت و وفاداری مشتریان را هدف خود می‌داند، یاری فرمایید.

امیدواریم همواره بهتر از گذشته خدمات و محصولات خود را تقدیم حضورتان نماییم.

مدیر انتشارات

مؤسسه فرهنگی هنری دیباگران تهران
bookmarket@mft.info

به نام خداوند جان و خرد

کزین برتر اندیشه برنگذرد

مقدمهٔ مؤلفان

خداوند را بسیار شاکریم که برای ما فرصت تهیهٔ کتاب یادگیری ماشین با پایتون را به زبان فارسی فراهم کرد. تمام تلاش خود را به کار بستیم تا مباحث اصلی یادگیری ماشین را به‌همراه پیاده‌سازی آن‌ها با زبان پایتون در این کتاب به‌گونه‌ای جمع‌آوری کنیم که علاوه بر مکفی بودن توضیحات الگوریتم‌ها، برای خوانندگان محترم خسته‌کننده نباشد. ازآنجایی‌که سیستم‌های یادگیری ماشین به‌صورت گسترده‌ای در تمام رشته‌ها در حال استفاده است، سعی ما بر این شد تا چندان وارد مباحث پیچیدهٔ محاسبات و اثبات ریاضی الگوریتم‌ها نشویم تا این کتاب برای طیف گسترده‌ای از علاقه‌مندان به یادگیری ماشین قابل‌استفاده باشد.

چنانچه خوانندگان عزیز پیشنهادات یا انتقاداتی بر این نوشتار داشته باشند، سپاسگزار خواهیم بود از طریق آدرس الکترونیکی زیر، برای پر بار کردن آثار بعدی، ما را مطلع کنند.

طاهره اسمعیلی ابهریان، محسن علی مرادی

Tahereh.abharian@gmail.com

msn.alimoradi@gmail.com

پیشگفتار

درصورتی‌که در هر حوزه‌ای از فناوری اطلاعات و نرم‌افزار مشغول به فعالیت هستید، یا در حوزهٔ مهندسی برق، مکانیک، مهندسی پزشکی، شیمی، بیوشیمی، بیوفیزیک، بیوانفورماتیک و ژنتیک فعالیت می‌کنید، یا حتی اگر یک داروساز یا پزشک هستید، این کتاب قطعاً به شما کمک خواهد کرد تا درک بهتری از یادگیری یک سیستم هوشمند داشته باشید.

پردازش تصاویر دیجیتال از طریق الگوریتم‌های یادگیری ماشین و تحلیل‌های آماری، کاربرد گسترده‌ای در حوزه‌های مختلفی مانند تشخیص توده‌های سرطانی، تشخیص هویت افراد با استفاده از تصاویر اثرانگشت و عنبیه، تشخیص پلاک خودرو، مسیریابی ربات‌ها در صنایع فضایی و دفاعی و طراحی ماشین‌های بدون سرنشین و... دارد.

پیش‌بینی بیماری‌ها از طریق پردازش فاکتورهای خونی و ژنتیکی، کمک شایانی به سلامت جامعه جهانی خواهد کرد. پژوهشگران حوزهٔ بیوانفورماتیک در تلاش‌اند تا با کمک داروسازان و پزشکان، با استفاده از داده‌هایی که هر روز به سرعت تولیدشان افزوده می‌شود، درمان بسیاری از بیماری‌ها را بیابند.

هدف این کتاب این است که یادگیری ماشین را برای دانش‌پژوهان به‌گونه‌ای کاربردی معرفی کند تا خوانندگان عزیز با داشتن اطلاعات پایه‌ای از برنامه‌نویسی و زبان پایتون بتوانند در حوزهٔ فعالیت خود از این الگوریتم‌ها استفاده کنند.

اکنون فرصت را غنیمت می‌شماریم تا زحمات بی‌دریغ پدران و مادران عزیزمان را با جملاتی هرچند کوتاه ارج نهیم. پدر و مادر عزیزمان، شما قدم‌به‌قدم چگونه زیستن را در مکتب عشق به ما آموختید. در مقابل عظمت و شکوه شما ما را نه توان سپاس است و نه کلام وصف. به پاس محبت‌های بی‌دریغتان، که هرگز فروکش نمی‌کند، این کتاب را به شما پدر و مادر عزیزمان تقدیم می‌کنیم.

تقدیم به پدر و مادر عزیزمان و فرزند دلبندمان

طاهره اسمعیلی ابهریان، محسن علی مرادی

فصل اول

مفاهیم یادگیری ماشین

PYTHON

در حال حاضر ما در عصر داده زندگی می‌کنیم که توان محاسباتی بالا به همراه منابع قدرتمند ذخیرهٔ اطلاعات برای ما به شکل‌های گوناگون فراهم است. اطلاعات یا داده‌ها روزبه‌روز در حال افزایش است و تحلیل بسیاری از این داده‌ها و درنهایت تصمیم‌گیری مبتنی بر آن‌ها در بسیاری از حوزه‌ها عملاً از توانایی انسان خارج است. بسیاری از سایت‌های تجاری مانند موتورهای جست‌وجو، شرکت‌های تبلیغاتی و مؤسسات مالی در تلاش‌اند تا به کمک الگوریتم‌های یادگیری ماشین[۱] از این داده‌ها برای پیش‌بینی رفتار مشتری، پیش‌بینی ریسک و پیشنهاد محتوای مناسب استفاده کنند. الگوریتم‌های یادگیری ماشین همچنین به‌صورت گسترده در علوم مختلف پایه و مهندسی در حال استفاده است مانند:

- بینایی ماشین[۲]
- بیوانفورماتیک[۳]
- پردازش زبان[۴]
- بازی‌ها[۵]
- سیستم‌های خبره[۶]
- رباتیک

آنچه به‌طورکلی در این فصل بیان می‌شود:

- مفاهیم کلی یادگیری ماشین
- سه رویکرد اصلی یادگیری ماشین
- آشنایی با روند کلی یک سیستم یادگیری ماشین
- نصب پایتون به‌منظور یادگیری ماشین

۱-۱- یادگیری ماشین چیست؟

انسان‌ها در این لحظه باهوش‌ترین و پیشرفته‌ترین گونه‌های روی زمین هستند، زیرا می‌توانند مشکلات پیچیده‌ای را با تفکر، ارزیابی و حل کنند. از طرف دیگر، هوش مصنوعی هنوز در مرحلهٔ اولیهٔ خود است و از بسیاری جنبه‌ها از هوش انسانی پیشی نگرفته است. سؤال اینجاست که ضرورت استفاده از یادگیری ماشین چیست؟ مناسب‌ترین دلیل برای انجام این کار، «تصمیم‌گیری بر اساس داده‌ها، با کارایی و مقیاس است.»

یادگیری ماشین کاربرد الگوریتم‌هایی است که برای ما حجم عظیمی از داده را به دانش تبدیل می‌کنند. یادگیری ماشین تلاش می‌کند که برنامه‌هایی را طراحی کند که از داده‌ها آموزش ببینند. به‌عبارت‌دیگر، این الگوریتم‌ها این

---

1. Machine Learning
2. Computer Visiom
3. Bioinformatics
4. Language processing
5. games
6. Expert systems

قابلیت را به ماشین می‌دهند تا رفتارش را متناسب با داده یا به عبارتی الگوها[۷] تغییر دهد. واقعیت این است که ما باید مشکلات دنیای واقعی را با راندمان در مقیاس بزرگ حل کنیم. به همین دلیل، نیاز به یادگیری ماشین به وجود می‌آید. گاهی با محیطی مواجه هستیم که علاوه بر پویا بودن عناصر محیط، در آن کمبود تخصص انسانی وجود دارد، بنابراین ما می‌خواهیم ماشینی برای یادگیری و تصمیم‌گیری‌های مبتنی بر داده وجود داشته باشد. این مثال‌ها می‌توانند پیمایش در مناطق ناشناخته یا سیارات فضایی باشند.

الگوریتم‌های یادگیری ماشین یک مدل ریاضی را بر اساس داده‌های نمونه، معروف به «داده‌های آموزش» برای پیش‌بینی یا تصمیم‌گیری بدون دستورالعمل‌های صریح برای انجام کار می‌سازند [۱, ۲].

ما در عصری زندگی می‌کنیم که به‌علت در دسترس بودن کتابخانه‌های متن باز قدرتمند، ورود به عرصهٔ یادگیری ماشین و استفاده از الگوریتم‌های قدرتمند برای مشاهدهٔ الگوهای موجود در داده‌ها و پیش‌بینی‌ها دربارهٔ وقایع آینده برای ما امکان‌پذیر و آسان است.

۱–۲– مدل یادگیری ماشین

قبل از بحث در مورد مدل یادگیری ماشین، ما باید تعریف رسمی زیر را که استاد میچل[۸] ارائه کرده است درک کنیم:

«به یک برنامهٔ رایانه‌ای گفته می‌شود که با استفاده از تجربهٔ (مجموعه داده) E، کلاسی از وظایف[۹] T را با اندازه‌گیری عملکرد[۱۰] P یاد می‌گیرد، اگر عملکرد آن در وظایف T، که با P اندازه‌گیری می‌شود، با تجربهٔ E بهبود یابد»[۱۱] [۳].

تعریف فوق در اصل بر سه پارامتر متمرکز است که مؤلفه‌های اصلی هر الگوریتم یادگیری است، این پارامترها عبارت‌اند از وظایف (T)، عملکرد (P) و تجربه (E).

۱–۳– چهار رویکرد اصلی یادگیری ماشین

در این بخش می‌خواهیم با سه رویکرد اصلی و کلی در یادگیری ماشین آشنا شویم:

- یادگیری با ناظر[۱۲]
- یادگیری بی‌ناظر[۱۳]
- یادگیری تقویتی[۱۴]

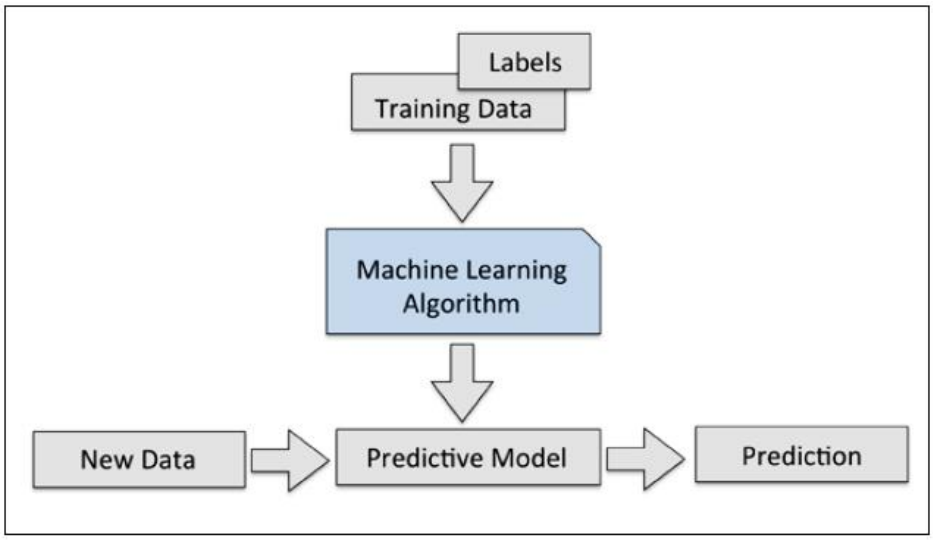
---

7. patterns
8. Tom Mitchell
9. Task
10. Performance
11. Experience
12. Supervised learning
13. Unsupervised learning
14. Reinforcement learning

۱-۳-۱- یادگیری با ناظر

در یادگیری با نظارت تلاش می‌شود که از داده‌های آموزش برچسب‌دار[۱۵] مدلی آموخته شود که ماشین بتواند در مورد برچسب داده‌هایی که تاکنون ندیده است و در آینده با آن‌ها مواجه می‌شود به‌خوبی تصمیم‌گیری کند. عبارت با نظارت به‌علت برچسب‌دار بودن مجموعه داده استفاده می‌شود به این معنا که برای هر الگوی این مجموعه داده مشخص شده است که این الگو به کدام دستهٔ تصمیم‌گیری ماشین متعلق است.

به‌عنوان‌مثال، مجموعه دادهٔ مربوط به سرطان رودهٔ بزرگ [۴] شامل ۶۲ نمونهٔ سلولی از بافت‌های روده است که تعدادی از این نمونه‌ها مربوط به بافت‌های تومور و تعدادی از آن‌ها مربوط به بافت‌های سالم از رودهٔ بیماران است. به‌عبارت‌دیگر، برچسب برخی از نمونه‌ها می‌تواند ۱ (بافت سرطانی) و برچسب برخی دیگر ۰ (بافت سالم) باشد. درصورتی‌که الگوریتم یادگیری ماشین بر اساس این مجموعه داده آموزش ببیند می‌تواند در مورد سالم یا سرطانی بودن نمونه‌های بافت دیگر تصمیم‌گیری کند.



شکل ۱-۱: شمای کلی سیستم یادگیری با نظارت [۵]

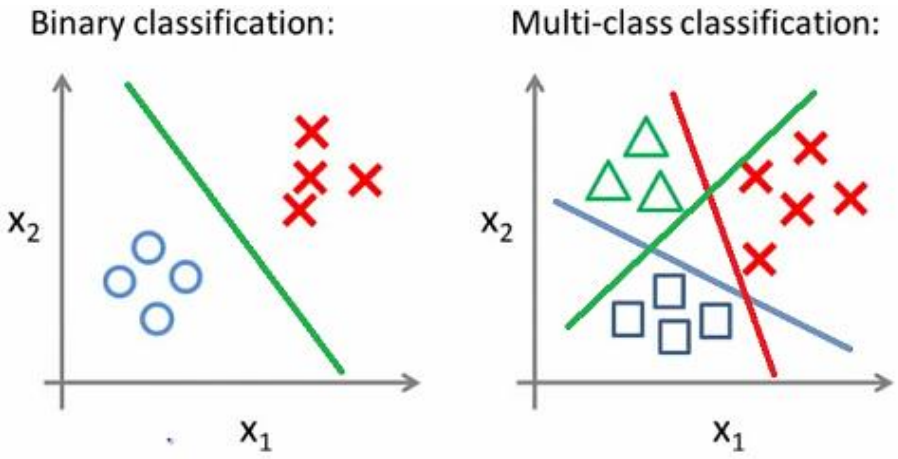
یادگیری با نظارت به دو صورت مطرح است:

- **دسته‌بندی:**[۱۶] در دسته‌بندی تلاش می‌شود تا برچسب نمونه‌ها بر اساس نمونه‌های پیشین پیش‌بینی شود و تعیین شود که هر نمونه به کدام دسته متعلق است. برچسب دسته‌ها مقادیری گسسته هستند. مثالی که در صفحه قبل برای تشخیص بافت سالم از بافت سرطانی مطرح شد، نمونه‌ای از دسته‌بندی دودویی[۱۷] است. به‌عنوان‌مثال،

15. labeled
16. classification
17. Binary classification

برای توضیح دسته‌بندی چنددسته‌ای[۱۸] می‌توان مجموعه داده‌ای را در نظر گرفت که در آن ویژگی‌های مربوط به آمادگی جسمانی ورزشکاران المپیک در رشتهٔ ورزشی خاصی ذخیره شده است. برچسب‌های آن‌ها در سه سطح با توجه به عملکرد آن‌ها در المپیک تعیین می‌شود: آمادگی جسمانی کامل، خوب و متوسط. حال اگر بعد از آموزش الگوریتم فاکتورهای آمادگی جسمانی فرد جدیدی را به سیستم بدهیم با توجه به عملکرد ورزشکاران قبلی در المپیک، سطح آمادگی این ورزشکار را مشخص می‌کند و این کمک مهمی به مربی در انتخاب یا عدم انتخاب فرد در مسابقات آینده المپیک خواهد کرد. مثال ساده از این دو مدل دسته‌بندی را در شکل ۱-۲ می‌بینید.

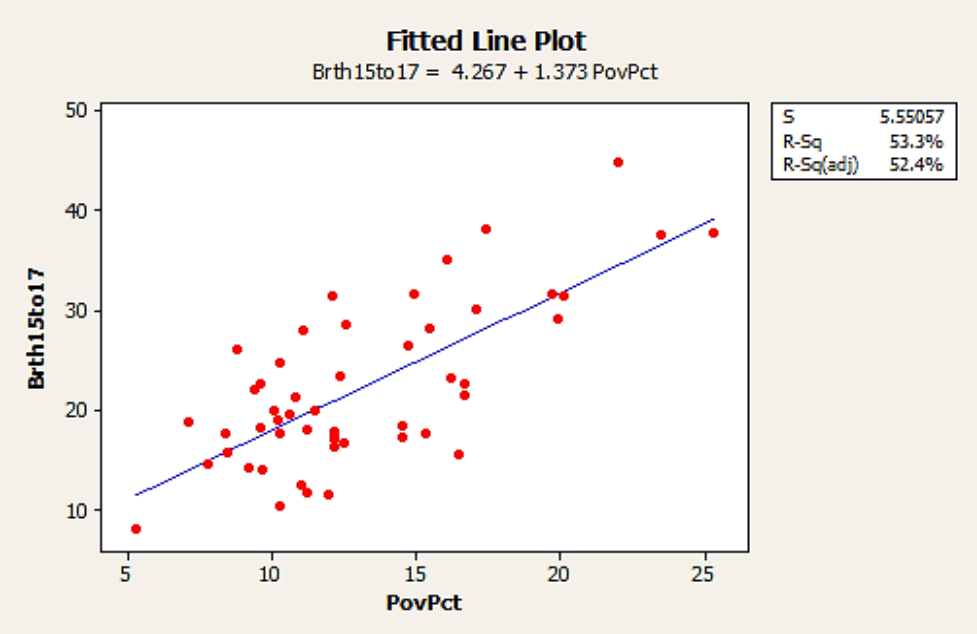
- **تحلیل خطی‌سازی**:[۱۹] تحلیل خطی روشی است که در آن ارتباط بین متغیر هدف (متغیر وابسته)[۲۰] و متغیرهای مستقل[۲۱] سنجیده می‌شود. متغیر هدف متغیر اصلی است که تلاش می‌شود پیش‌بینی شود و متغیرهای مستقل متغیرهایی هستند که فرض شده است روی متغیر هدف تأثیرگذارند و می‌توان با آن‌ها متغیر هدف را پیش‌بینی کرد.



شکل ۱-۲: مفهوم دسته‌بندی دودویی و دسته‌بندی چند دسته‌ای

به‌عنوان‌مثال، فرض می‌کنیم می‌خواهیم تأثیر نرخ زایش در میان نوجوانان را در نرخ فقر بسنجیم. متغیر مستقل عبارت است از نرخ زایش در میان ۱۰۰۰ زن با سنین بین ۱۵ تا ۱۷ سال در سال ۲۰۰۲
و متغیر هدف برابر نرخ فقر که عبارت است از درصد جمعیت خانه‌دار که درآمدشان زیر خط فقر است.
ارتباط بین متغیر هدف و متغیر مستقل با تخمین بهترین خط مستقیم در شکل ۱-۳ نشان داده شده است[۶].

18. Multiclass classification
19. Regression analysis
20. Dependent variable
21. Independent variables



شکل ۱-۳: خط رگرسیون برای نمایش تأثیر زایش نوجوانان در سطح فقر [۶].

خطی‌سازی بر اساس تعداد متغیرهای مستقل، ساختار خط و نوع متغیر وابسته انواع مختلفی دارد: خطی‌سازی خطی،[۲۲] خطی‌سازی لجستیک،[۲۳] خطی سازی چندجمله‌ای[۲۴] و... .

۱-۳-۲- یادگیری بدون ناظر

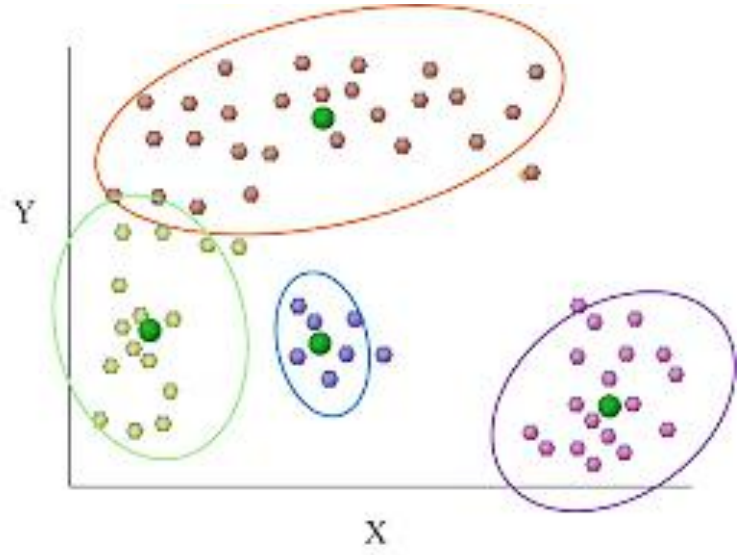
در این نوع یادگیری تلاش می‌شود اطلاعات معناداری از داده‌ها بدون راهنمایی برچسب‌ها استخراج شود. پس لزوماً به این معنا نیست که مجموعه دادهٔ مورد استفاده حتماً باید بدون برچسب باشد. می‌توان از مجموعه دادهٔ برچسب‌دار استفاده کرد اما برچسب‌های آن را در نظر نگرفت.

- **خوشه‌بندی داده:**[۲۵] از جمله روش‌های یادگیری است که بدون راهنمایی برچسب، داده‌ها را بر اساس میزان شباهتشان گروه‌بندی می‌کند به‌گونه‌ای که داده‌هایی که با هم در یک گروه قرار می‌گیرند با داده‌های گروه‌های دیگر متفاوت هستند. شباهت یا نزدیکی داده‌ها به هم با معیار[۲۶]های مختلفی در روش‌های مختلف خوشه‌بندی داده سنجیده می‌شود. خوشه‌بندی روش مناسبی برای استخراج ارتباطات معنادار میان داده‌هاست.

به‌عنوان‌مثال، الگوریتم‌های خوشه‌بندی برای بازاریاب‌ها امکان گروه‌بندی مشتریان را بر اساس علاقه‌هایشان فراهم می‌کند تا بتوانند برنامه‌های بازاریابی مجزایی را برای هر گروه توسعه دهند [۵].

---

22. Linear regression
23. Logestic regression
24. Polynomial regression
25. Data clustering
26. metric



شکل ۱-۴: مثالی از خوشه‌بندی داده

- **کاهش ابعاد داده[۲۷] برای فشرده‌سازی داده:** معمولاً در هر مجموعه داده‌ای که با آن مواجهیم سطرهای آن نمونه‌ها[۲۸] و ستون‌های آن ویژگی‌های هر نمونه را مشخص می‌کنند. به‌عبارت‌دیگر، تعداد ستون‌های ستون‌های مجموعه داده، ابعاد آن مجموعه داده را مشخص می‌کند. بسیاری از مجموعه داده‌ها ابعاد بسیار بالایی دارند. در این مجموعه داده‌ها هر نمونه‌ای با اندازه‌گیری‌های زیادی در مجموعه داده قرار گرفته است که می‌تواند چالش بزرگی در فضای ذخیره‌سازی و کارایی محاسباتی الگوریتم‌های یادگیری ماشین به وجود آورد.

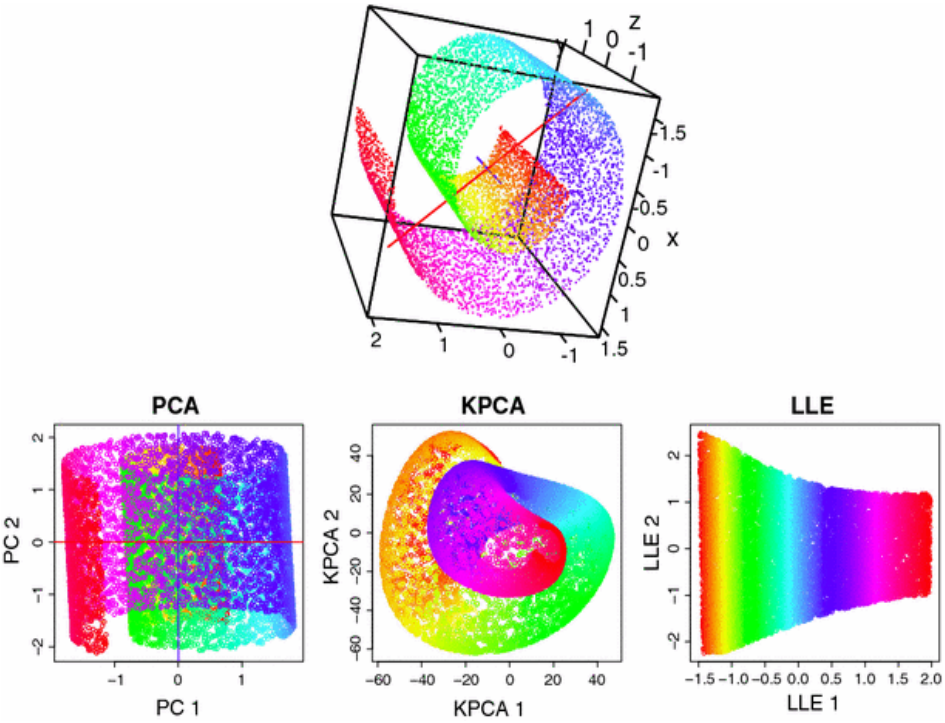
به‌عنوان‌مثال، در مجموعه دادهٔ بیان ژن[۲۹] نمونه‌های مختلف سلولی، به‌عنوان نمونه‌های مجموعه داده و بیان‌های مختلفی از ژن‌های مختلف به‌عنوان ویژگی‌های هر نمونهٔ مجموعه داده مطرح می‌شوند. غالباً در این نوع مجموعه داده‌ها ما با حجم بالایی از ویژگی مواجه هستیم.

روش‌های کاهش ابعاد داده با حذف ویژگی‌های غیرمرتبط کارایی اجرای برنامه را افزایش می‌دهند.

در شکل ۱-۵ نتیجهٔ حاصل از اعمال سه روش از روش‌های کاهش ابعاد داده را مشاهده می‌کنید. همان‌طور که مشخص است کاهش ابعاد داده با کمک روش تعبیه شدهٔ خطی محلی (LLE[۳۰]) در مورد این مجموعه داده نسبت به دو روش دیگر به نحو مناسب‌تری صورت گرفته است، زیرا داده‌ها به‌خوبی تفکیک شده‌اند و همپوشانی کمتری نسبت به روش‌های دیگر دارند.

---

27. Dimensionality reduction
28. Samples
29. Gene expression data set
30. Localy linear embedding



شکل ۱-۵: اعمال روش‌های خطی و غیرخطی کاهش ابعاد [۷].

۱-۳-۳- یادگیری نیمه‌نظارتی[۳۱]

اگر برخی از داده‌های مجموعه داده برچسب داشته باشند و برخی دیگر برچسب نداشته باشند در این صورت داده‌هایی که برچسب ندارند در بخش آموزش الگوریتم استفاده می‌شوند و داده‌هایی که برچسب دارند در بخش ارزیابی الگوریتم استفاده می‌شوند. یادگیری نیمه‌نظارتی اغلب زمانی استفاده می‌شود که امکان برچسب‌گذاری بخش زیادی از داده‌ها وجود ندارد که می‌تواند دلایل متفاوتی داشته باشد. به‌عنوان‌مثال، ممکن است نیاز به ابزارهای خاصی برای برچسب‌گذاری باشد یا هزینهٔ زیادی داشته باشد.

۱-۳-۴- یادگیری تقویتی[۳۲]

در یادگیری تقویتی، هدف توسعهٔ سیستمی است (عامل)[۳۳] که کارایی‌اش را بر اساس تعامل با محیط بهبود می‌بخشد [۵]. در این نوع یادگیری، عامل به‌جای اینکه از تجربه‌های پیشین استفاده کند سعی می‌کند خودش تجربه کند و بر اساس بازخوردهایی که دریافت می‌کند عملکرد خود را تنظیم می‌کند. عامل به‌خاطر عملی که

31. Semi-supervied learning
32. Reinforcement learning
33. Agent

انجام می‌دهد پاداش[۳۴] دریافت می‌کند یا جریمه[۳۵] می‌شود. طراح عامل تابع پاداش را تعریف می‌کند و هدف عامل این است که برای دریافت پاداش بیشتر این تابع را بیشینه کند. در ابتدا با آزمون‌وخطا و کاملاً تصادفی عملکرد خود را انتخاب می‌کند و در پایان کارش را با تاکتیک‌های پیچیده به پایان می‌رساند.

۱-۴- معرفی مجموعه داده[۳۶]

مجموعه داده معمولاً در سطرهایش نمونه‌های آن مشخص است و در ستون‌هایش پارامترهایی که برای هر نمونه اندازه‌گیری شده است. به‌عنوان‌مثال، در شکل ۱-۶ مجموعه دادهٔ مربوط به تشخیص دیابت نشان داده شده است. هر سطر این مجموعه داده مربوط به یک بیمار است. پس نمونه‌ها بیماران هستند و ستون‌های این مجموعه داده فاکتورهایی را نشان می‌دهد که برای تشخیص دیابت فرد اندازه‌گیری شده‌اند که به آن‌ها ابعاد مجموعه داده یا ویژگی[۳۷] می‌گوییم. به‌عنوان‌مثال، سن فرد، تعداد دفعات بارداری او، غلظت گلوکز پلاسما و... . یک سیستم یادگیری ماشین برای تشخیص ابتلای فرد به دیابت قطعاً به ویژگی‌های مجموعه داده نیازمند است. برچسب‌های این مجموعه داده نیز در ستون آخر مشخص است که یا منفی است یا مثبت. این مجموعه داده حاوی ۱۴ نمونه و ۸ ویژگی است که می‌توان آن را به‌صورت یک ماتریس ۱۴X۸ نشان داد.

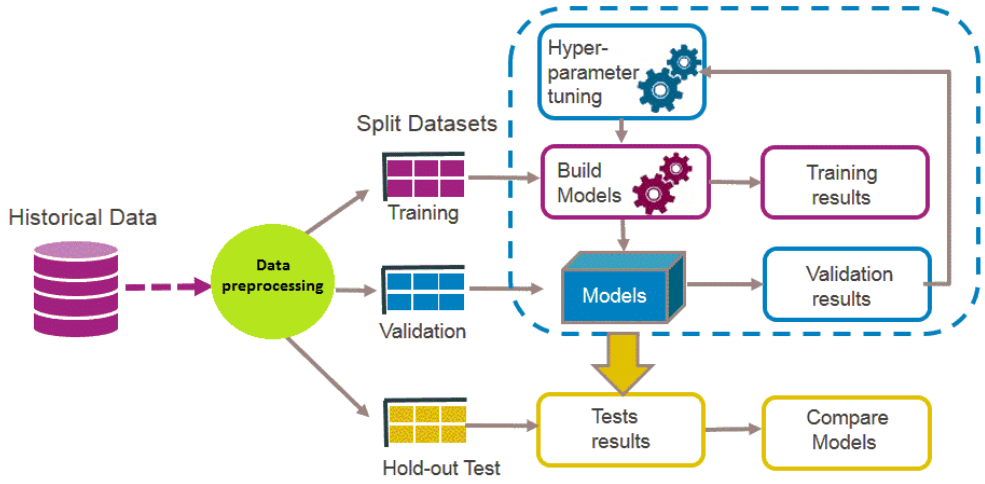
No.	Number of times pregnant	Plasma glucose concentration	Diastolic blood pressure	Triceps skin fold thickness	2-Hour serum insulin	Body mass index	Diabetes pedigree function	Age	Diabetes
1	6	148	72	35	0	33.6	0.627	50	tested_positive
2	1	85	66	29	0	26.6	0.351	31	tested_negative
3	8	183	64	0	0	23.3	0.672	32	tested_positive
4	1	89	66	23	94	28.1	0.167	21	tested_negative
5	0	137	40	35	168	43.1	2.288	33	tested_positive
6	5	116	74	0	0	25.6	0.201	30	tested_negative
7	3	78	50	32	88	31.0	0.248	26	tested_positive
8	10	115	0	0	0	35.3	0.134	29	tested_negative
9	2	197	70	45	543	30.5	0.158	53	tested_positive
10	8	125	96	0	0	0.0	0.232	54	tested_positive
11	4	110	92	0	0	37.6	0.191	30	tested_negative
12	10	168	74	0	0	38.0	0.537	34	tested_positive
13	10	139	80	0	0	27.1	1.441	57	tested_negative
14	1	189	60	23	846	30.1	0.398	59	tested_positive

شکل ۱-۶: مجموعه داده برای تشخیص بیماری دیابت [۸] .

34. Reward
35. Penalty
36. Dataset
37. Feature

۱–۵– معرفی روند کلی یک سیستم یادگیری ماشین

در بخش‌های پیشین راجع به سه نوع کلی از الگوریتم‌های یادگیری ماشین صحبت کردیم. در این بخش می‌خواهیم به بررسی عوامل دیگری بپردازیم که در یک سیستم یادگیری ماشین نقش دارند. روند کلی یک سیستم یادگیری ماشین در شکل ۱–۷ نشان داده شده است که در ادامه به بررسی بخش‌های مختلف آن می‌پردازیم.



شکل ۱–۷: نگاه کلی به روند یک سیستم یادگیری ماشین [۹].

۱–۵–۱– پیش‌پردازش داده[۳۸]

پیش‌پردازش داده مرحلهٔ بسیار مهمی در یادگیری ماشین است. داده‌های اولیهٔ جمع‌آوری شده اغلب نیازمند پیش‌پردازش‌هایی هستند تا برای اعمال الگوریتم‌های یادگیری ماشین آماده شوند. مراحل پیش‌پردازش داده عبارت است از:

- **تمیز کردن داده:**[۳۹]
  - ✓ **داده‌های گم شده:**[۴۰] گاهی در برخی از نمونه‌ها بعضی از ویژگی‌ها مقدار ندارند که اگر تعداد این نمونه‌ها زیاد باشد و مجموعه داده بزرگ باشد اغلب این نمونه‌ها حذف می‌شوند.

گاهی هم در صورت امکان (اگر تعداد این نمونه‌ها کم باشد) می‌توان این مقادیر را با میانگین مقادیر آن ویژگی یا محتمل‌ترین مقدار برای آن به‌صورت دستی پر کرد.

---

38. Data preprocessing
39. Data cleaning
40. Missing data

✓ **داده‌های هرز:**[۴۱] داده‌های بی‌معنی هستند که ممکن است هنگام جمع‌آوری داده به‌صورت اشتباه وارد مجموعه داده شده باشند. که می‌توان به کمک روش‌هایی که در ادامه آورده می‌شوند با این مشکل مقابله کرد.

- **روش قطعه‌بندی:**[۴۲] این روش برای کاهش تأثیر داده‌های هرز، داده‌های مرتب شده را هموار[۴۳] می‌کند. داده‌ها بعد از مرتب شدن به قطعاتی با اندازه‌های برابر تقسیم‌بندی می‌شوند. سپس در هر قطعه تمام مقادیر آن قطعه با روش‌های مختلفی جایگزین می‌شوند. به‌عنوان‌مثال، می‌توان تمام مقادیر هر قطعه را با میانگین مقادیر آن قطعه جایگزین کرد یا اینکه می‌توان در هر قطعه هر مقدار را با نزدیک‌ترین مقدار مرز آن قطعه جایگزین کرد.
- **خطی‌سازی:** در این روش داده‌ها با برازش آن‌ها روی یک تابع خطی‌سازی، هموار می‌شوند و این باعث کم شدن تأثیر داده‌های هرز می‌شود.
- **خوشه‌بندی داده:** در این روش داده‌های مشابه در یک خوشه قرار می‌گیرند در این صورت داده‌های هرز به‌علت اینکه شبیه داده‌ها نیستند خارج از خوشه‌ها قرار می‌گیرند.

- **تبدیل داده‌ها :**[۴۴] برای اینکه داده‌ها آمادهٔ پردازش با الگوریتم‌های یادگیری شوند نیازمند برخی تبدیل‌ها هستند:

✓ **نرمال‌سازی:**[۴۵] در این روش داده‌ها در محدودهٔ خاصی مقیاس می‌شوند. در محدودهٔ 0 تا 1 یا در محدودهٔ 1− تا 1.

✓ **استخراج ویژگی:**[۴۶] ویژگی‌های جدیدی با استفاده از مجموعه ویژگی‌ها ساخته می‌شود.

✓ **گسسته‌سازی:**[۴۷] مقادیر خام ویژگی‌های عددی (مانند سن) سطح‌بندی می‌شوند به‌طوری‌که مقادیر بازه‌ای جایگزین مقادیرشان می‌شود (مانند 0-10، 10-15، ...) یا برچسب‌های مفهومی به خود می‌گیرند (مانند جوان، بالغ، مسن و...).

✓ **تولید سلسله‌مراتب مفهومی برای داده‌های اسمی:**[۴۸] به‌عنوان‌مثال ویژگی مانند خیابان می‌تواند به مفاهیم سطوح بالاتر مانند شهر یا کشور در مجموعه داده تعمیم داده شود.

- **کاهش داده:**[۴۹] گاهی مجموعه داده آن‌قدر حجیم است که کاوش در این داده‌ها عملی نیست یا مدت زمان زیادی را می‌طلبد. روش‌های کاهش داده، مجموعه دادهٔ کاهش‌یافته‌ای را ارائه می‌کنند که علاوه بر اینکه حجم

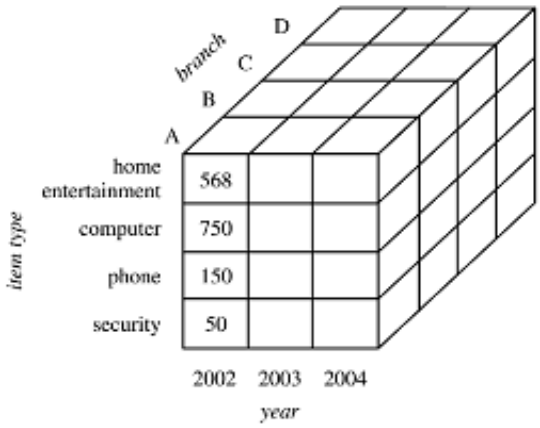
---

41. Noise data
42. Binning method
43. Smoothing
44. Data transformation
45. Normalization
46. Feature selection
47. Discretization
48. Concept Hierarchy Generation for nominal data
49. Data reduction

کمتری دارند یکپارچگی مجموعه داده را نیز حفظ می‌کنند [۱۰]. کاوش در این مجموعه داده، کاراتر یا تقریباً برابر کاوش در مجموعه دادهٔ اصلی است. روش‌های کاهش داده در ادامه بیان شده‌اند:

✓ **جمع‌آوری اطلاعات به‌صورت مکعب داده:**[۵۰] هنگامی‌که عملگرهای جمع‌آوری مانند جمع، میانگین، بیشینه یا کمینه کردن برحسب نیاز به داده اعمال شوند، اطلاعات خلاصه‌تری را برای ما فراهم می‌آورند. به‌عنوان‌مثال، ممکن است در تحلیل داده نیازی به داشتن میزان فروش هر فصل در مجموعه داده نباشد و داشتن میزان فروش سالانه کافی باشد. ازاین‌رو با استفاده از عملگر جمع، اطلاعات خلاصه‌تر و مفیدتری فراهم می‌شود. یک مکعب داده، اطلاعات جمع‌آوری شدهٔ چندبعدی است [۱۰]. در شکل ۱-۸ یک مکعب داده نشان داده شده است.

✓ **انتخاب زیرمجموعه‌ای از ویژگی‌ها:**[۵۱] ویژگی‌های کاملاً مرتبط برای فرایند پردازش انتخاب می‌شوند و بقیهٔ ویژگی‌ها در نظر گفته نمی‌شوند. به‌عنوان‌مثال در دسته‌بندی مشتریان یک شرکت، شماره تلفن آن‌ها می‌تواند ویژگی غیرمرتبط باشد، درحالی‌که سن آن‌ها یک ویژگی مرتبط است. البته اغلب اوقات انتخاب ویژگی‌های مرتبط به این سادگی نیست، مخصوصاً زمانی که رفتار داده به‌خوبی شناخته شده نباشد. درصورتی‌که ویژگی‌های غیرمرتبط از مجموعه داده حذف نشوند مدل یادگیری به دست آمده کیفیت پایینی خواهد داشت. از طرف دیگر، نگه داشتن این ویژگی‌ها از سرعت پردازش می‌کاهد. هدف انتخاب ویژگی این است که مجموعه‌ای با تعداد کمینه‌ای از ویژگی‌ها را انتخاب کند، به‌گونه‌ای که توزیع احتمال داده‌های کلاس‌ها تا جایی که ممکن است به توزیع اصلی با تمام ویژگی‌ها نزدیک باشد. با روش‌های مختلف انتخاب ویژگی در فصل‌های آینده بیشتر آشنا خواهید شد.



شکل ۱-۸: یک مکعب داده برای فروش در شرکت `AllElectronics` [۱۰].

[50]. Data cube aggregation
[51]. Feature subset selection

- ✓ **کاهش عددی:**[۵۲] در این روش می‌توان داده را با مدل‌های پارامتریک (به‌جای اینکه کل داده ذخیره شود پارامترهای مدل ذخیره می‌شوند) یا مدل‌های غیرپارامتریک (مانند خوشه‌بندی، نمونه‌برداری[۵۳] و هیستوگرام) تخمین زد [۱۰].
- ✓ **کاهش ابعاد:**[۵۴] روش‌های رمزنگاری برای کاهش اندازهٔ مجموعه داده استفاده می‌شود [۱۰].
- ✓ **گسسته‌سازی و تولید سلسله‌مراتب مفهومی:**[۵۵] مقادیر خام ویژگی‌ها با سطوح مفهومی جایگزین می‌شوند. گسسته‌سازی داده گونه‌ای از روش کاهش عددی است که برای تولید سلسله‌مراتب مفهومی بسیار مناسب است [۱۰].

مطلبی که در اینجا بسیار حائز اهمیت است این است که زمانی که صرف پردازش الگوریتم کاهش ویژگی می‌شود نباید برابر یا بیشتر باشد با زمانی که برای ایجاد مدل یادگیری روی مجموعه داده با اندازهٔ کاهش یافته ذخیره کرده‌ایم.

۱-۵-۲- تقسیم مجموعه داده

چگونگی تقسیم مجموعه داده به دو عامل اصلی بستگی دارد: اول تعداد نمونه‌ها و دوم نوع مدل یادگیری. برخی از مدل‌ها برای آموزش نیاز به تعداد داده‌های قابل‌توجهی دارند، بنابراین تقسیم‌بندی باید به‌گونه‌ای باشد که مجموعه داده آموزش بزرگی داشته باشیم. آموزش و تنظیم مدل‌هایی که هایپرپارامتر[۵۶]های کمی دارند آسان است و می‌توان حجم کمی از داده را برای مجموعهٔ تأیید در نظر گرفت. از طرف دیگر، درصورتی‌که مدل هایپرپارامترهای زیادی داشته باشد مجموعهٔ اعتبارسنجی بزرگ‌تری موردنیاز است. همچنین درصورتی‌که مدل یادگیری هایپرپارامتر نداشته باشد یا هایپرپارامترهای آن به‌گونه‌ای باشد که به‌آسانی قابل‌تنظیم نباشد می‌توان مجموعهٔ اعتبارسنجی برای آن مدل در نظر نگرفت **[۱۱]**.

- **مجموعه دادهٔ آموزش:**[۵۷] مجموعه‌ای از داده‌ها که برای آموزش مدل از آن استفاده می‌کنیم. مدل این داده‌ها را می‌بیند و یاد می‌گیرد.
- **مجموعه دادهٔ اعتبارسنجی:**[۵۸] نمونه‌هایی از داده که برای مدلی که با مجموعه دادهٔ آموزش، آموزش دیده است، ارزیابی بدون جهتی ارائه می‌کند. این ارزیابی چندین بار انجام می‌شود و هر بار هایپرپارامترهای مدل مجدداً تنظیم می‌شود [۱۲].

---

52. Numerosity Reduction
53. sampling
54. Dimensionality Reduction
55. discretization and concept hierarchy generation
۵۶. پارامتری که قبل از شروع فرایند یادگیری باید مقدار بگیرد.
57. Train dataset
58. Validation dataset

- **مجموعه دادهٔ آزمون:**[۵۹] نمونه‌هایی از داده که برای مدلی که آموزش دیده است، ارزیابی بدون جهتی ارائه می‌کند. درواقع، نتیجهٔ مدل بر اساس مجموعه داده آزمون بیان می‌شود. این مجموعه داده فقط یک بار زمانی که مدل به‌طور کامل آموزش دیده باشد (با استفاده از مجموعه داده‌های آموزش و اعتبارسنجی) استفاده می‌شود.

به‌طورکلی می‌توان گفت نرخ تقسیم مجموعه داده کاملاً به مسئلهٔ مورد بررسی بستگی دارد و هرچه مدل‌های بیشتری را آموزش دهید تقسیم مجموعه داده راحت‌تر خواهد بود و تجربهٔ بیشتری دراین‌باره کسب خواهید کرد.

۱-۵-۳- انتخاب و آموزش یک مدل یادگیری

بسیاری از الگوریتم‌های یادگیری ماشین برای حل مشکلات مختلفی طراحی شده‌اند. این سؤال ممکن است برایتان پیش آید که از کجا بدانم کدام الگوریتم را استفاده کنم؟ پاسخ به این سؤال مصداق جمله مشهوری از آقای ماسلو[۶۰] است: «به نظر من وسوسه‌انگیز است شرایطی که تنها ابزاری که دارید چکش است، آنگاه با همه‌چیز با فرض اینکه میخ است، برخورد می‌کنید» [۱۳]. به‌عنوان‌مثال در حوزهٔ دسته‌بندی داده، باید چند الگوریتم دسته‌بندی را برای آموزش در نظر بگیریم. هر الگوریتمی خواص ذاتی خودش را دارد و هیچ الگوریتمی بدون فرضیات ما در مورد مسئله نمی‌تواند نسبت به دیگری برتری پیدا کند. بنابراین در عمل، لازم است که حداقل تعداد معدودی از الگوریتم‌های مختلف را برای آموزش مدل انتخاب کنیم و با مقایسه عملکرد آن‌ها، بهترین مدل را انتخاب کنیم. اما قبل از مقایسه مدل‌های مختلف، ابتدا باید برای اندازه‌گیری عملکرد مدل‌ها، معیار مناسبی داشته باشیم. یکی از معیارهای متداول، دقت طبقه‌بندی است که به‌عنوان نسبت نمونه‌های صحیح طبقه‌بندی شده تعریف می‌شود.

برای اینکه بتوانیم از عملکرد مدل‌هایی که آموزش دیده‌اند قبل از استفاده از مجموعه داده آزمون یا برخورد با داده‌های واقعی مطلع شویم از تکنیک‌های مختلف «اعتبارسنجی متقابل»[۶۱] استفاده می‌کنیم. بعد از کنار نهادن مجموعه داده آزمون برای مرحلهٔ ارزیابی، ادامهٔ مجموعه داده به زیرمجموعه‌های آموزشی و اعتبارسنجی تقسیم می‌شود. سپس مدل با استفاده از مجموعه دادهٔ آموزش، آموزش می‌بیند و با استفاده از مجموعه دادهٔ اعتبارسنجی اعتبار مدل سنجیده می‌شود و هایپرپارامترها تنظیم می‌شوند. تقسیم مجموعه داده به داده‌های آموزش و اعتبارسنجی چندین بار تکرار می‌شود و هر بار آموزش و اعتبارسنجی صورت می‌گیرد. چگونگی تقسیم روش‌های مختلفی دارد به‌عنوان‌مثال، در روش اعتبارسنجی `k-fold` بعد از کنار نهادن مجموعه داده آزمون، ادامهٔ مجموعه داده به `k` قسمت تقسیم می‌شود. در هر تکرار یک بخش از `k` قسمت مجموعه داده، به‌عنوان مجموعه دادهٔ اعتبارسنجی در نظر گرفته می‌شود و `k-1` بخش باقی‌مانده برای آموزش استفاده می‌شوند. این کار آن‌قدر تکرار می‌شود تا هر `k` بخش یک بار به‌عنوان مجموعه داده اعتبارسنجی در نظر گرفته شده باشند. در ضمن ما

---

[59]. Test dataset
[60]. Abraham Maslow
[61]. Cross validation

نمی‌توانیم انتظار داشته باشیم که پارامترهای پیش‌فرض الگوریتم‌های مختلف یادگیری ارائه شدهٔ کتابخانه‌های نرم‌افزاری برای کار مشکل خاص ما بهینه باشند. بنابراین، ما از تکنیک‌های بهینه‌سازی هایپرپارامترها استفاده خواهیم کرد که به ما در تنظیم دقیق عملکرد مدل کمک می‌کند [۵].

۱-۵-۴- ارزیابی مدل و پیش‌بینی داده‌هایی که تاکنون با مدل دیده نشده

بعد از اینکه فرایند یادگیری مدل با مجموعه داده آموزش انجام شد و هایپرپارامترهای آن تنظیم شد، مدل درنهایت با مجموعه داده آزمون سنجیده می‌شود تا خطای تعمیم این مدل مشخص شود. اگر نتایج راضی‌کننده بود این مدل می‌تواند برای تصمیم‌گیری در آینده در مورد داده‌هایی که تاکنون ندیده است به کار گرفته شود.

۱-۶- شروع استفاده از پایتون[۶۲] در یادگیری ماشین

پایتون زبانی قدرتمند برای پردازش داده است و توجه بسیاری از برنامه‌نویسان را به خود جلب کرده است. زبانی شیءگرا و سطح بالا. زبانی کامل که از آن می‌توان هم در حوزهٔ تحقیق و هم برای توسعهٔ سیستم‌های نرم‌افزاری استفاده کرد. استفاده از آن به‌علت دستورات ساده آسان است. کتابخانه‌های بسیار ارزشمندی برای کار با الگوریتم‌های یادگیری ماشین در این زبان به شرح زیر فراهم شده:

- **Numpy:** برای اشیایی که به‌صورت آرایهٔ N بعدی تعریف شده‌اند استفاده می‌شود.
- **Pandas:** کتابخانه‌ای برای تحلیل داده است و فریم‌های داده‌ای مختلفی را در بر دارد.
- **Matplotlib:** کتابخانه‌ای برای رسم نمودارها و گراف‌های دوبعدی است.
- **Scikit-learn:** حاوی الگوریتم‌هایی برای تحلیل داده است.
- **Seaborn:** کتابخانه‌ای مبتنی بر `matplotlib` برای رسم گراف‌های آماری جذاب و نمودارها است.

۱-۶-۱- نصب پایتون

- **روش اول:** پایتون را از سایت `python.org` دانلود کنید. سپس به روش زیر نصب کنید:

برای نصب پایتون پس از دانلود، روی فایل (`exe.` برای ویندوز) یا (`pkg.` برای Mac) دو بار کلیک کنید و دستورالعمل‌های موجود در صفحه را دنبال کنید.

برای سیستم‌عامل لینوکس، با استفاده از دستور زیر در `prompt` بررسی کنید که آیا پایتون از قبل نصب شده است یا خیر.

```
$ python --version
```

[62]. Python

اگر پایتون نصب نشده است، Python را نصب کنید. در نسخه‌های Debian مانند Ubuntu، می‌توانید از apt استفاده کنید:

```
$ sudo apt-get install python3
```

حال command prompt را اجرا کنید و در مسیری قرار گیرید که پایتون نصب شده است. دستوری را که در ادامه آمده است، اجرا کنید تا مطمئن شوید پایتون به‌درستی نصب شده است.

```
$ python3 --version
 Python 3.6.2
```

حال می‌توانید هر کدام از کتابخانه‌های موردنیاز را با نصب‌کنندهٔ pip دانلود و نصب کنید.

```
$pip install numpy

$pip install matplotlib

$pip install pandas

$pip install seaborn
```

- **روش دوم:** می‌توانید برای اجرای برنامه‌های پایتون از Anaconda استفاده کنید. Anaconda شامل بسته‌های مختلف موردنیاز یادگیری ماشین مانند numpy و matplotlib و... است. همچنین شامل Jupyter notebook، یک محیط تعاملی برای python است به‌گونه‌ای که می‌توان برنامه را به هر اندازه‌ای که باشد بدون اینکه نیاز به ذخیره‌سازی آن باشد، اجرا کرد. محیط بسیار جذابی برای برنامه‌نویس فراهم کرده که تجربهٔ استفاده از آن را توصیه می‌کنیم. نسخه‌های Anaconda برای سیستم‌های عامل windows، Mac و Linux در سایت رسمی Anaconda برای دانلود در دسترس است. برای اطمینان از نصب پایتون، دستوری را که در ادامه آمده است در command line، در مسیر نصب پایتون، اجرا کنید:

```
$ python
```

در خروجی می‌بینیم:

```
Python 3.6.3 |Anaconda custom (32-bit)| (default,
Oct 13 2017, 14:21:34)

[GCC 7.2.0] on linux
```

حال می‌توان کتابخانه‌های موردنیاز را import کرد و از نسخهٔ آن‌ها اطلاع یافت:

```
import pandas
print (pandas.__version__)
0.22.0
import seaborn
print (seaborn.__version__)
0.8.1
```

فصل دوم

دسته‌بندی داده[1] با پایتون

PYTHON

[1]. Data classification

در این فصل تلاش کرده‌ایم مهم‌ترین و متداول‌ترین الگوریتم‌های دسته‌بندی یا طبقه‌بندی داده را مطرح کنیم. در دسته‌بندی تلاش می‌شود تا مدل یادگیری، داده‌های پیشین را یاد بگیرد و وضعیت نمونه‌های جدید را بر اساس برچسب نمونه‌های پیشین، پیش‌بینی کند و تعیین کند که هر نمونه به کدام دسته متعلق است.

آنچه در این فصل بیان می‌شود:

- k تا نزدیک‌ترین همسایه (knn)
- شبکه‌های عصبی مصنوعی (NN)
- ماشین بردار پشتیبان (SVM)
- بیزین
- درخت تصمیم (DT)

۲-۱- k تا نزدیک‌ترین همسایه[۲] (knn)

الگوریتم knn یکی از ساده‌ترین الگوریتم‌های دسته‌بندی داده است. با وجود این، این الگوریتم در بسیاری از مسائل دسته‌بندی داده موفق عمل کرده است. این الگوریتم، یک الگوریتم تنبل[۳] و یک الگوریتم یادگیری بدون پارامتر[۴] است. به این علت که این الگوریتم یک تابع جداساز را یاد نمی‌گیرد بلکه مجموعه داده آموزش را به خاطر می‌سپارد.

فرض کنیم مجموعه دادهٔ آموزش (TS[۵]) ما شامل نمونه‌های دارای برچسب است. به‌عبارت‌دیگر، برای هر نمونه مشخص است که به کدام دسته متعلق است. می‌خواهیم برچسب نمونهٔ x را که در TS حضور ندارد، پیش‌بینی کنیم. X با تمام نمونه‌های موجود در TS مقایسه می‌شود. معیارهای مختلفی برای مقایسهٔ نمونه‌ها و اندازه‌گیری فاصلهٔ آن‌ها از هم وجود دارد. در این الگوریتم با اندازه‌گیری فاصلهٔ x از تمام نمونه‌ها، k تا نزدیک‌ترین نمونه‌ها

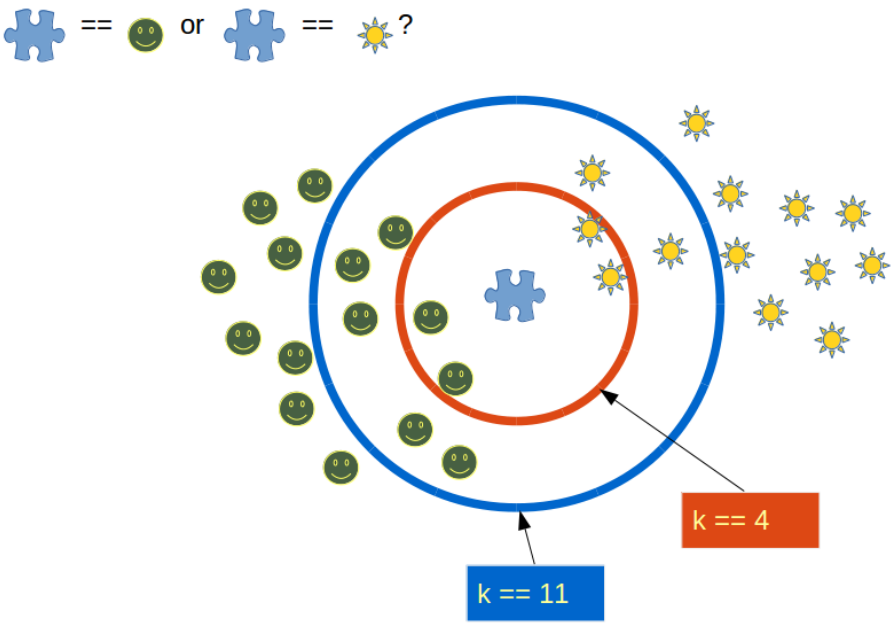
---

2. k-nearest neighbour
3. Lazy learner algorithm
۴. الگوریتم‌های یادگیری ماشین را می‌توان به دو گروه مدل‌های پارامتری و غیرپارامتری تقسیم‌بندی کرد. با استفاده از مدل‌های پارامتری،
پارامترهای مجموعه داده‌های آموزش را برای یادگیری یک تابع تخمین می‌زنیم. این تابع می‌تواند داده‌های جدید را بدون استفادهٔ مجدد از داده‌های آموزش دسته‌بندی کند. نمونه‌های بارز مدل‌های پارامتری، پرسپترون، رگرسیون لجستیک و SVMخطی هستند. در مقابل، مدل‌های غیرپارامتری را نمی‌توان با تعداد ثابتی از پارامترها مشخص کرد و گاهی تعداد پارامترها با داده‌های آموزش رشد می‌کنند. درخت تصمیم و جنگل تصادفی نمونه‌هایی از مدل‌های غیرپارامتری هستند. KNN به زیرمجموعه‌ای از مدل‌های غیرپارامتری به نام یادگیری مبتنی بر نمونه متعلق است. در مدل‌های مبتنی بر نمونه، یادگیری با به خاطر سپردن مجموعه داده‌های آموزش انجام می‌شود. یادگیری تنبل یک مورد خاص از یادگیری مبتنی بر نمونه است که بدون هیچ (صفر) تابع هزینه‌ای در طول یادگیری همراه است۵. Raschka, S., *Python machine learning*. 2015: Packt Publishing Ltd.
5. Train set

به x مشخص می‌شود. K عدد ثابت مثبتی است که کاربر قبل از اجرای برنامه مشخص می‌کند که معمولاً عدد کوچکی است. دسته‌ای که شامل بیشترین تعداد از این k نمونه است به‌عنوان دستهٔ نمونهٔ x در نظر گرفته می‌شود. اگر k=1، در این صورت x به‌سادگی به دستهٔ نزدیک‌ترین نمونه نگاشت می‌شود. شکل ۲-۱ به‌خوبی گویای این مطلب است.

قبل از اینکه به کدنویسی این الگوریتم بپردازیم، مجموعه دادهٔ استفاده شده را معرفی می‌کنیم. مجموعه دادهٔ استاندارد "iris" به دفعات زیادی در برنامه‌های مختلفی استفاده شده است. این مجموعه داده شامل ۵۰ نمونه از سه گونه مختلف از گیاه زنبق (iris) است:

- Iris setosa
- Iris virginica
- Iris versicolor



شکل ۲-۱: چگونگی دسته‌بندی الگوریتم knn با مقدارهای متفاوت k [۱۴].

چهار ویژگی برای نمونه‌های این مجموعه داده اندازه‌گیری شده است. این ویژگی‌ها عبارت است از طول و عرض کاسبرگ[۶] و گلبرگ[۷] نمونه‌ها.

---

[6]. sepal
[7]. petal

✓ **مثال:**

در ابتدا کتابخانهٔ numpy را فراخوانی می‌کنیم و برای استفادهٔ راحت‌تر نام np را برای آن در نظر می‌گیریم. سپس از ماژول sklearn مجموعه داده‌های آن را وارد برنامه می‌کنیم. از بین مجموعه داده‌ها، مجموعه دادهٔ iris را برای استفاده لود می‌کنیم. نمونه‌ها و برچسب‌های این مجموعه داده را هر کدام جداگانه ذخیره‌سازی می‌کنیم [۱۴]:

```
import numpy as np
from sklearn import datasets
iris = datasets.load_iris()
iris_data = iris.data
iris_labels = iris.target
```

از کتابخانهٔ np و کلاس random، متد seed فراخوانی می‌شود. معمولاً در برنامه‌ها قبل از تولید عدد تصادفی، این متد با ارسال عددی صحیح به آن فراخوانی می‌شود. این عدد مقدار اولیه‌ای است که برای تولید عدد تصادفی در کلاس random از آن استفاده می‌شود و هر بار دیگر که برنامه اجرا شود همان عدد تصادفی تولید می‌شود. در اینجا این عدد ۴۲ است. سپس جایگشتی تصادفی از اندیس‌های نمونه‌ها تولید می‌کنیم. سپس همهٔ نمونه‌های مجموعه دادهٔ iris به‌جز ۱۲ تای آخر را به‌عنوان مجموعه دادهٔ آموزش و آنچه باقی می‌ماند را به‌عنوان مجموعه دادهٔ آزمون انتخاب می‌کنیم.

```
np.random.seed(42)
indices = np.random.permutation(len(iris_data))
n_training_samples = 12
learnset_data = iris_data[indices[:-
n_training_samples]]
learnset_labels = iris_labels[indices[:-
n_training_samples]]
testset_data = iris_data[indices[-
n_training_samples:]]
testset_labels = iris_labels[indices[-
n_training_samples:]]
```

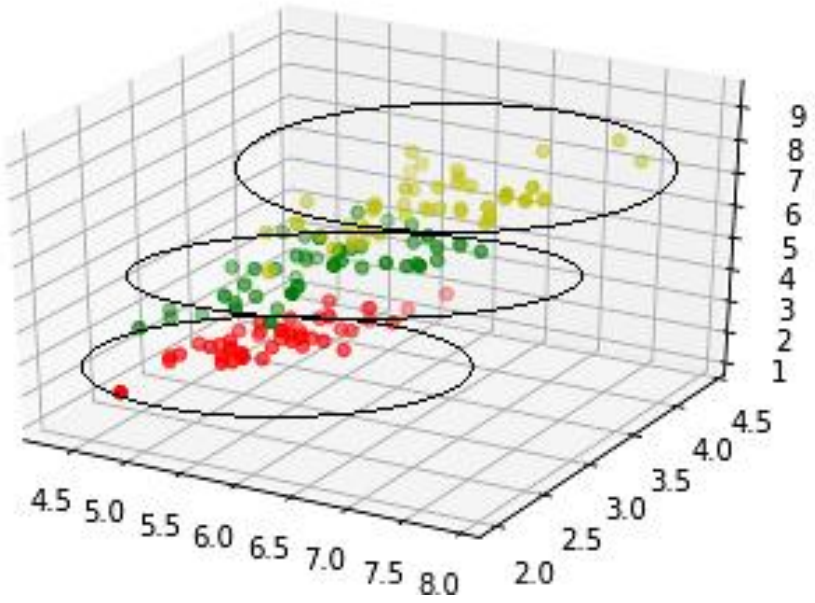
حال برای نمایش مجموعه دادهٔ آموزش، کتابخانهٔ matplotlib و Axes3D را فراخوانی می‌کنیم. X را به‌صورت لیست تعریف می‌کنیم. سپس به آن لیستی از لیست‌ها را الحاق می‌کنیم. در دو حلقهٔ for به‌ازای هر دسته بررسی می‌کنیم که آیا شمارهٔ این دسته با برچسب هر نمونه از مجموعه داده یکسان است یا خیر. درصورتی‌که یکسان باشد، نمونهٔ موردنظر به‌صورت لیست، به لیست X در ردیف مربوط به آن دسته الحاق می‌شود. سپس با مشخص کردن سه رنگ قرمز و سبز و زرد در colors به‌صورت یک tuple و استفاده از متد scatter نمونه‌های هر کلاس را نشان می‌دهیم (شکل ۲-۲) (به علت سطوح خاکستری تصاویر کتاب، نمونه های هر دسته را با خطوط منحنی جدا کرده ایم):

در فضای سه‌بعدی، ابعاد (ویژگی‌های) سوم و چهارم مجموعه داده را با هم جمع می‌کنیم:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
X = []
for iclass in range(3):
    X.append([[], [], []])
    for i in range(len(learnset_data)):
        if learnset_labels[i] == iclass:

X[iclass][0].append(learnset_data[i][0])

X[iclass][1].append(learnset_data[i][1])

X[iclass][2].append(sum(learnset_data[i][2:]))
colours = ("r", "g", "y")
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for iclass in range(3):
      ax.scatter(X[iclass][0], X[iclass][1],
                 X[iclass][2], c=colours[iclass])
plt.show()
```

حال برای تعیین فاصلهٔ بین دو نمونه از یک تابع برای محاسبهٔ فاصله استفاده می‌کنیم. در این مثال از تابع فاصلهٔ اقلیدسی استفاده می‌کنیم:

```
def distance(instance1, instance2):
# just in case, if the instances are lists ortuples:
    instance1 = np.array(instance1)
    instance2 = np.array(instance2)
    return np.linalg.norm(instance1 - instance2)
```



شکل ۲-۲: نمایش نمونه‌های هر دسته

تابع `get_neighbors` لیستی از `k` تا نزدیک‌ترین نمونه به نمونهٔ آزمون را برمی‌گرداند. به‌ازای هر نمونه از مجموعه داده، فاصلهٔ آن تا نمونهٔ آزمون سنجیده می‌شود و این فاصله به همراه نمونهٔ متناظر و برچسب آن نمونه به‌صورت `tuple` به لیست `distances` الحاق می‌شود. ازآنجایی‌که مقدار فاصله به‌عنوان عنصر دوم هر `tuple` ذخیره می‌شود، در متد `sort` با تعریف تابع `lambda` مشخص می‌کنیم که مرتب کردن لیست `x` (`distances`) بر اساس عنصر دوم هر `tuple` (مقدار فاصله) باشد:

```
def get_neighbors(training_set,
                  labels,
                  test_instance,
```

```
                k,
                distance=distance):
    """
     get_neighors calculates a list of the k
     nearest neighbors of an instance
     'test_instance'.
     The list neighbors contains 3-tuples with
     (index, dist, label) where
     index    is the index from the training_set,
     dist    is the distance between the
     test_instance and the instance
     training_set[index]
     distance is a reference to a function used to
     calculate the distances
    """
    distances = []
    for index in range(len(training_set)):
        dist = distance(test_instance,
        training_set[index])
        distances.append((training_set[index],
        dist, labels[index]))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    return neighbors
```

حال تابع get_neighbors را برای نمونه‌های مجموعه دادهٔ iris امتحان می‌کنیم:

```
for i in range(5):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              3,
```

```
                                    distance=distance)
        print(i,
              testset_data[i],
              testset_labels[i],
              neighbors)
```

و آنچه به‌عنوان خروجی خواهیم داشت:

```
0 [5.7 2.8 4.1 1.3] 1 [(array([5.7, 2.9, 4.2,
1.3]), 0.14142135623730995, 1), (array([5.6, 2.7,
4.2, 1.3]), 0.17320508075688815, 1), (array([5.6,
3., 4.1, 1.3]), 0.22360679774997935, 1)]

1 [6.5 3.  5.5 1.8] 2 [(array([6.4, 3.1, 5.5,
1.8]), 0.1414213562373093, 2), (array([6.3, 2.9,
5.6, 1.8]), 0.24494897427831783, 2), (array([6.5,
3., 5.2, 2. ]), 0.3605551275463988, 2)]

2 [6.3 2.3 4.4 1.3] 1 [(array([6.2, 2.2, 4.5,
1.5]), 0.2645751311064586, 1), (array([6.3, 2.5,
4.9, 1.5]), 0.574456264653803, 1), (array([6., 2.2,
4., 1. ]), 0.5916079783099617, 1)]

3 [6.4 2.9 4.3 1.3] 1 [(array([6.2, 2.9, 4.3,
1.3]), 0.20000000000000018, 1), (array([6.6, 3.,
4.4, 1.4]), 0.2645751311064587, 1), (array([6.6,
2.9, 4.6, 1.3]), 0.3605551275463984, 1)]

4 [5.6 2.8 4.9 2. ] 2 [(array([5.8, 2.7, 5.1,
1.9]), 0.3162277660168375, 2), (array([5.8, 2.7,
5.1, 1.9]), 0.3162277660168375, 2), (array([5.7,
2.5, 5., 2. ]), 0.33166247903553986, 2)]
```

حال تابعی می‌نویسیم تا کلاسی را مشخص کند که بیشترین تعداد از k تا همسایه به آن تعلق دارند. برای این کار از کلاس Counter، یک شیء به نام class-counter می‌سازیم تا با در نظر گرفتن عنصر سوم لیست neighbors، که نشان‌دهندهٔ برچسب هر همسایه است، تعداد نمونه‌های متعلق به هر دسته را بشمارد. سپس متد most_common(1) دسته‌ای که بیشترین نمونه را دارد به همراه تعداد نمونهٔ متعلق به آن دسته را در یک لیست از tuple دوتایی برمی‌گرداند. most_common(1)[0][0]، عنصر اول از tuple اول را برمی‌گرداند که همان شماره دسته‌ای است که بیشترین نمونه را در بر دارد.

```
from collections import Counter
def vote(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    return class_counter.most_common(1)[0][0]
```

سپس به‌ازای تمام نمونه‌های مجموعه دادهٔ آزمون در یک حلقهٔ for، تابع get_neighbors را فراخوانی می‌کنیم. بعد از مشخص شدن همسایه‌های هر نمونه با فراخوانی تابع vote برچسب نمونهٔ مورد بررسی را مشخص می‌کنیم:

```
for i in range(n_training_samples):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              3,
                              distance=distance)
    print("index: ", i,
          ", result of vote: ", vote(neighbors),
          ", label: ", testset_labels[i],
          ", data: ", testset_data[i])
```

در هر خط از خروجی به ترتیب شمارهٔ نمونهٔ آزمون، شمارهٔ دسته‌ای که با الگوریتم knn برای نمونه در نظر گرفته شد، برچسب نمونه و خود نمونه را خواهیم داشت:

```
index:  0, result of vote:  1, label:  1, data:
[5.7 2.8 4.1 1.3]
index:  1, result of vote:  2, label:  2, data:
[6.5 3.  5.5 1.8]
index:  2, result of vote:  1, label:  1, data:
[6.3 2.3 4.4 1.3]
```

```
index:  3, result of vote:  1, label:  1, data:
[6.4 2.9 4.3 1.3]
index:  4, result of vote:  2, label:  2, data:
[5.6 2.8 4.9 2. ]
index:  5, result of vote:  2, label:  2, data:
[5.9 3.  5.1 1.8]
index:  6, result of vote:  0, label:  0, data:
[5.4 3.4 1.7 0.2]
index:  7, result of vote:  1, label:  1, data:
[6.1 2.8 4.  1.3]
index:  8, result of vote:  1, label:  2, data:
[4.9 2.5 4.5 1.7]
index:  9, result of vote:  0, label:  0, data:
[5.8 4.  1.2 0.2]
index:  10, result of vote:  1, label:  1, data:
[5.8 2.6 4.  1.2]
index:  11, result of vote:  2, label:  2, data:
[7.1 3.  5.9 2.1]
```

همان‌طور که در خروجی مشاهده می‌کنید به‌جز اندیس شمارهٔ ۸ بقیهٔ نمونه‌ها به‌درستی دسته‌بندی شده‌اند.

نکته: قانون کلی برای تعیین k وجود ندارد و این مقدار کاملاً به داده بستگی دارد. اما می‌توان به‌صورت کلی گفت که با افزایش k تأثیر نویز در تصمیم‌گیری کاهش می‌یابد اگرچه تفکیک‌پذیری مرزهای دسته‌ها کمتر می‌شود.

۲-۱-۱- استفاده از ماژول sklearn برای دسته‌بند knn

کتابخانهٔ neighbors از ماژول sklearn امکان استفاده از دو نوع مختلف از دسته‌بندهای knn را به ما می‌دهد.

- **KNeighborsClassifiers:** بر اساس k تا نزدیک‌ترین همسایهٔ یک نمونه، آن نمونه را طبقه‌بندی یا دسته‌بندی می‌کند. k یک عدد صحیح است که کاربر آن را تعیین می‌کند.

- **RadiusNeighborsClassifier:** بر اساس تعداد همسایه‌های نمونهٔ مورد بررسی در شعاع ثابت r، آن نمونه را دسته‌بندی می‌کند. r یک عدد اعشاری است که کاربر آن را تعیین می‌کند.

قبل از اینکه با اجرای یک برنامه از این کتابخانهٔ قدرتمند استفاده کنیم، کمی با پارامترهای آن آشنا می‌شویم:

- **پارامترها**

✓ **n_neighbors:** عددی صحیح است و برای تعیین تعداد همسایه‌ها (k) استفاده می‌شود. به‌صورت پیش‌فرض برابر 5 است و مقداردهی به آن اختیاری است.

✓ **Weights:** تصمیم‌گیری بر اساس نزدیک‌ترین همسایه می‌تواند بر اساس وزن‌دار کردن نمونه‌ها انجام شود. در این صورت می‌توان به نمونه‌هایی که نزدیک‌تر هستند نسبت به نمونه‌هایی که دورتر هستند وزن بیشتری در تصمیم‌گیری داد. وزن‌دار کردن نمونه‌ها در کتابخانهٔ neighbors با کلمه کلیدی weights صورت می‌گیرد:

- **Weights='uniform':** در این صورت وزن همهٔ نمونه‌ها یکسان است که این همان مقدار پیش‌فرض است.
- **Weights='distance':** وزنی است که بر اساس معکوس فاصلهٔ هر نمونه تا نمونهٔ مورد آزمون محاسبه می‌شود.
- **قابل‌فراخوانی:** این امکان وجود دارد که کاربر تابعی را برای محاسبهٔ وزن تعریف کند. این تابع می‌تواند آرایه‌ای شامل فاصله‌های نمونه‌ها از نمونهٔ مورد آزمون را بگیرد و آرایهٔ وزن‌ها را برگرداند.

✓ **الگوریتم:** الگوریتم‌هایی که به‌دلخواه می‌توان برای یافتن نزدیک‌ترین همسایه‌ها استفاده کرد:

✓ **'bal_tree':** از کلاس 'BallTree' استفاده می‌کند.

✓ **'kd_tree':** از کلاس 'KDTree' استفاده می‌کند.

✓ **'brute':** از جست‌وجوی brute force استفاده می‌کند.

✓ **'auto':** تلاش می‌کند مناسب‌ترین الگوریتم را انتخاب کند. به‌عنوان‌مثال درصورتی‌که ماتریس داده (مجموعه دادهٔ آموزش) به‌صورت پراکنده[8] باشد از جست‌وجوی brute force استفاده می‌کند.

✓ **Leaf_size:** عددی صحیح است. مقدار پیش‌فرض آن 30 است و مقداردهی به این پارامتر اختیاری است. این مقدار برای الگوریتم‌های BallTree یا KDTree ارسال می‌شود. مقدار

[8]. sparse

این پارامتر روی سرعت اجرای برنامه تأثیرگذار است و مقدار بهینهٔ آن به چگونگی مسئله مربوط است.

- ✓ **P:** عددی صحیح است و مقداردهی به آن اختیاری است. مقدار پیش‌فرض آن 2 است. پارامتر توان برای معیار اندازه‌گیری فاصلهٔ Minkowski است. درصورتی‌که مقدار آن 1 باشد، فاصلهٔ منهتن[9] محاسبه می‌شود. درصورتی‌که مقدار 2 باشد، فاصلهٔ اقلیدسی[10] محاسبه می‌شود. برای p دلخواه فاصلهٔ Minkowski (lp) محاسبه می‌شود.
- ✓ **Metric:** از نوع رشته است یا قابل فراخوانی است. مقدار پیش‌فرض آن Minkowski است. در درخت برای محاسبهٔ فاصله استفاده می‌شود.
- ✓ **Metric_params:** از نوع دیکشنری است و مقدار پیش‌فرض آن None است. برای آرگومان‌های کلیدی برای تابع metric تعبیه شده است.
- ✓ **N_jobs:** عددی صحیح است و مقدار پیش‌فرض آن 1 است. تعیین‌کنندهٔ تعداد وظایف موازی برای انجام عمل جست‌وجوی همسایه‌ها است. اگر مقدار آن 1- باشد در این صورت تعداد وظایف برابر است با تعداد هسته‌های cpu.

حال می‌خواهیم از کلاس KNeighborsClassifier از کتابخانهٔ sklearn.neighbors روی مجموعه دادهٔ iris استفاده کنیم [۱۴]:

```
# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(learnset_data, learnset_labels)
KNeighborsClassifier(algorithm='auto',
                     leaf_size=30,
                     metric='minkowski',
                     metric_params=None,
                     n_jobs=1,
                     n_neighbors=5,
                     p=2,
                     weights='uniform')
```

9. Manhattan(l1)
10. Euclidean(l2)

```
print("Predictions form the classifier:")
print(knn.predict(testset_data))
print("Target values:")
print(testset_labels)
```

در خروجی خواهیم داشت:

```
Predictions form the classifier:
[1 2 1 1 2 2 0 1 1 0 1 2]
Target values:
[1 2 1 1 2 2 0 1 2 0 1 2]
```

۲-۲- شبکه‌های عصبی مصنوعی[11]

شبکه‌های عصبی مصنوعی همان‌طور که از نام آن مشخص است الهام گرفته از سیستم یادگیری طبیعی مغز است. تخمین زده می‌شود که مغز انسان شامل شبکهٔ به‌هم‌پیوسته و فشرده‌ای از 10^{11} نرون است که هر کدام به‌صورت متوسط به 10^{4} نرون متصل است [۳]. هر نرون شامل سه بخش اصلی بدنه سلول،[12] دندریت‌ها[13] و آکسون[14] است. شاخه‌های دندریت بسان شاخه‌های درخت از بدنهٔ سلول خارج شده‌اند و سیگنال‌ها را در سیناپس‌ها از سایر نرون‌ها دریافت می‌کنند. آکسون برای ارسال خروجی نرون به سیناپس‌های سایر نرون‌ها استفاده می‌شود (شکل ۲-۳).

۲-۲-۱- شبکهٔ عصبی پرسپترون[15] تک لایه

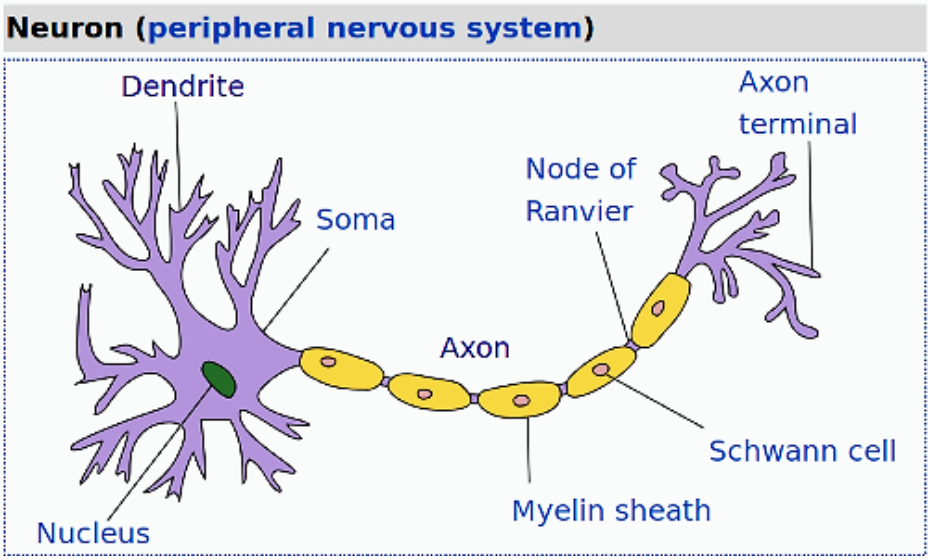
یک واحد ادراک‌کننده یا پرسپترون از شبکه‌های عصبی مصنوعی، به‌صورت نشان داده شده در شکل ۲-۴، یک نرون زیستی را شبیه‌سازی می‌کند. روند یادگیری یک پرسپترون بسیار ساده است و شامل مراحل زیر است:

- بردار وزن مقداردهی اولیه می‌شود. این مقدار می‌تواند یک مقدار کوچک تصادفی باشد.
- به‌ازای هر نمونه:
  - ✓ خروجی پرسپترون محاسبه می‌شود.
  - ✓ بردار وزن به‌روز می‌شود.

---

11. Artificial Neural Networks
12. Soma
13. Dendrites
14. Axon
15. Perceptron

در بدنهٔ یک نرون یا پرسپترون سیگنال‌های ورودی در وزن‌های متناظرشان ضرب می‌شوند (وزن‌ها در فاز یادگیری نرون تنظیم می‌شوند). تأثیر هر ورودی در شبکهٔ عصبی با وزن آن مشخص می‌شود. هر چقدر وزن آن بیشتر باشد تأثیر آن نمونه در شبکه بیشتر است. در ادامه سیگنال‌های ورودی تغییر یافتهٔ وزن‌ها، با هم جمع می‌شوند. البته این امکان وجود دارد که مقداری به نام bias را به این جمع اضافه کنیم که می‌تواند در طول یادگیری تنظیم شود. درنهایت تابع فعال‌سازی[۱۶] به حاصل این جمع وزن‌دار اعمال می‌شود. بایاس می‌تواند خروجی تابع فعال‌سازی را کمی به راست یا چپ حرکت دهد که این می‌تواند برای یک یادگیری موفق حیاتی باشد [۱۵].



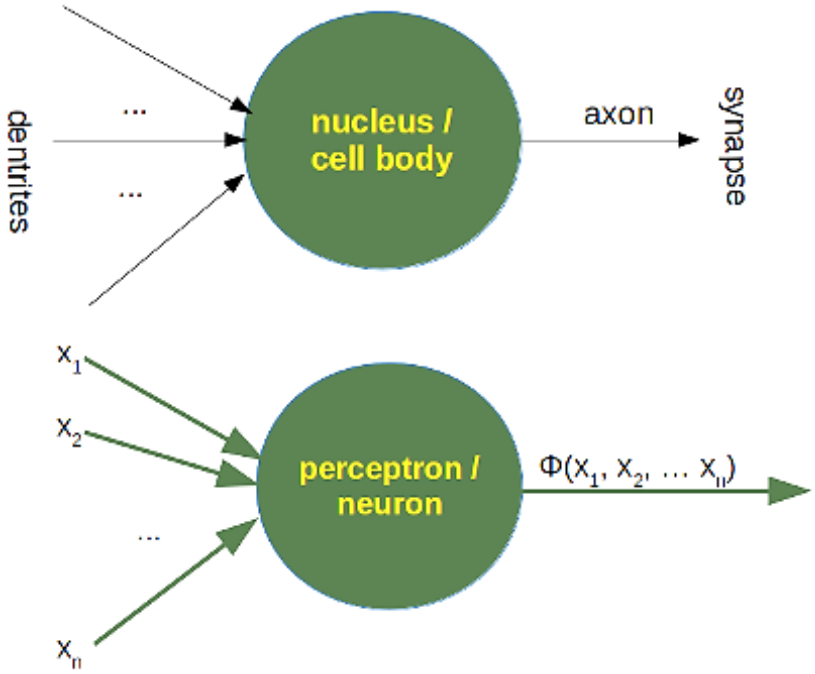
شکل ۲-۳: ساختار یک نرون زیستی [۱۶].

ساده‌ترین صورت این تابع فعال‌سازی عبارت است از:

$$\Psi(x) = \begin{cases} 1 & wx + b > t \\ . & otherwise \end{cases} \qquad (1)$$

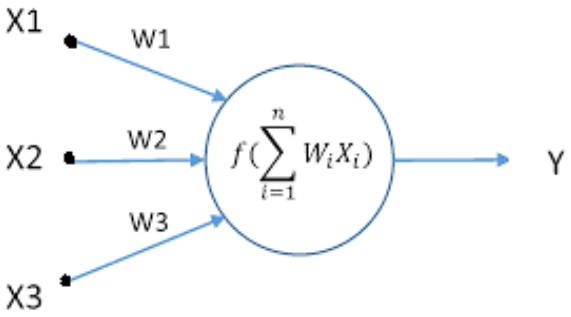
x بردار ورودی شبکه است (تعداد عناصر این بردار برابر است با تعداد ویژگی‌های هر نمونه) و بردار w، بردار وزن متناظر آن است. b میزان بایاس شبکه را نشان می‌دهد. همان‌طور که در عبارت (۱) مشخص است درصورتی‌که $wx + b$ از یک مقدار ازپیش‌تعیین‌شده‌ای بیشتر باشد، خروجی نرون 1 خواهد بود (معادل آتش گرفتن[۱۷] در نرون زیستی) در غیر این صورت خروجی نرون 0 است.

16. Activation function
17. firing



شکل ۲-۴: شبیه‌سازی نرون زیستی با یک پرسپترون در شبکه‌های عصبی مصنوعی [۱۴].

شکل ۲-۵ شکل ساده‌ای از ساختار یک شبکهٔ عصبی مصنوعی پرسپترون تک لایه را نشان می‌دهد که ساده‌ترین شکل از شبکه‌های عصبی پیشخور[۱۸] است [۱۷]:



شکل ۲-۵: ساختار یک شبکهٔ عصبی سادهٔ پرسپترون با یک نرون

[18]. Feedforward: تفاوت شبکه‌های پسخور با شبکه‌های پیشخور این است که در شبکه‌های پسخور یک سیگنال برگشتی از یک نرون به همان نرون یا نرون‌های همان لایه یا لایهٔ قبل وجود دارد.

بعد از محاسبهٔ خروجی نرون، به‌روزرسانی وزن‌ها انجام می‌شود:

ابتدا میزان خطای خروجی پرسپترون محاسبه می‌شود:

$$\text{Error} = \Psi(x) - target\ value \quad (۲)$$

$\Psi(x)$ برچسب نمونه است که با پرسپترون محاسبه شده است. $target\ value$ برچسب نمونه است که در مجموعه داده آموزش داشته‌ایم.

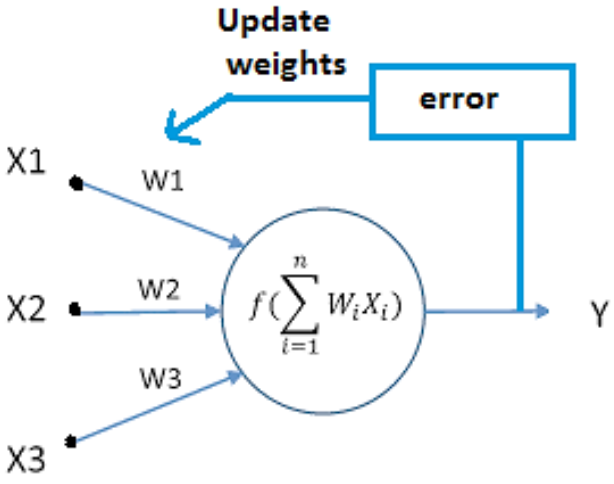
برای هر ویژگی از نمونهٔ x داریم:

$$w_j = w_j + \Delta w_j \quad (۳)$$

$$\Delta w_j = \alpha * \text{error} * x_j \quad (۴)$$

با به دست آوردن میزان خطا و ضرب آن در نرخ یادگیری α و ویژگی jام از نمونهٔ x (x_j)، میزان Δw_j محاسبه می‌شود و این مقدار به w_j اضافه می‌شود. در ادامه، نمونهٔ بعدی که قرار است برچسب آن با پرسپترون محاسبه شود، با بردار وزن جدید محاسبه انجام می‌شود و بعد از به دست آمدن خروجی مجدداً بردار وزن به‌روز می‌شود. این روند ادامه می‌یابد تا تمام نمونه‌های مجموعه دادهٔ آموزش به سیستم یادگیرندهٔ آموزش داده شوند. البته این امکان هم وجود دارد که تعدادی از نمونه‌ها به سیستم یادگیرنده داده شود سپس بردار وزن به‌روز شود. این مدل را یادگیری شبکهٔ پرسپترون به‌صورت دسته‌ای[19] گویند.

شکل ۲-۶ ساختار کامل‌تری از سیستم یادگیری یک پرسپترون را نشان می‌دهد.



شکل ۲-۶: شکل کامل‌تری از سیستم یادگیری پرسپترون [۵]

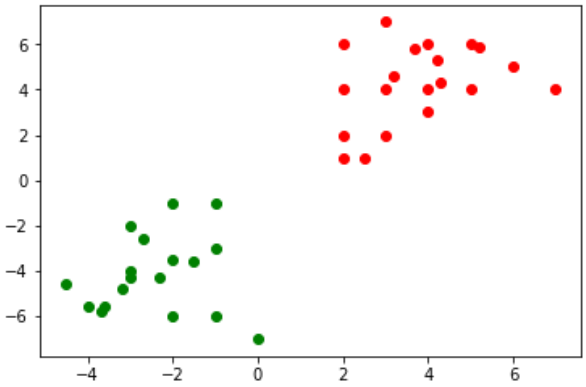
[19]. batch

حال در برنامه‌ای که در ادامه خواهید دید می‌خواهیم یک شبکهٔ عصبی را برای دسته‌بندی دوطبقه از داده‌ها در یک فضای دوبعدی آموزش دهیم:

ابتدا تعدادی نقاط برای هر دو دسته به‌عنوان مجموعه دادهٔ آموزش تولید می‌کنیم. سپس با استفاده از کلاس `zip` دو `tuple` به دست می‌آوریم که `tuple` اول مختصات `x` نقاط است و `tuple` دوم حاوی مختصات `y` نقاط است. سپس با استفاده از کلاس `pyplot` و متد `scatter` نقاط دو کلاس را به تفکیک رنگ رسم می‌کنیم:

```
from matplotlib import pyplot as plt
class1 = [(2, 4),(2, 6),(2, 1),(2.5, 1),(3, 4), (4, 4),
(5, 6),(2, 2),(3, 2),(4.2, 5.3), (4, 3), (6, 5), (4, 6),
(3.7, 5.8),(3.2, 4.6), (5.2, 5.9), (5, 4), (7, 4), (3,
7), (4.3, 4.3)]
class2 = [(0, -7),(-1, -1),(-1, -3),(-2, -6), (-2, -1),
(-3, -2),(-3, -4), (-2, -3.5), (-1, -6), (-3, -4.3), (-
4, -5.6), (-3.2, -4.8), (-2.3, -4.3), (-2.7, -2.6), (-
1.5, -3.6), (-3.6, -5.6), (-4.5, -4.6), (-3.7, -5.8)]
X,Y=zip(*class1)
plt.scatter(X,Y,c="r")
X,Y=zip(*class2)
plt.scatter(X,Y,c="g")
```

در خروجی خواهیم داشت:



شکل ۲-۷: نقاط مربوط به دو کلاس که قرار است دسته‌بندی شوند.

کتابخانهٔ numpy را برای تولید عدد تصادفی فراخوانی می‌کنیم و آن را به‌اختصار np می‌نامیم. از کتابخانهٔ collection کلاس Counter را فراخوانی می‌کنیم تا تعداد نمونه‌های اختصاص داده شده به هر دسته را در انتهای برنامه بشماریم. حال کلاس Perceptron را تعریف می‌کنیم. در تابع مربوط به مقداردهی اولیه (init) بر اساس ابعاد مجموعه داده بردار وزن را تعریف می‌کنیم، زیرا همان‌طور که در شکل ۲-۵ مشخص است به‌ازای هر بعد داده وزنی به آن متناظر می‌شود. سپس نرخ یادگیری را نیز مقداردهی می‌کنیم. در ادامه تابع فعال‌سازی perceptron یا نرون را تعریف می‌کنیم. در تابع call عملیات ضرب داده در وزن‌ها و سپس جمع آن‌ها و فراخوانی تابع فعال‌سازی انجام می‌شود. در تابع adjust برچسب محاسبه شده برای هر نقطه از مقدار واقعی آن (هدف) کم می‌شود و به‌عنوان خطای آن نقطه در نظر گرفته می‌شود. سپس این میزان خطا در آن نقطه ضرب می‌شود. در ادامه حاصل آن در نرخ یادگیری ضرب می‌شود. آنچه به دست می‌آید (correction) به وزن متناظر با نقطهٔ موردنظر افزوده می‌شود:

```
import numpy as np
from collections import Counter

class Perceptron:

    def __init__(self, input_length, weights=None):
        if weights==None:
            self.weights =
            np.random.random((input_length)) * 2 - 1
        self.learning_rate = 0.1

    @staticmethod
    def unit_step_function(x):
        if x < 0:
            return 0
        return 1

    def __call__(self, in_data):
        weighted_input = self.weights * in_data
```

```
        weighted_sum = weighted_input.sum()
        return
Perceptron.unit_step_function(weighted_sum)


    def adjust(self,
               target_result,
               calculated_result,
               in_data):
        error = target_result - calculated_result
        for i in range(len(in_data)):
            correction = error * in_data[i]
            *self.learning_rate
            self.weights[i] += correction
```

اکنون از کلاس Perceptron یک شیء به نام p می‌سازیم. مجموعه داده به‌صورت دوبعدی است. این عدد را به‌عنوان پارامتر هنگام ساختن شیء از کلاس Perceptron به آن ارسال می‌کنیم. برای هر نمونه در دستهٔ 1 متد adjust از شیء p را فراخوانی می‌کنیم. هنگام فراخوانی متد، برچسب نمونه، برچسب محاسبه شده و نمونه را به متد ارسال می‌کنیم تا عملیات یادگیری در متد adjust انجام شود. همین کار را برای هر نمونه در دستهٔ 2 انجام می‌دهیم. از کلاس Counter یک شیء به نام evaluation می‌سازیم. سپس با استفاده از شیء chain در یک حلقهٔ for ابتدا در نمونه‌های دستهٔ 1 سپس در نمونه‌های دستهٔ 2 iteration انجام می‌دهیم؛ برای هر نمونه اگر برچسب محاسبه شده 1 باشد، به تعداد نمونه‌های کلاس 1 یک واحد افزوده می‌شود و در غیر این صورت به تعداد نمونه‌های کلاس 2 یک واحد اضافه می‌شود. در ادامه برچسب دو نمونهٔ دلخواه به‌عنوان نمونه‌های آزمون با شیء p مشخص می‌شود. در ادامه با متد most_common() تعداد عناصر هر دسته به همراه شمارهٔ دسته به ترتیب نزولی چاپ می‌شود.

```
from itertools import chain
p = Perceptron(2)
for point in class1:
  p.adjust(1,
      p(point),
      point)
```

```
for point in class2:
  p.adjust(0,
      p(point),
      point)

evaluation = Counter()
for point in chain(class1, class2):
  if p(point) == 1:
    evaluation["class1"] += 1
  else:
    evaluation["class2"] += 1

testpoints = [(3.9, 6.9), (-2.9, -5.9)]
for point in testpoints:
  print(p(point))

print(evaluation.most_common())
```

در خروجی خواهیم داشت:

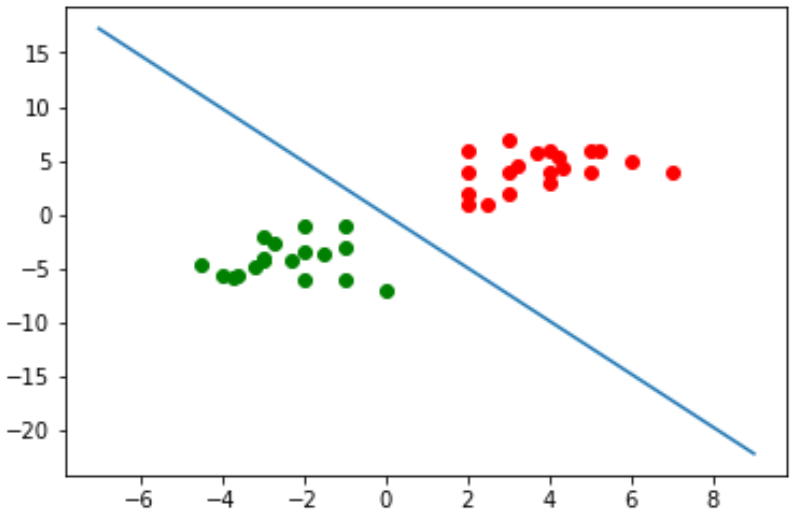
```
1
0
[('class1', 20), ('class2', 18)]
```

اکنون می‌خواهیم خط تفکیک‌کنندهٔ دو دسته را که با شبکهٔ عصبی به دست آمد رسم کنیم:

```
from matplotlib import pyplot as plt
X, Y = zip(*class1)
plt.scatter(X, Y, c="r")
X, Y = zip(*class2)
plt.scatter(X, Y, c="g")
```

```
x = np.arange(-7, 10)
y = 5*x + 10
m = -p.weights[0] / p.weights[1]
plt.plot(x, m*x)
plt.show()
```

در خروجی خواهیم داشت:



شکل ۲-۸: خط تفکیک‌کنندهٔ دو کلاس که با شبکهٔ عصبی تک‌لایه به دست آمد

مشکل اساسی شبکه‌های تک‌لایه این است که تنها توانایی حل آن دسته از مسائل طبقه‌بندی را دارند که به‌طور خطی از هم مستقل‌اند [۱۷]. ازاین‌رو شبکه‌های چندلایه مطرح شد:

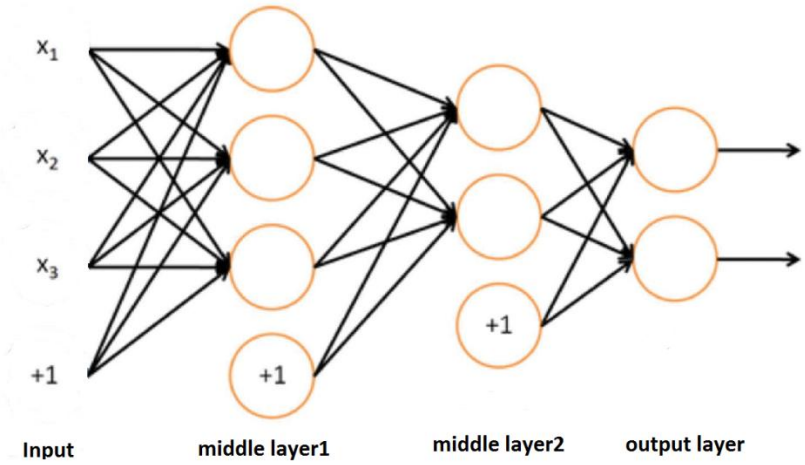
۲-۲-۲- شبکهٔ عصبی پرسپترون چندلایه (MLP[۲۰])

شبکه‌های عصبی پرسپترون، به‌ویژه پرسپترون چندلایه، در زمرهٔ کاربردی‌ترین شبکه‌های عصبی هستند. این شبکه‌ها قادرند با انتخاب مناسب تعداد لایه‌ها و سلول‌های عصبی، که اغلب زیاد هم نیستند، یک نگاشت غیرخطی را با دقت دلخواه انجام دهند. این همان چیزی است که در بسیاری از مسائل فنی‌مهندسی به‌عنوان راه‌حل اصلی مطرح است [۱۷]. در این شبکه‌ها هر نرون به تمامی نرون‌های لایهٔ قبل متصل است. به چنین شبکه‌هایی، شبکه‌های کاملاً مرتبط می‌گویند [۱۷]. یک شبکهٔ پرسپترون سه‌لایه شامل دو لایهٔ میانی و یک لایهٔ خروجی است. خروجی‌های لایهٔ اول، بردار ورودی لایهٔ دوم را تشکیل می‌دهند و به همین ترتیب بردار خروجی

[20]. Multi-layer Perceptron

لایهٔ دوم، ورودی‌های لایهٔ سوم را می‌سازند و خروجی‌های لایهٔ سوم، پاسخ نهایی شبکه را تشکیل می‌دهند. به عبارت روشن‌تر، روند جریان سیگنالی در شبکه، در یک مسیر پیشخور صورت می‌گیرد (از چپ به راست از لایه‌ای به لایهٔ دیگر) [۱۷].

شکل ۲-۹، شکل کلی از یک شبکهٔ عصبی پرسپترون سه‌لایه را نشان می‌دهد:



شکل ۲-۹: شکل کلی شبکهٔ عصبی پرسپترون سه‌لایه (ورودی به‌عنوان یک لایه تلقی نمی‌شود [۱۷])

شبکهٔ MLP همان‌طور که عنوان شد یک شبکهٔ عصبی پیشخور است که هر لایه کاملاً به لایهٔ بعدی متصل است. در این شبکه لایه‌های میانی متعددی می‌تواند وجود داشته باشد.

ورودی هر نرون در لایهٔ میانی اول برابر است با:

$$y_j = \sum_{i=1}^{n} w_{ij} x_i \qquad (۵)$$

به‌طوری‌که w_{ij} وزن‌های وارد شده به نرون j ام است و x_i ورودی i ام است. y_j ورودی نرون j ام است. سپس تابع فعال‌سازی روی این ورودی اعمال می‌شود و حاصل در وزن لایهٔ بعد ضرب می‌شود. پس ورودی نرون‌ها در لایهٔ بعد برابر است با:

$$y_k = \sum_{j=1}^{n} f(y_j) w_{jk} \qquad (۶)$$

بعد از پیشنهاد شبکه‌های عصبی چندلایه، به‌علت عدم ارائهٔ قانون یادگیری که بتوان برای تنظیم پارامترهای شبکه به کار برد، توپولوژی شبکهٔ MLP ناقص بود. به‌عبارت‌دیگر، اگرچه آن‌ها توانسته بودند شبکه‌های تک‌لایه را به چندلایه تعمیم بدهند، اما قادر نشدند قانون یادگیری LMS[۲۱] یا SLPR[۲۲] را برای شبکه‌های چندلایه

[21]. Least Mean Square
[22]. Single Layer Perceptron Rule

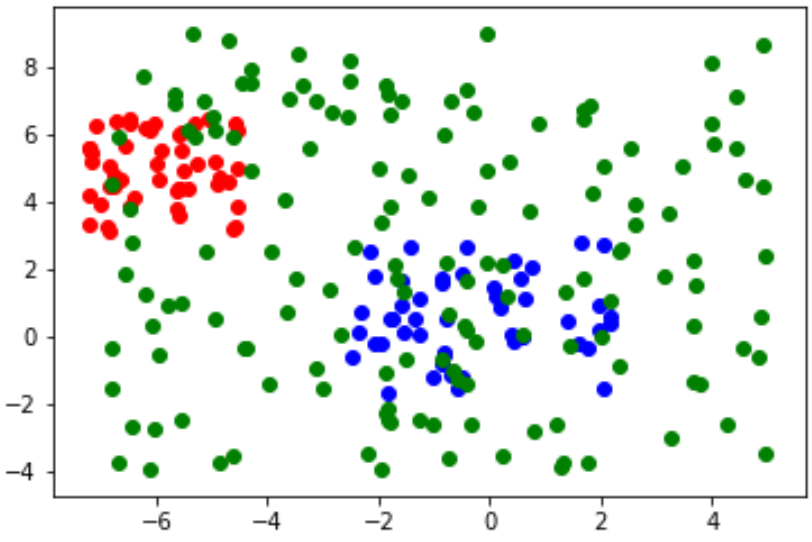
تعمیم بدهند [۱۷]. توسعهٔ الگوریتم «پس انتشار (BP[۲۳])» با فراهم آوردن روشی از نظر محاسباتی کارا، رنسانسی در شبکه‌های عصبی ایجاد کرد، زیرا شبکه‌های MLP با قاعدهٔ آموزش BP همچنان بیشترین کاربرد را در حل مسائل فنی‌مهندسی دارند که در بخش بعدی با این الگوریتم آشنا خواهیم شد.

در این قسمت می‌خواهیم با استفاده از کلاس MLPClassifier از ماژول sklearn.neural_network نقاطی را که در فضای دوبعدی ایجاد می‌کنیم دسته‌بندی کنیم [۱۴]. ابتدا با استفاده از تابع توزیع یکنواخت ۵۰ نقطه را به‌عنوان نقاط دستهٔ اول و ۵۰ نقطه را به‌عنوان نقاط دستهٔ دوم ایجاد می‌کنیم. سپس سه برابر این نقاط را به‌عنوان نقاط مجموعه دادهٔ آزمون تعریف می‌کنیم. شکل ۲-۱۰ به‌خوبی گویای این مطلب است:

```
import numpy as np
import matplotlib.pyplot as plt
npoints=50
X,Y=[],[]
#class0
X.append(np.random.uniform(low=-2.5, high=2.3,
size=(npoints,)))
Y.append(np.random.uniform(low=-1.7, high=2.8,
size=(npoints,)))
#class1
X.append(np.random.uniform(low=-7.2, high=-4.4,
size=(npoints,)))
Y.append(np.random.uniform(low=3, high=6.5,
size=(npoints,)))
learnset=[]
learnlabels=[]
for i in range(2):
    # adding points of class i to learnset
    points=zip(X[i],Y[i])
    for p in points:
```

[23]. Back Propagation

```
            learnset.append(p)
            learnlabels.append(i)
npoints_test=3*npoints
TestX=np.random.uniform(low=-7.2, high=5,
size=(npoints_test,))
TestY=np.random.uniform(low=-4, high=9,
size=(npoints_test,))
test_set=[]
points=zip(TestX,TestY)
for p in points:
    test_set.append(p)
colors=["b","r"]
for i in range(2):
    plt.scatter(X[i],Y[i], c=colors[i])
plt.scatter(TestX,TestY, c="g")
plt.show()
```



شکل ۲-۱۰: مجموعه دادهٔ آموزش و آزمون تولید شده برای آموزش شبکهٔ عصبی MLP

در ادامه به آموزش یک MLPClassifier می‌پردازیم. در اولین پارامتر این کلاس تعداد نرون‌های لایه‌های میانی مشخص می‌شود. i‌امین عدد tuple، معرف تعداد نرون‌های iامین لایهٔ میانی است. پارامتر alpha معرف حد نرم L2 است. در پارامتر solver مشخص می‌کنیم که برای به‌روزرسانی وزن‌ها از چه روشی استفاده می‌کنیم. به‌عنوان‌مثال، درصورتی‌که این پارامتر برابر باشد با 'sgd' از روش «کاهش گرادیان تصادفی»[۲۴] استفاده می‌شود. درصورتی‌که از روش‌های مبتنی بر کاهش گرادیان برای بهینه‌سازی وزن استفاده شود، پارامتر max_iter مشخص کنندهٔ تعداد دفعاتی است که هر نقطه برای آموزش استفاده می‌شود. در صورت استفاده از روش‌های دیگر، این پارامتر معنای دیگری دارد. پارامتر verbose یک پارامتر بولین است که مقدار پیش‌فرض آن false است. درصورتی‌که true شود پیشرفت الگوریتم را در خروجی چاپ می‌کند. پارامتر tol مشخص می‌کند درصورتی‌که مقدار تابع loss بعد از تعداد تکرارهای مشخص شده‌ای تغییر نکرد همگرایی حاصل شده است و باید آموزش به پایان برسد (البته در شرایطی که پارامتر learning_rate تطبیقی[۲۵] نباشد). این تعداد تکرارها با پارامتر n_iter_no_change مشخص می‌شود که مقدار پیش‌فرض آن 10 است. پارامتر random_state درصورتی‌که مقدار int داشته باشد به‌عنوان Seed برای تولید عدد تصادفی عمل می‌کند. پارامتر learning_rate هم می‌تواند مقدار ثابتی داشته باشد و هم متغیر باشد. برای کسب اطلاعات بیشتر در این زمینه به help("sklearn.neural_network.MLPClassifier") مراجعه کنید.

در ادامه از کلاس MLPClassifier، یک شیء می‌سازیم و پارامترهای آن را مقداردهی می‌کنیم:

```
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
mlp=MLPClassifier(hidden_layer_sizes=(20,3),
max_iter=150,alpha=1e-4, solver='sgd', verbose=10,
tol=1e-4, random_state=1, learning_rate_init=0.1)
mlp.fit(learnset, learnlabels)
print("Training set score: %f"
%mlp.score(learnset,learnlabels))
print("Test set score: %f"
%mlp.score(learnset,learnlabels))
mlp.classes_
```

[24]. Stochastic gradient descent
[25]. adaptive

در خروجی خواهیم داشت:

```
Iteration 1, loss = 0.50797205
Iteration 2, loss = 0.46516639
Iteration 3, loss = 0.42332629
Iteration 4, loss = 0.38026803
Iteration 5, loss = 0.35050637
Iteration 6, loss = 0.31915585
Iteration 7, loss = 0.28860783
Iteration 8, loss = 0.26111990
Iteration 9, loss = 0.23599666
Iteration 10, loss = 0.21283000
.
.
.
Iteration 97, loss = 0.00892032
Iteration 98, loss = 0.00883781
Iteration 99, loss = 0.00875684
Training loss did not improve more than
tol=0.000100 for 10 consecutive epochs. Stopping.
Training set score: 1.000000
Test set score: 1.000000
array([0, 1])
```

اکنون که آموزش به پایان رسیده است می‌خواهیم برچسب‌های مجموعه دادهٔ آزمون را پیش‌بینی کنیم:

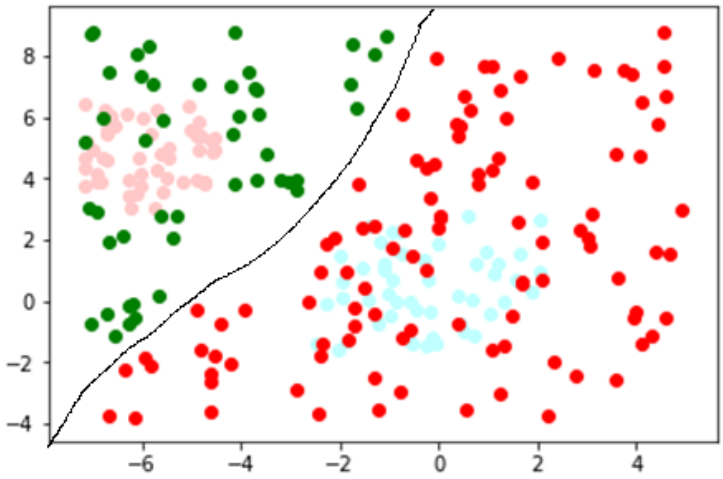
```
predictions=mlp.predict(test_set)
predictions
```

در خروجی خواهیم داشت:

```
array([0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0,
0, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
0, 0, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0, 0])
```

برای درک بهتر نتیجهٔ دسته‌بندی، خروجی را به‌صورت تصویری نشان می‌دهیم:

```
testset=np.array(test_set)
testset[predictions==1]
colors=['#C0FFFF', "#FFC8C8"]
for i in range(2):
    plt.scatter(X[i],Y[i],c=colors[i])
colors=["r","g"]
for i in range(2):
    cls=testset[predictions==i]
    Xt,Yt=zip(*cls)
  plt.scatter(Xt,Yt,c=colors[i])
```



شکل ۲-۱۱: دسته‌بندی نقاط تولید شدهٔ شبکهٔ عصبی MLP (به علت خاکستری بودن تصاویر کتاب، خط جداکننده به صورت دستی رسم شده است)

۲-۲-۳- شبکهٔ عصبی پس انتشار (BP[۲۶])

الگوریتم BP یکی از پرکاربردترین الگوریتم‌ها برای آموزش شبکه‌های عصبی است. این الگوریتم تلاش می‌کند مینیمم تابع خطا را در فضای مربوط به وزن شبکهٔ عصبی با استفاده از روش «کاهش گرادیان» بیابد. بنابراین هدف این الگوریتم یافتن بردارهای وزن برای شبکهٔ عصبی چندلایه است، به‌گونه‌ای که این بردارها تابع خطا را مینیمم کنند. به‌دلیل اینکه در هر تکرار این الگوریتم نیازمند محاسبهٔ کاهش گرادیان هستیم، پیوستگی و مشتق‌پذیری تابع خطا الزامی است. این امر مستلزم این است که تابع فعال‌سازی نیز پیوسته و مشتق‌پذیر باشد [۱۸]. یکی از توابع متداول در شبکه‌های عصبی BP، تابع سیگموید[۲۷] است:

$$f_c(x) = \frac{1}{1+e^{-cx}} \qquad (۷)$$

عدد ثابت c به‌دلخواه انتخاب می‌شود. شکل تابع سیگموید بر اساس مقدار c انتخاب می‌شود. هرچه مقدار c بیشتر باشد، شکل تابع سیگموید به تابع پله‌ای نزدیک‌تر می‌شود [۱۸]. برای سادگی در این کتاب این مقدار را 1 در نظر می‌گیریم. مشتق تابع سیگموید برحسب x که در ادامه به آن نیاز خواهیم داشت عبارت است از:

$$\frac{d}{dx} f(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x)\big(1 - f(x)\big) \qquad (۸)$$

---

[26]. Back Propagation
[27]. Sigmoid function

تعریف تابع خطای شبکه به این شرح است:

$$E = \frac{1}{2}\sum_{i=1}^{p}\|o_i - t_i\|^2 \quad (۹)$$

بعد از مینیمم‌سازی تابع خطای (۹) برای مجموعه دادهٔ آموزش، انتظار می‌رود که شبکه برای یک ورودی جدید، خروجی را تشخیص دهد. الگوریتم BP برای یافتن مینیمم محلی تابع خطا استفاده می‌شود. شبکه ابتدا با وزن‌های تصادفی مقداردهی اولیه می‌شود، سپس با محاسبهٔ گرادیان تابع خطا به‌صورت بازگشتی، وزن‌های شبکه اصلاح می‌شود [۱۸].

۱-۳-۲-۲- مراحل الگوریتم BP

بعد از اینکه در ابتدا وزن‌های شبکه به‌صورت تصادفی مقداردهی اولیه شد، الگوریتم BP برای اصلاحات لازم محاسبه می‌شود. این الگوریتم در چهار مرحله اجرا می‌شود [۱۸]:

- انجام محاسبات پیشخور
- پس انتشار به لایهٔ خروجی
- پس انتشار به لایهٔ مخفی
- به‌روزرسانی وزن‌های شبکه

مرحلهٔ اول: انجام محاسبات پیشخور: در این مرحله بعد از اعمال تابع فعال‌سازی به حاصل‌ضرب وزن‌ها در ورودی، خروجی لایهٔ مخفی (o_1) مشخص می‌شود و سپس بعد از اعمال تابع فعال‌سازی به حاصل‌ضرب وزن‌های لایهٔ مخفی در خروجی لایهٔ مخفی، خروجی لایهٔ خروجی (o_2) مشخص می‌شود.

مرحلهٔ دوم: پس انتشار به لایهٔ خروجی: در این مرحله مشتق جزئی $\frac{\partial E}{\partial w_2}$ محاسبه می‌شود. به‌عبارت‌دیگر، مشتق جزئی E (تابع خطا) برحسب بردار وزن لایهٔ مخفی به لایهٔ خروجی (w_2) محاسبه می‌شود:

با توجه به عبارات (۸و۹) داریم:

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial w} = (o_2 - t)(o_2(1 - o_2)o_1)$$

$$((o_2 - t)\,o_2(1 - o_2))o_1 = \delta_2 o_1 \quad (۱۰)$$

درحالی‌که δ_2 معرف خطای پس انتشار لایهٔ خروجی است.

مرحلهٔ سوم: پس انتشار به لایهٔ مخفی: در این مرحله تلاش بر این است که مشتق جزئی $\frac{\partial E}{\partial w_1}$ محاسبه شود. به‌عبارت‌دیگر، مشتق جزئی E (تابع خطا) برحسب وزن بردار ورودی به لایهٔ مخفی (w_1) محاسبه می‌شود:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial o_1}\frac{\partial o_1}{\partial w} = \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial o_1}\frac{\partial o_1}{\partial w_1} = (o_2 - t)(o_2(1-o_2)w_2)(o_1(1-o_1)x) =$$

$$\delta_2 w_2(o_1(1-o_1)x) = \delta_1 x \qquad (11)$$

درحالی‌که δ_1 معرف خطای پس انتشار لایهٔ مخفی است.

مرحلهٔ چهارم: به‌روزرسانی وزن‌های شبکه: بعد از محاسبهٔ مشتقات جزئی خطا، وزن‌های شبکه در جهت کاهش گرادیان خطا به‌روزرسانی می‌شوند. مقدار ثابت γ طول قدم اصلاح وزن را مشخص می‌کند. میزان تغییرات بردارهای وزن، که به مقدار قبلی وزن افزوده می‌شود، برابر است با:

$$\Delta w_1 = -\gamma x \delta_1 \qquad (12)$$

$$\Delta w_2 = -\gamma o_1 \delta_2 \qquad (13)$$

در حالت یادگیری برخط[28] وقتی برای هر الگو[29] (نمونه) Δw_1 و Δw_2 محاسبه شد، این مقدارها به ترتیب به میزان w_1 و w_2 افزوده می‌شود. به عبارتی به‌روزرسانی وزن، الگو به الگو انجام می‌شود. در مقابل، در حالت یادگیری دسته‌ای[30] اگرچه به‌ازای هر الگو Δw_1 و Δw_2 محاسبه می‌شود، اما در انتها بعد از یک epoch (یک بار اعمال همهٔ الگوها به شبکه) به‌روزرسانی وزن‌ها انجام می‌شود. مسلماً باید شبکه چندین epoch تکرار شود تا به خطای مینیمم برسیم. در این صورت هنگامی‌که با چندین هزار داده مواجه هستیم آموزش شبکه به‌صورت دسته‌ای منطقی به نظر نمی‌رسد [۱۸].

با یک مثال کاربردی آموزش شبکه‌های عصبی با الگوریتم BP را ادامه می‌دهیم:

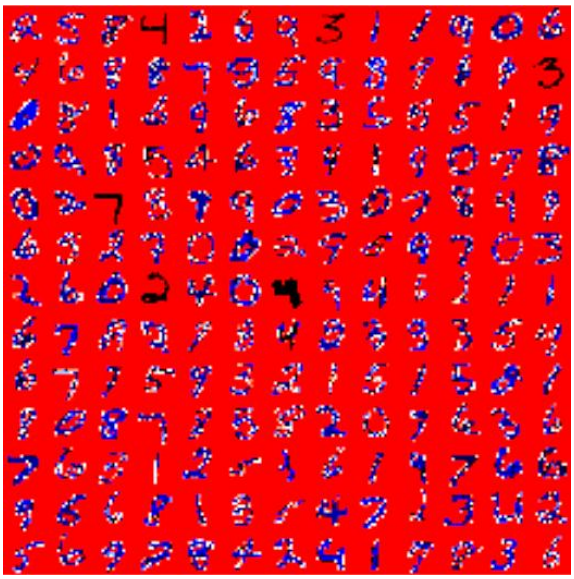
✓ **مثال:**

در این مثال [۱۴] می‌خواهیم از مجموعه دادهٔ MNIST[31] استفاده کنیم که شامل تصاویری از ارقام دست‌نویس است. مجموعه دادهٔ آموزش آن 60000 نمونه و مجموعه دادهٔ آزمون آن 10000 نمونه دارد. این مجموعه داده یک زیرمجموعه از مجموعه دادهٔ NIST است [۱۴]. شکل ۲-۱۰ بخشی از این مجموعه داده را در قالب تصویر نشان می‌دهد. اندازهٔ تصاویر این مجموعه داده 28 X 28 است که در فایل‌های mnist_train.csv و mnist_test.csv قرار دارند. هر سطر از این فایل‌ها شامل یک

---

28. online
29. pattern
30. Batch or offline
31. Modified National Institute of Standards and Technology

تصویر است. این مجموعه داده ۷۸۵ ستون دارد که معرف پیکسل‌های تصویر است و مقادیر آن‌ها بین 0 و 255 است. ستون اول معرف برچسب نمونه‌ها است.



شکل ۲-۱۲: بخشی از مجموعه دادهٔ MNIST

با استفاده از کلاس numpy.loadtxt مجموعه داده را با استفاده از آدرس آن در یک آرایه قرار می‌دهیم. لازم است که مشخص کنیم از چه کاراکتری برای جداسازی عناصر مجموعه داده استفاده شده است. به‌عنوان‌مثال در این مجموعه داده از کاما (,) برای جداسازی استفاده شده است:

```
import numpy as np
import matplotlib.pyplot as plt
image_size=28
no_different_labels=10
image_pixels=image_size*image_size
data_path="C:/Users/APPLE/Desktop/Folders/job/dibag
aran/database/MNIST/"
train_data=np.loadtxt(data_path+"mnist_train.csv",d
elimiter=",")
test_data=np.loadtxt(data_path+"mnist_test.csv",del
imiter=",")
```

- **پیش‌پردازش:**

تصاویر مجموعه داده دارای سطوح خاکستری است. مقادیر آن را به `[0.01, 1]` نگاشت می‌کنیم. علت این کار این است که می‌خواهیم ورودی 0 نداشته باشیم. برای این کار ابتدا با استفاده از `numpy.asfarray` نوع آرایه را به `float` تبدیل می‌کنیم، سپس مقدار هر پیکسل را در `0.99/255` ضرب می‌کنیم و به حاصل آن `0.01` اضافه می‌کنیم:

```
fac=0.99/255
train_imgs=np.asfarray(train_data[:, 1:])*fac +
0.01
test_imgs=np.asfarray(test_data[:, 1:])*fac + 0.01
test_labels=np.asfarray(test_data[:,:1])
train_labels=np.asfarray(train_data[:,:1])
```

ما نیاز داریم که برچسب‌ها را به‌صورت one-hot نمایش دهیم. این کار به کمک دستور `(lr==label).astype(np.int)` صورت می‌گیرد. در ضمن دستور `lr=np.arange(10)` آرایه‌ای با یک سطر و 10 ستون ایجاد می‌کند که شامل اعداد 0 تا 9 است.

```
import numpy as np
lr=np.arange(10)
for label in range(10):
    one_hot=(lr==label).astype(np.int)
    print("label= ", label, " in one-hot
    representation: ",one_hot)
```

در خروجی خواهیم داشت:

```
label=  0  in one-hot representation:  [1 0 0 0 0 0 0 0 0 0]
label=  1  in one-hot representation:  [0 1 0 0 0 0 0 0 0 0]
label=  2  in one-hot representation:  [0 0 1 0 0 0 0 0 0 0]
label=  3  in one-hot representation:  [0 0 0 1 0 0 0 0 0 0]
label=  4  in one-hot representation:  [0 0 0 0 1 0 0 0 0 0]
label=  5  in one-hot representation:  [0 0 0 0 0 1 0 0 0 0]
label=  6  in one-hot representation:  [0 0 0 0 0 0 1 0 0 0]
```

```
label= 7 in one-hot representation: [0 0 0 0 0 0 0 1 0 0]
label= 8 in one-hot representation: [0 0 0 0 0 0 0 0 1 0]
label= 9 in one-hot representation: [0 0 0 0 0 0 0 0 0 1]
```

پس این‌گونه عمل می‌کنیم:

```
lr=np.arange(no_of_different_labels)
# transform labels into one hot representation
train_labels_one_hot=(lr==train_labels).astype(np.f
loat)
test_labels_one_hot=(lr==test_labels).astype(np.flo
at)
# we don't want zeroes and ones in the labels neither:
train_labels_one_hot[train_labels_one_hot==0]=0.1
test_labels_one_hot[test_labels_one_hot==0]=0.1
train_labels_one_hot[train_labels_one_hot==1]=0.99
train_labels_one_hot[train_labels_one_hot==1]=0.99
```

حتماً تاکنون متوجه شده‌اید که خواندن داده‌ها از فرمت csv بسیار کند است، بنابراین داده‌ها را به فرمت باینری تبدیل می‌کنیم. این کار با تابع dump از ماژول pickle انجام می‌شود:

```
import pickle
with
open("C:/Users/APPLE/Desktop/Folders/job/dibagaran/
database/MNIST/
pickled_mnist.pkl","bw") as fh:
  data=(train_imgs,
     test_imgs,
     train_labels,
     test_labels,
     train_labels_one_hot,
     test_labels_one_hot)
  pickle.dump(data,fh)
```

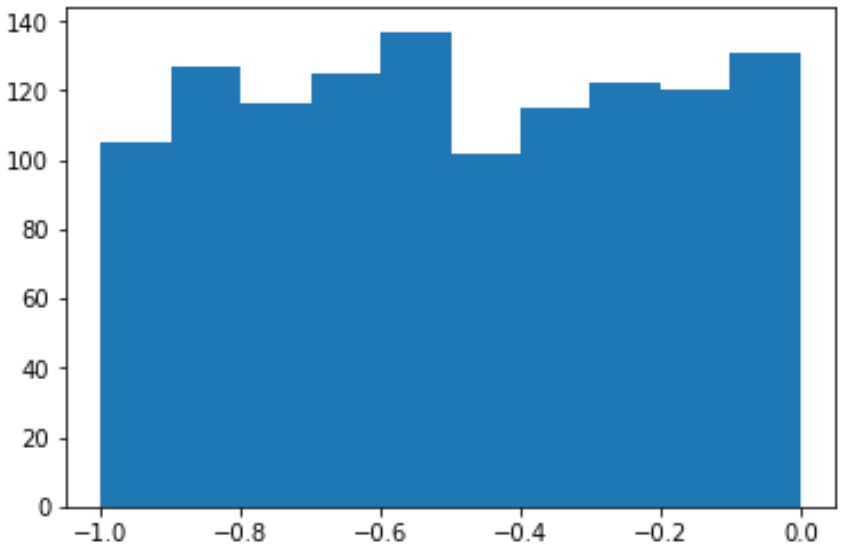
حال می‌توانیم داده‌ها را با دستور pickle.load بخوانیم که بسیار سریع‌تر از استفاده از loadtxt روی فایل‌های csv است:

```
import pickle
with
open("C:/Users/APPLE/Desktop/Folders/job/dibagaran/
database/MNIST/pickled_mnist.pkl","br") as fh:
    data=pickle.load(fh)
train_imgs=data[0]
test_imgs=data[1]
train_labels=data[2]
test_labels=data[3]
train_labels_one_hot=data[4]
test_labels_one_hot=data[5]
image_size=28
no_of_different_labels=10
image_pixels=image_size*image_size
```

- **مدل‌سازی:**

قبل از شروع آموزش شبکهٔ عصبی آنچه مهم است مقداردهی اولیه به ماتریس وزن است. ما می‌توانیم با انتخاب یک توزیع نرمال تصادفی مقادیر تصادفی وزن‌ها را تولید کنیم. تابع `uniform` از کلاس `numpy.random` مقادیرش را از بازهٔ `[low,high)`، که به‌صورت یکنواخت توزیع شده است، انتخاب می‌کند:

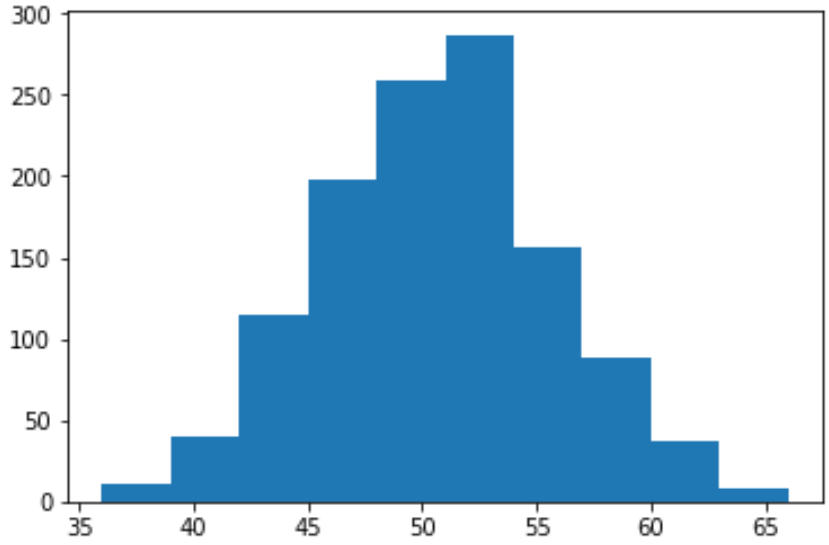
```
import numpy as np
number_of_samples=1200
low=-1
high=0
s=np.random.uniform(low,high,number_of_samples)
import matplotlib.pyplot as plt
plt.hist(s)
plt.show()
```



شکل ۲-۱۳: نمودار هیستوگرام از تولید ۱۲۰۰ عدد تصادفی با توزیع نرمال

تابع بعدی تابع `binomial` از کلاس `numpy.random` است. `n` معرف تعداد اعدادی است که قرار است از بین آن‌ها انتخاب انجام شود. `P` احتمال موفقیت است و `size` تعداد نمونه‌هایی است که انتخاب می‌شود:

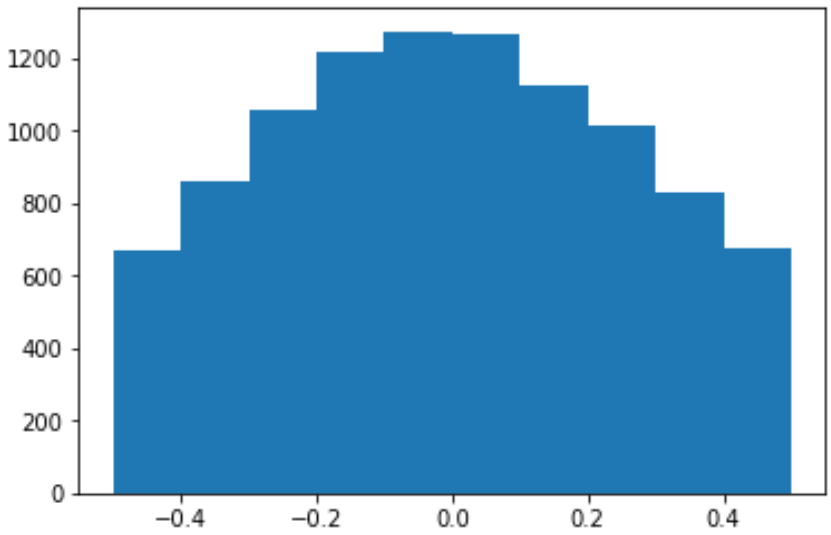
```
n=100
p=0.5
size=1200
s=np.random.binomial(n,p,size)
plt.hist(s)
plt.show()
```



شکل ۲-۱۴: نمودار هیستوگرام از تولید ۱۲۰۰ عدد تصادفی با توزیع bionomial

تابع بعدی `truncnorm` است که برای ساده‌تر شدن، تابعی را به نام `truncate_normal` تعریف می‌کنیم و در داخل این تابع `truncnorm` را فراخوانی می‌کنیم:

```
from scipy.stats import truncnorm
def truncate_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low-mean)/sd,(upp
    mean)/sd,loc=mean,scale=sd)
X=truncate_normal(mean=0, sd=0.4, low=-0.5,
upp=0.5)
s=X.rvs(10000)
plt.hist(s)
plt.show()
```



شکل ۲-۱۵: نمودار هیستوگرام از تولید ۱۰۰۰ عدد تصادفی با توزیع truncnorm

در ادامه به تعریف کلاس شبکهٔ عصبی می‌پردازیم. قبل از این کار ماژول‌های `numpy` و `truncnorm` را فراخوانی می‌کنیم و تابع `truncated_normal` را برای ایجاد اعداد تصادفی برای مقداردهی اولیه به وزن‌ها تعریف می‌کنیم. در تابع `create_weight_matrices` با استفاده از تابع `truncated_normal` بردار وزن را با مقادیر تصادفی تولید شده، مقداردهی اولیه می‌کنیم. `self.wih` معرف وزن‌های بردار ورودی به لایهٔ مخفی است. `self.who` معرف وزن‌های لایهٔ مخفی به لایهٔ خروجی است. در تابع `train` به‌ازای هر بردار ورودی (نمونه یا الگو) و مقدار برچسب آن، توابع

فعال‌سازی و خطای لایهٔ مخفی و خطای لایهٔ خروجی با توجه به عبارات (۱۰-۱۱) محاسبه می‌شود و مقادیر وزن شبکه با توجه به عبارات (۱۲-۱۳) به‌روزرسانی می‌شوند (آموزش برخط). بعد اتمام آموزش و ارائهٔ همهٔ نمونه‌های آموزش به شبکهٔ عصبی، از تابع run برای محاسبهٔ برچسب داده‌های آزمون استفاده می‌شود. در این برنامه از مجموعه دادهٔ بزرگی که ده دسته را در بر دارد استفاده شده است. بنابراین باید نحوهٔ آموزش انجام شده به گونهٔ مناسبی ارزیابی شود. برای این کار نیاز است که ماتریس تداخل[۳۲] تعریف شود. در ادامه، در کلاس NeuralNetwork تابعی به نام confusion_matrix تعریف می‌کنیم. هر سطر این ماتریس معرف کلاس‌های واقعی و هر ستون آن معرف کلاس‌های پیش‌بینی شده است. هر درایه از این ماتریس در سطر i و ستون j، گویای این مطلب است که چند نمونه به کلاس i متعلق است درحالی‌که کلاس j برای این نمونه‌ها پیش‌بینی شده است. در حالتی که مسئله دو دسته‌ای است، این ماتریس به‌صورت جدول‌های ۲-۱ و ۲-۲ است.

```
import numpy as np
@np.vectorize
def sigmoid(x):
    return 1 / (1 + np.e ** -x )
activation_function=sigmoid
from scipy.stats import truncnorm
def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm((low-mean)/sd, (upp-mean)/sd,
    loc=mean, scale=sd)


class NeuralNetwork:
    def __init__(self,
                 no_in_nodes,
                 no_out_nodes,
                 no_hidden_nodes,
                 learning_rate):
        self.no_in_nodes=no_in_nodes
        self.no_out_nodes=no_out_nodes
        self.no_hidden_nodes=no_hidden_nodes
```

[32]. Confusion matrix

```
        self.learning_rate=learning_rate
        self.create_weight_matrices()

    def create_weight_matrices(self):

        rad= 1 / np.sqrt(self.no_in_nodes)
        X=truncated_normal(mean=0, sd=1, low=-rad,
        upp=rad)

        self.wih=X.rvs((self.no_hidden_nodes,self.n
        o_in_nodes))

        rad= 1/np.sqrt(self.no_hidden_nodes)
        X=truncated_normal(mean=0, sd=1, low=-rad,
        upp=rad)

        self.who=X.rvs((self.no_out_nodes,self.no_h
        idden_nodes))

    def train(self,input_vector,target_vector):
        """
        input_vector and target_vector can
        be tuple, list or ndarray
        """
        input_vector=np.array(input_vector,
        ndmin=2).T

        target_vector=np.array(target_vector,
        ndmin=2).T

        output_vector1=np.dot(self.wih,
        input_vector)
```

```
        output_hidden=activation_function(output_v
        ector1)

        output_vector2=np.dot(self.who,output_hidd
        en)

        output_network=activation_function(output_
        vector2)


        output_error=target_vector-output_network


    # update the weights:
        tmp=output_error * output_network * (1.0 -
        output_network)
        tmp=self.learning_rate * np.dot(tmp,
        output_hidden.T)
        self.who += tmp


    # calculate hidden errors:
        hidden_errors = np.dot(self.who.T,
        output_error)


         # update the weights:
        tmp = hidden_errors * output_hidden * (1.0
        - output_hidden)
         self.wih += self.learning_rate *
         np.dot(tmp, input_vector.T)


def run(self, input_vector):
    # input_vector can be tuple, list or ndarray
```

```
        input_vector=np.array(input_vector,ndmin=2
        ).T
        output_vector= np.dot( self.wih,
        input_vector)

        output_vector=activation_function(output_v
        ector)
        output_vector=np.dot(self.who,
        output_vector)

        output_vector=activation_function(output_v
        ector)
        return output_vector


    def confusion_matrix(self, data_array, labels):
        cm=np.zeros((10,10),int)
        for i in range(len(data_array)):
            res=self.run(data_array[i])
            res_max=res.argmax()
            target=labels[i][0]
            cm[res_max,int(target)] += 1
        return cm


    def precision(self, label, confusion_matrix):
        row=confusion_matrix[:,label]
        return
        confusion_matrix[label,label]/row.sum()


    def recall(self, label, confusion_matrix):
        column=confusion_matrix[label,:]
```

```
        return
        confusion_matrix[label,label]/column.sum()


    def evaluate(self,data,labels):
        corrects,wrongs=0,0
        for i in range(len(data)):
            res=self.run(data[i])
            res_max=res.argmax()
            if res_max==labels[i]:
                corrects+=1
            else:
                wrongs+=1
        return corrects,wrongs
```

جدول ۰-۱: ماتریس تداخل برای یک مسئلهٔ دو دسته‌ای

		Predicted class	
Actual class		No	**Yes**
	No	12	**1**
	Yes	2	**17**

جدول ۰-۲: معانی مقدار هر درایه از ماتریس تداخل

		Predicted class	
Actual class		Negative	**Positive**
	Negative	TN=True Negative	**FP=False Positive**
	Positive	FN=False Negative	**TP=True Positive**

در ادامه معیارهای مهمی در اندازه‌گیری کارایی الگوریتم‌های یادگیری ماشین ارائه می‌شود:

- **درستی:**[۳۳] این معیار همیشه برای ارزیابی صحت الگوریتم‌ها کافی نیست. مخصوصاً زمانی که با مجموعه داده‌ای مواجه باشیم که متعادل نباشد. به عبارت روشن‌تر تعداد نمونه‌های کلاس مثبت خیلی بیشتر از تعداد نمونه‌های کلاس منفی باشد:

$$AC = \frac{TN+TP}{TN+FN+TP+FP} \quad (۱۴)$$

- **نرخ تشخیص درست کلاس مثبت:**[۳۴]

$$recall = \frac{TP}{TP+FN} \quad (۱۵)$$

- **نرخ تشخیص درست کلاس منفی:**[۳۵]

$$TNR = \frac{TN}{TN+FP} \quad (۱۶)$$

- **دقت:**[۳۶]

$$precision = \frac{TP}{TP+FP} \quad (۱۷)$$

معیارهای اندازه‌گیری کارایی الگوریتم یادگیری ماشین را در حالتی که با چند دسته مواجه هستیم، بررسی می‌کنیم:

جدول ۰-۳: تداخل برای یک مسئلهٔ سه دسته‌ای

	Predicted class			
Actual class		Orange	apple	**Pear**
	Orange	10	1	**2**
	Apple	2	8	**1**
	Pear	1	1	**8**

[33]. Accuracy
[34]. True positive rate
[35]. True negative rate
[36]. Precision

با در نظر گرفتن جدول ۲-۳ داریم:

$$precision_i = \frac{M_{ii}}{\sum_j M_{ji}} \quad (۱۸)$$

$$recall_i = \frac{M_{ii}}{\sum_j M_{ij}} \quad (۱۹)$$

به ادامهٔ حل مسئلهٔ تشخیص ارقام دست‌نویس برمی‌گردیم:

به‌علت اینکه تعداد نودهای خروجی 10 عدد است، در تابع confusion_matrix یک ماتریس صفر 10X10 ایجاد می‌کنیم. سپس برای هر بردار ورودی، تابع run اجرا می‌شود تا دستهٔ پیش‌بینی شده برای این بردارها مشخص شود. با استفاده از argmax() اندیس عنصری از بردار خروجی تابع run، که بیشترین مقدار را دارد، مشخص می‌کنیم. مقدار این اندیس همان شماره دسته‌ای است که برای بردار ورودی پیش‌بینی شده است. از طرف دیگر، برچسب این بردار را به‌عنوان target در نظر می‌گیریم. در ماتریس 10X10ی که ایجاد کرده بودیم، خانه‌ای که اندیس سطر آن برابر است با شماره دستهٔ پیش‌بینی شده و شماره ستون آن همان target بردار ورودی است، یک عدد به مقدار آن اضافه می‌کنیم (توجه: در کد برنامه سطر و ستون تعریف شده برای ماتریس تداخل ترانهاده شده است). تابع evaluate هم که کاملاً مشخص است تعداد نمونه‌هایی را که به‌درستی یا به‌اشتباه پیش‌بینی شده‌اند مشخص می‌کند.

اکنون از کلاس شبکهٔ عصبی NeuralNetwork یک شیء می‌سازیم و پارامترهای آن را مقداردهی می‌کنیم. سپس به‌ازای هر تصویری در مجموعه دادهٔ آموزش، تابع train را فراخوانی می‌کنیم. بعد از آموزش همهٔ تصاویر مجموعه دادهٔ آموزش به شبکهٔ عصبی، به‌ازای هر کدام از تصاویر مجموعه دادهٔ آزمون، تابع run را فراخوانی می‌کنیم تا پیش‌بینی شود که هر کدام از تصاویر مجموعه دادهٔ آزمون به کدام‌یک از ده دستهٔ ارقام دیجیتال (0,...9) متعلق است. سپس برچسب تصاویر مجموعه دادهٔ آزمون (که مشخص می‌کند هر تصویر به کدام دسته متعلق است) را به همراه آرگومان بزرگ‌ترین عنصر بردار خروجی (که معرف شماره دستهٔ پیش‌بینی شده برای تصویر است) به همراه مقدار بزرگ‌ترین عنصر بردار خروجی در کنار هم چاپ می‌کنیم:

```
ANN=NeuralNetwork(no_in_nodes=image_pixels,
                  no_out_nodes=10,
                  no_hidden_nodes=100,
                  learning_rate=0.1)
for i in range(len(train_imgs)):

ANN.train(train_imgs[i],train_labels_one_hot[i])
for i in range(20):
```

```
    res=ANN.run(test_imgs[i])
    print(test_labels[i], np.argmax(res),
    np.max(res))
```

در خروجی داریم:

```
[7.] 7 0.986174995873378
[2.] 2 0.769559764098301
[1.] 1 0.9854811042320714
[0.] 0 0.9775832109633547
[4.] 4 0.883493758867519
[1.] 1 0.9872677525541655
[4.] 4 0.9864167275003782
[9.] 9 0.9639239724916239
.
.
.
```

در ادامه مقادیر Accuracy را برای مجموعه داده آزمون و آموزش چاپ می‌کنیم. ماتریس تداخل را ایجاد می‌کنیم و چاپ می‌کنیم. مقادیر recall و precision را برای همهٔ دسته‌ها محاسبه و چاپ می‌کنیم:

```
corrects, wrongs =
ANN.evaluate(train_imgs,train_labels)
print("accuracy of train :
",corrects/(corrects+wrongs))
corrects, wrongs = ANN.evaluate(test_imgs,
test_labels)
print("accuracy of test : ",
corrects/(corrects+wrongs))
cm=ANN.confusion_matrix(train_imgs,train_labels)
print(cm)
for i in range(10):
```

```
    print("digit ",i," percision:
    ",ANN.precision(i,cm), " recall:
    ",ANN.recall(i,cm))
```

در خروجی داریم:

```
accuracy of train :  0.9429666666666666

accuracy of test :  0.9399

[[5790    0   57   19   10   32   34   10   17   19]
 [   0 6617   45   15   18   31   25   44   86    9]
 [   0   18 5375   46   16   14    3   35    9    0]
 [   3   24  142 5783    1  115    3   34   81   66]
 [   5   13   62    5 5457   33   11   55   27   70]
 [  10    6    6   74    1 4988   64    0   27    5]
 [  26    3   45   16   45   61 5730    5   22    2]
 [   1   12   47   35    3    5    1 5644    2   28]
 [  73   29  161   69   12   60   47   24 5468   24]
 [  15   20   18   69  279   82    0  414  112 5726]]

digit  0  percision:  0.9775451629241938  recall:
0.966933867735471

digit  1  percision:  0.9814595075645209  recall:
0.960377358490566

digit  2  percision:  0.9021483719368916  recall:
0.9744379985496737

digit  3  percision:  0.9432392758114501  recall:
0.9249840051183621

digit  4  percision:  0.9340979116740842  recall:
0.9510282328337399

digit  5  percision:  0.9201254381110496  recall:
0.9627485041497781
```

```
digit 6 percision: 0.9682325109834403 recall:
0.9622166246851386

digit 7 percision: 0.900877893056664 recall:
0.9768085842852198

digit 8 percision: 0.9345411040847719 recall:
0.9163733869616223

digit 9 percision: 0.9625147083543453 recall:
0.8501855976243504
```

۲-۳- ماشین بردار پشتیبان (SVM[۳۷])

ماشین بردار پشتیبان (SVM) روشی قدرتمند و انعطاف‌پذیر از نوع الگوریتم‌های با ناظر یادگیری ماشین برای مسائل طبقه‌بندی و نیز رگرسیون است.

- **SVM** برای طبقه‌بندی
- **SVR**[۳۸] برای رگرسیون

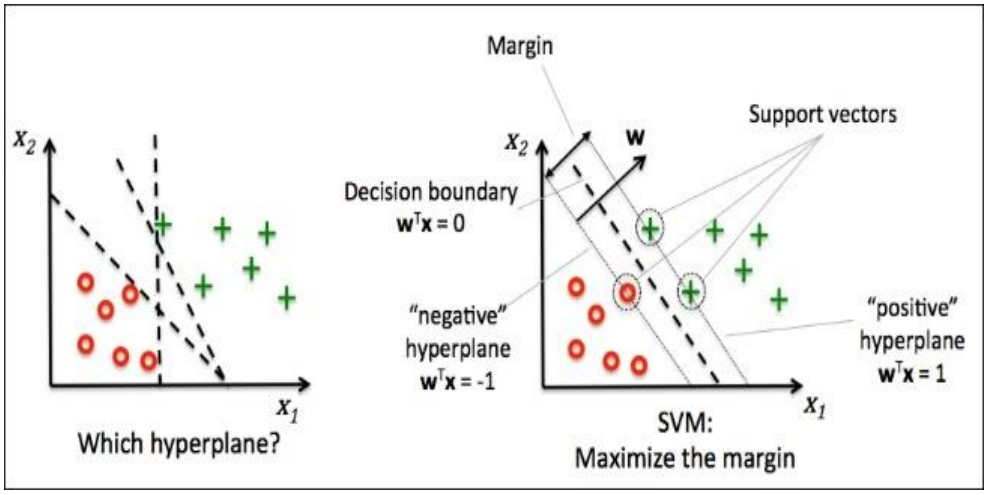
ما در این بخش، به واکاوی این روش و استفاده از آن در مسائل طبقه‌بندی، که متداول‌تر است، می‌پردازیم [۱۹].

روش‌های طبقه‌بندی خطی سعی می‌کنند که با ساختن یک ابرسطح (که عبارت است از یک معادلهٔ خطی)، داده‌ها را از هم تفکیک کنند. روش طبقه‌بندی خطی ماشین بردار پشتیبان، بهترین ابرسطحی را پیدا می‌کند که با حداکثر حاشیه،[۳۹] داده‌های مربوط به دو طبقه را از هم تفکیک کند.

فرض کنیم مجموعه نقاط داده را در اختیار داریم؛ هر x_i یک بردار p بعدی از اعداد حقیقی است که به کلاس c_i متعلق است. به‌منظور درک بهتر مطلب، فرض می‌کنیم که با یک مسئلهٔ دو دسته‌ای مواجه هستیم. در شکل ۲-۱۶ تصویری از یک مجموعه دادهٔ متعلق به دو کلاس نشان داده شده که روش ماشین بردار پشتیبان بهترین ابرسطح را برای جداسازی آن‌ها انتخاب می‌کند.

---

37. Support Vector Machine
38. Support Vector Regression
39. margin



شکل ۲-۱۶: **ابرسطح با حداکثر مرز جداکننده به همراه مرزهای جداکننده برای طبقه‌بندی نمونه داده‌های مربوط به دو طبقهٔ متفاوت. نمونه‌های قرار گرفته روی مرزها، بردارهای پشتیبان نام دارند.**

هدف اصلی SVM این است که یک تفکیک‌کنندهٔ مناسب انتخاب شود. منظور صفحه‌ای است که بیشترین فاصله را با نقاط همسایه از هر دو طبقه دارد. به‌عبارت‌دیگر، با دو ابرسطح موازی، که حداقل از یکی از نقاط دو دسته عبور می‌کنند، بیشترین فاصله را دارد. این نقاط بردارهای پشتیبان نام دارند. این دو ابرسطح موازی در عبارات ۲۰ و ۲۱ نشان داده شده است:

$$w.x - b = ۱ \quad (۲۰)$$

$$w.x - b = -۱ \quad (۲۱)$$

نکتهٔ قابل‌توجه این است که اگر داده‌های آموزش به‌صورت خطی تفکیک‌پذیر باشند، دو ابرسطح مرزی به‌گونه‌ای انتخاب می‌شوند که هیچ داده‌ای بین آن‌ها نباشد. در ضمن باید فاصلهٔ بین این دو ابرسطح موازی بیشینه باشد. با به‌کارگیری قضایای هندسی، فاصلهٔ این دو ابرسطح عبارت است از $2 / \|w\|$، پس باید $\|w\|$ را به حداقل رساند. برای هر i، با اعمال محدودیت‌های زیر اطمینان می‌یابیم که هیچ نقطه‌ای در مرز قرار نمی‌گیرد:

$$wx_i - b \geq ۱ \quad (۲۲)$$ برای داده‌های مربوط به طبقهٔ اول

$$wx_i - b \leq -۱ \quad (۲۳)$$ برای داده‌های مربوط به طبقهٔ دوم

می‌توان این محدودیت را به‌صورت رابطهٔ زیر نشان داد:

$$c_i(w\,x_i - b) \geq 1, \quad ۱ \leq i \leq n \quad (۲۴)$$

به‌طوری‌که $c_i = ۱$ درصورتی‌که نمونهٔ iام به کلاس اول تعلق داشته باشد و $c_i = -۱$ درصورتی‌که نمونهٔ iام به کلاس دوم تعلق داشته باشد. بنابراین مسئلهٔ بهینه‌سازی به صورت عبارت (۲۵) تعریف می‌شود:

به حداقل رساندن w، با در نظر گرفتن محدودیت زیر:

$$\min \|w\|$$
$$c_i(w.x_i - b) \geq 1, \quad 1 \leq i \leq n \qquad (۲۵)$$

بعد از حل این مسئله مینیمم‌سازی و به دست آوردن w، معادلهٔ ابرسطح‌های موازی با خط جداکنندهٔ دو کلاس به دست می‌آید و با ورود دادهٔ جدید، درصورتی‌که $wx_i - b \geq 1$ باشد، نمونهٔ آزمایشی به طبقهٔ اول تعلق می‌گیرد و درصورتی‌که $w.x_i - b \leq -1$ برقرار باشد، نمونهٔ آزمایشی به طبقهٔ دوم تعلق می‌گیرد.

این روش، هر نمونه داده را به‌صورت یک نقطه در فضای n بعدی (n، تعداد ویژگی‌هایی است که یک نمونه داده دارد) روی نمودار پراکندگی داده‌ها ترسیم می‌کند و با ترسیم یک خط راست، داده‌های مختلف و متمایز از یکدیگر را دسته‌بندی می‌کند. در اینجا ما یک خط یا منحنی[۴۰] پیدا می‌کنیم که کلاس‌ها را از یکدیگر جدا کند.

در پایتون، کتابخانهٔ scikit-learn به‌طور گسترده برای پیاده‌سازی الگوریتم‌های یادگیری ماشین استفاده می‌شود. الگوریتم ماشین بردار پشتیبان نیز در این کتابخانه موجود است که در ادامه بیشتر با این کتابخانه آشنا می‌شویم. حال با افزودن کتابخانه‌های استاندارد مربوط به این روش در پایتون، پیاده‌سازی این روش را شروع می‌کنیم:

✓ **مثال:**

به‌عنوان‌مثال [۱۹]، نمونه‌ای ساده از یک طبقه‌بندی را در نظر بگیرید که در آن نقاط دو کلاس به‌خوبی از هم جدا می‌شوند. نقاط دو دسته را با تابع make_blobs ایجاد می‌کنیم. تعداد نمونه‌ها را ۵۰ عددی در نظر می‌گیریم. با پارامتر centers مشخص می‌کنیم که چند دسته دادهٔ جدا از هم می‌خواهیم داشته باشیم. انحراف معیار نمونه‌های هر دسته را 0.6 قرار می‌دهیم. درصورتی‌که مقدار پارامتر random_state را یک عدد صحیح قرار دهیم، با هر بار اجرا همان نمونه‌ها تولید می‌شوند. سپس نمونه‌ها را با رنگ‌بندی زمستان!! ☺ با تابع scatter رسم می‌کنیم. پارامتر c در تابع scatter معرف توالی است که با رنگ‌های cmap متناظر می‌شود. در اینجا این پارامتر را برابر برچسب‌های مجموعه داده قرار داده‌ایم. پارامتر s هم اندازهٔ قلم را نشان می‌دهد:

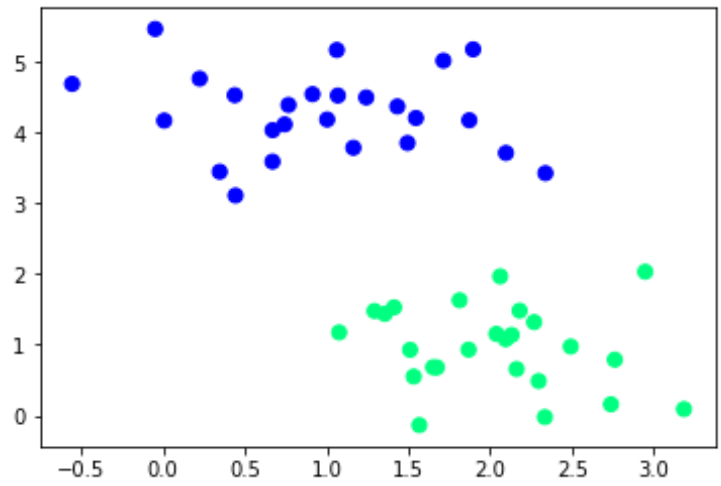
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
from sklearn.datasets.samples_generator import make_blobs
X, y=make_blobs(n_samples=50, centers=2,
random_state=0, cluster_std=0.6)
```

---

40. curve

```
plt.scatter(X[:,0],X[:,1], c=y, s=50,
cmap='winter')
```

و آنچه در خروجی مشاهده می‌کنیم:



شکل۲-۱۷: نمونه‌های تولید شده به‌صورت تصادفی

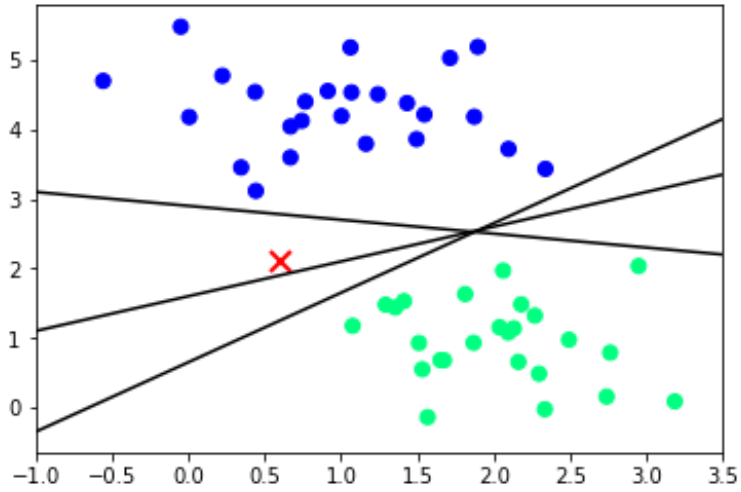
یک طبقه‌بند متمایزکننده خطی[41] سعی در ترسیم خط مستقیمی دارد که دو مجموعه از داده‌ها را از هم جدا کرده و بدین ترتیب الگویی برای طبقه‌بندی ایجاد کند. برای داده‌های دوبعدی، مانند آنچه در شکل ۲-۱۷ نشان داده شد، می‌توانستیم این کار را با دست نیز انجام دهیم اما مسئله‌ای که وجود دارد این است که بیش از یک خط تقسیم احتمالی وجود دارد که می‌تواند کاملاً دو کلاس را از هم جدا کند [۱۹].

برای نشان دادن این موضوع، ابتدا تعدادی عدد در بازهٔ `[-1, 3.5]` با تابع `linspace` تولید می‌کنیم. بعد از رسم نقاط، خطوطی را با شیب‌ها و عرض از مبدأهای متفاوت، به‌گونه‌ای که جداکنندهٔ دو دسته نمونه باشند، با استفاده از نقاطی که با تابع `linspace` ایجاد کرده بودیم، رسم می‌کنیم:

```
xfit=np.linspace(-1,3.5)
plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap='winter')
plt.plot([0.6],[2.1],'x',color='red',markeredgewidt
h=2,markersize=10)
for m,b in [(1,0.65),(0.5,1.6),(-0.2,2.9)]:
  plt.plot(xfit,m*xfit+b,'-k')
  plt.xlim(-1,3.5)
```

41. linear discriminative classifier

در خروجی داریم:



شکل ۲-۱۸: جداسازی نمونه‌های دو دسته بدون استفاده از SVM

این سه خط جداکننده تفاوت زیادی با یکدیگر دارند که با وجود این، دو کلاس را کاملاً از هم جدا کرده‌اند. اکنون، این پرسش مطرح می‌شود که «چگونه می‌توان این خط راست مناسب را تعیین کرد؟»

ماشین‌های بردار پشتیبان راهی برای بهبود این زمینه ارائه می‌دهند. با این نگاه به‌جای اینکه یک خط نازک بین کلاس‌ها بکشیم، می‌توانیم برای هر خط یک حاشیه از عرض تا نزدیک‌ترین نقطه داشته باشیم. در ماشین‌های بردار پشتیبان، خطی که این حاشیه را به حداکثر برساند، خطی است که ما به‌عنوان مدل بهینه[۴۲] انتخاب خواهیم کرد[۱۹].

از کتابخانهٔ `Scikit-Learn` برای آموزش یک مدل SVM با کرنل خطی استفاده می‌کنیم. ابتدا پارامتر C را برای تعداد بسیار بالایی تنظیم می‌کنیم این پارامتر، جریمهٔ[۴۳] مربوط به ترم خطا را مشخص می‌کند. وقتی برای این پارامتر مقدار زیادی را در نظر می‌گیریم به این معنی است که باید بیشترین مینیمم‌سازی را روی ترم خطا داشته باشیم. پارامتر `kernel` را خطی در نظر می‌گیریم. بعد از مشخص کردن این پارامترها و ساختن یک شیء از `SVC (Support Vector Classifier)` به نام `model`، از تابع `fit` استفاده می‌کنیم تا الگوریتم را به مجموعه داده اعمال کنیم:

```
from sklearn.svm import SVC
model=SVC(kernel='linear', C=1E10)
model.fit(X,y)
```

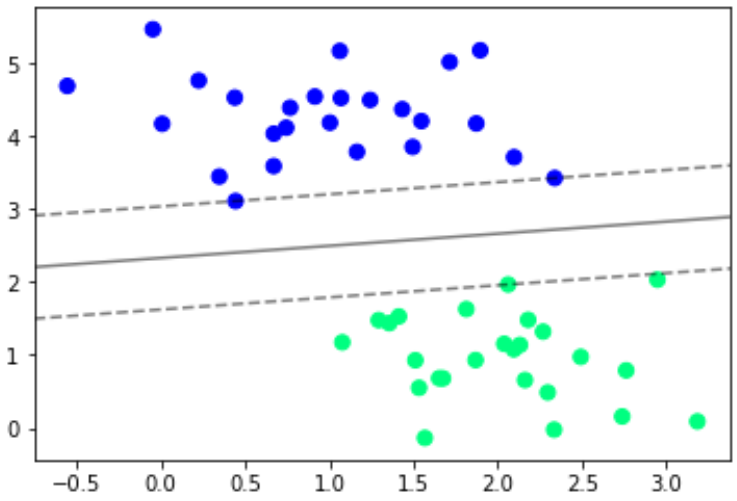
---

42. optimal model
43. penalty

تابع plot_svc_decision_function را برای رسم نتیجهٔ دسته‌بندی تعریف می‌کنیم و مدلی را که ساخته‌ایم به‌عنوان پارامتر برای این تابع می‌فرستیم. ابتدا صفحهٔ مختصات را با تابع gca() تعریف می‌کنیم. سپس محدودهٔ محورهای مختصات را با استفاده از توابع get_xlim() و get_ylim() به دست می‌آوریم. با استفاده از تابع linspace، 30 عدد در هر محور مختصات، در محدوده‌های این محورها تولید می‌کنیم. در ادامه با تابع meshgrid شبکه‌ای از همهٔ نقاط مختصات را به وجود می‌آوریم. مختصات x این نقاط را با تابع ravel به یک آرایه تبدیل می‌کنیم. همین کار را برای مختصات y این نقاط انجام می‌دهیم. این دو آرایه را در آرایه‌ای از نوع stack قرار می‌دهیم و stack را به تابع decision_function از مدلی که قبلاً ساخته بودیم ارسال می‌کنیم تا مقادیر مختصات را در تابع تصمیم به دست آوریم. مقدار تابع تصمیم در این مختصات را به همراه مختصات نقاط به تابع contour برای رسم تابع تصمیم و محورهای موازی ارسال می‌کنیم و با استفاده از تابع scatter، support_vectorهای مدل ارزیابی شده را در شکل مشخص می‌کنیم. شکل ۲-۱۹ به‌خوبی دسته‌بندی نمونه‌های دو دسته را با روش SVM خطی نشان می‌دهد.

```
def plot_svc_decision_function(model,
                               ax=None,
                               plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax=plt.gca()
    xlim=ax.get_xlim()
    ylim=ax.get_ylim()

    # create grid to evaluate model
    x=np.linspace(xlim[0],xlim[1],30)
    y=np.linspace(ylim[0],ylim[1],30)
    Y,X=np.meshgrid(y,x)
    xy=np.vstack([X.ravel(),Y.ravel()]).T
    P=model.decision_function(xy).reshape(X.shape)
    # plot decision boundary and margins
    ax.contour(X,Y,P,colors='k',
```

```
                levels=[-1.0,0,1.0],
                alpha=0.5,linestyles=['--','-','--'])
    # plot support vectors
    if plot_support:
      ax.scatter(model.support_vectors_[:,0],
                 model.support_vectors_[:,1],
                 s=300,
                 linewidth=1,facecolors='none')
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap='winter')
plot_svc_decision_function(model)
```



شکل ۲-۱۹: دسته‌بندی نمونه‌های دو دسته با استفادهٔ SVM خطی (نمونه‌هایی که روی خطوط مرزی قرار دارند، بردارهای پشتیبان هستند).

۲-۳-۱- فراتر از مرزهای خطی: هستهٔ SVM

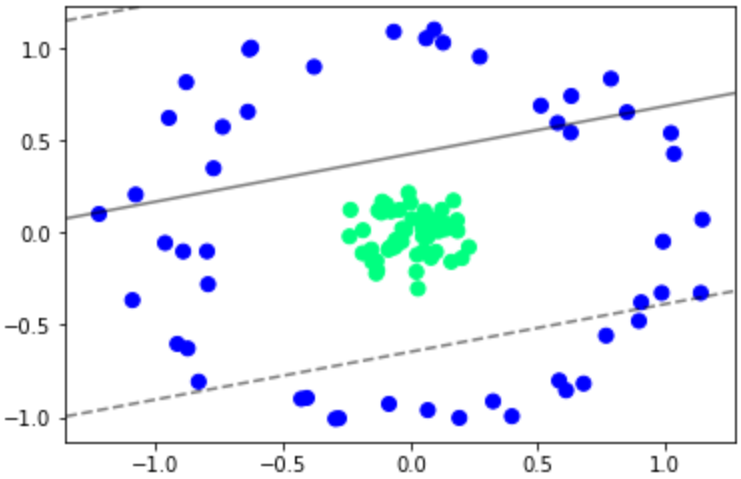
SVM وقتی با هسته‌ها[۴۴] ترکیب می‌شود، بسیار قدرتمند می‌شود. درواقع توابعی وجود دارند که فضای ورودی بعد پایین را دریافت کرده و آن را به بعد بالاتر تبدیل می‌کنند. این تبدیل که با توابع هسته انجام می‌شود، مسئله‌ای را که به‌صورت خطی قابل جداسازی نیست، به مسئلهٔ قابل جداسازی مبدل می‌کند [۱۹]. به‌عنوان یک چالش برای این بخش داده‌هایی را در نظر می‌گیریم که از نظر خطی قابل جداسازی نیستند. برای تولید چنین داده‌هایی از تابع `make_circles` استفاده می‌کنیم. البته قبل از آن کتابخانهٔ متناظر آن را به برنامه فراخوانی می‌کنیم. این تابع نقاط هر دسته را به‌صورتی تولید می‌کند که پراکندگی آن‌ها دایره‌گونه باشد. سپس از کلاس SVC یک شیء به نام `clf` می‌سازیم و با تابع `fit` مجموعه داده را به مدل اعمال می‌کنیم و درنهایت تابع تصمیم تولید شده را به همراه نقاط مجموعه داده رسم می‌کنیم:

```
from sklearn.datasets.samples_generator import make_circles
from sklearn.svm import SVC
from matplotlib import pyplot as plt
import numpy as np

#create dataset
X,Y=make_circles(100, factor=0.1, noise=0.1)

#apply linear SVM
clf=SVC(kernel='linear').fit(X,Y)

#plot patterns and decision function
plt.scatter(X[:,0],X[:,1],c=Y,s=50,cmap='winter')
plot_svc_decision_function(clf, plot_support=True)
```

همان‌طور که در خروجی (شکل ۲-۲۰) مشاهده می‌کنید نمونه‌ها با SVM خطی تفکیک‌پذیر نیستند.

---

44. Kernel



شکل ۲-۲۰: دسته‌بندی نمونه‌هایی که به‌صورت خطی تفکیک‌پذیر نیستند با SVM خطی

مشخص است که هیچ تمایزکنندهٔ خطی هرگز قادر به جدا کردن این داده‌ها نخواهد بود؛ اما ما می‌توانیم به این فکر کنیم که چگونه می‌توانیم داده‌ها را در یک بعد بالاتر قرار دهیم، به‌طوری‌که یک جداکنندهٔ خطی برای جدا کردن کلاس‌ها کافی باشد. به‌عنوان‌مثال، می‌توانیم محاسبهٔ یکی از توابع پایهٔ شعاعی[۴۵] را با محوریت تودهٔ میانی[۴۶] در نظر بگیریم:

$$r = e^{-\|x-c\|^2} \qquad (۲۶)$$

```
r=np.exp(-(X**2).sum(1))
```

عبارت `(X**2).sum(1)` مؤلفه‌های هر نمونه را به توان دو می‌رساند و با هم جمع می‌کند. `c` به‌عنوان مرکز مختصات در نظر گرفته شده است (نقطهٔ `[0,0]`).

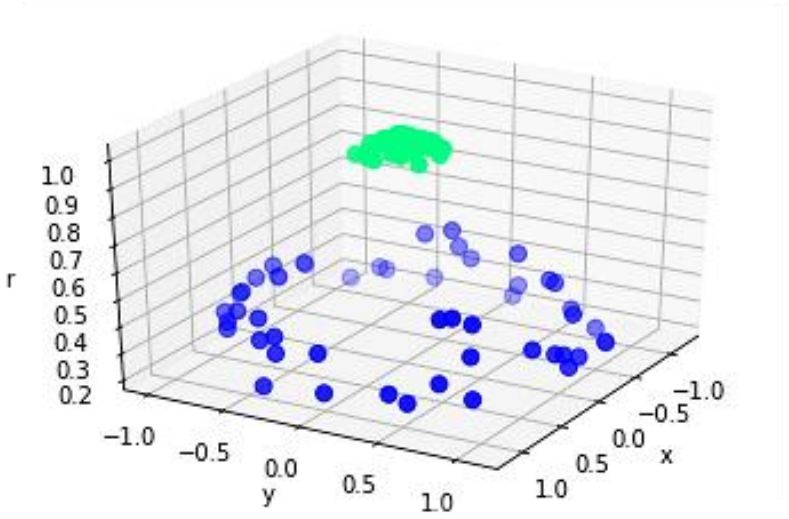
حال ما می‌توانیم با استفاده از یک طرح سه‌بعدی این بعد از اطلاعات اضافی را رسم کنیم. برای این کار تابع `plot_3D` را تعریف می‌کنیم. در این تابع ابتدا یک `subplot` تعریف می‌کنیم که `projection` آن `'3d'` باشد. سپس با استفاده از تابع `scatter3D` نمونه‌ها را در یک بعد بالاتر به همراه مقادیر `r` در محور `z` رسم می‌کنیم. با تابع `view_init` زاویه‌های دید محور `z` و محورهای افقی `x, y` را مشخص می‌کنیم (به ترتیب `elev` و `azim`). درنهایت تابع `plot_3D` را با مقادیر نمونه‌های مجموعه داده به همراه برچسب‌هایشان فراخوانی می‌کنیم:

```
def plot_3D(elev=30, azim=30, X=X, y=y):
```

[45]. radial basis function
[46]. middle clump

```
    ax=plt.subplot(projection='3d')

    ax.scatter3D(X[:,0],X[:,1],r,c=y,s=50,cmap='win
    ter')
    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('r')


plot_3D(X=X, y=y)
```

همان‌طور که در خروجی (شکل ۲-۲۱) مشاهده می‌کنید داده‌ها در یک بعد بالاتر قرار گرفته‌اند:



شکل ۲-۲۱: انتقال داده‌ها به یک بعد بالاتر

می‌توانیم ببینیم که با نگاشت داده‌ها به بعد جدید، با ترسیم یک صفحه جداکننده (در $r=0.7$)، به‌راحتی داده‌ها به‌صورت خطی از هم جدا می‌شوند. در اینجا ما مجبور بودیم نگاشت خود را انتخاب کرده و با دقت تنظیم کنیم. اگر تابع پایهٔ شعاعی خود را در مکان مناسب متمرکز نکرده بودیم (نقطهٔ $[0,0]$)، دیگر شاهد چنین نتایج خوب و جداکنندهٔ خطی نبودیم. به‌طورکلی، نیاز به چنین انتخاب درستی همیشه راحت نیست و معمولاً با مشکل همراه است، پس بهتر است به‌نوعی به‌طور خودکار بهترین تابع پایه انتخاب شود [۱۹].

یک استراتژی برای این منظور محاسبهٔ یک تابع پایه محور در هر نقطه از مجموعه داده است و در ادامه نیز اجازه دهیم که الگوریتم SVM نتایج را غربال کند. این نوع تبدیل تابع پایه را تبدیل هسته[۴۷] می‌نامند، چون بر اساس یک رابطهٔ شباهت (یا هسته) بین جفت نقاط به دست آمده است. یک مشکل بالقوه در مورد این استراتژی (نگاشت N نقطه به N بعد) این است که بزرگ شدن N محاسبات بسیار سنگینی را به‌دنبال خواهد داشت. بااین‌حال، به‌دلیل وجود یک روش خوب و شسته‌رفته، به نام ترفند هسته،[۴۸] متناسب‌سازی این داده‌ها می‌تواند به‌طور ضمنی انجام شود؛ یعنی بدون ایجاد نگاشت کامل Nبعدی هسته! ترفند هسته در روش SVM ایجاد شده و یکی از دلایلی است که نشان می‌دهد این روش بسیار قدرتمند است[۱۹].

در Scikit-Learn، ما می‌توانیم این نوع روش SVM را به‌سادگی با تغییر هستهٔ خطی خود به هستهٔ RBF (تابع پایهٔ شعاعی) با استفاده از ابرپارامتر kernel پیاده‌سازی کنیم:

```
clf=SVC(kernel='rbf', C=1E6)
clf.fit(X,y)
```

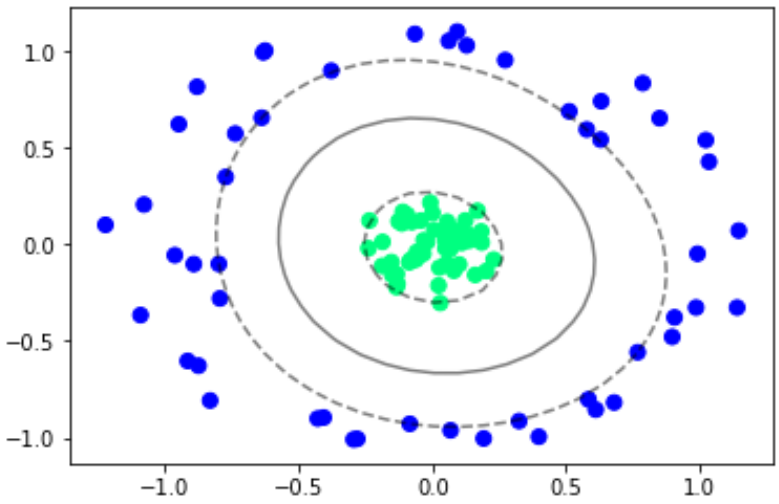
بعد از اجرا خواهیم داشت:

```
SVC(C=1000000.0, cache_size=200, class_weight=None,
coef0=0.0,decision_function_shape='ovr', degree=3,
gamma='auto_deprecated',kernel='rbf', max_iter=-1,
probability=False, random_state=None,shrinking=True,
tol=0.001, verbose=False)
```

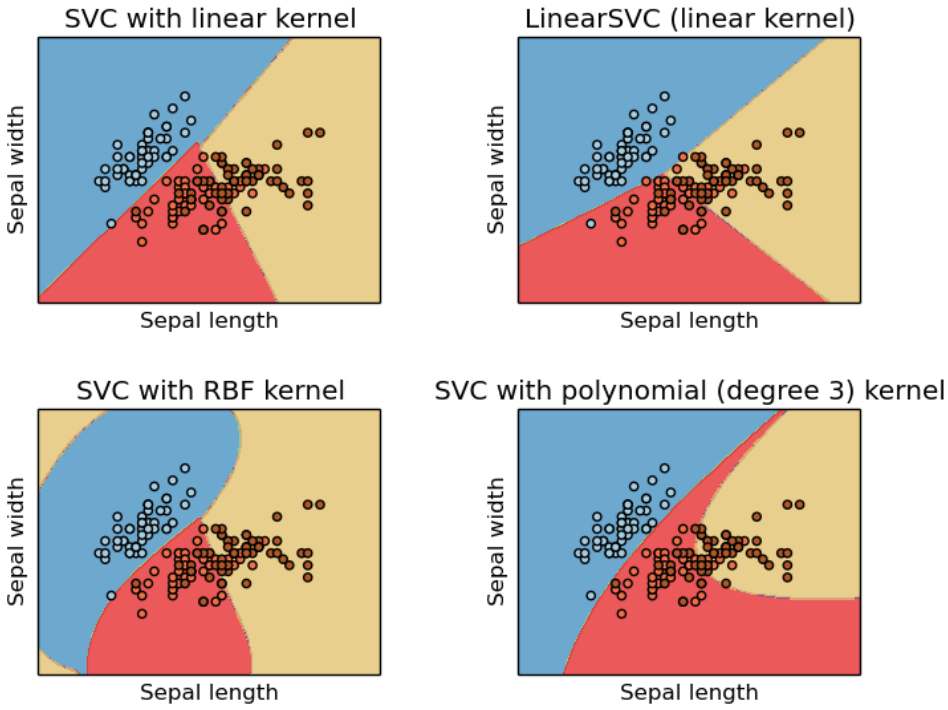
و درنهایت رسم تابع تصمیم:

```
plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap='winter')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:,0],clf.support_v
ectors_[:,1],s=300,linewidth=1, facecolors='none')
```

47. kernel transformation
48. kernel trick



شکل ۲-۲۲: جداسازی دو طبقه از داده‌ها با SVM غیرخطی با کرنل RBF

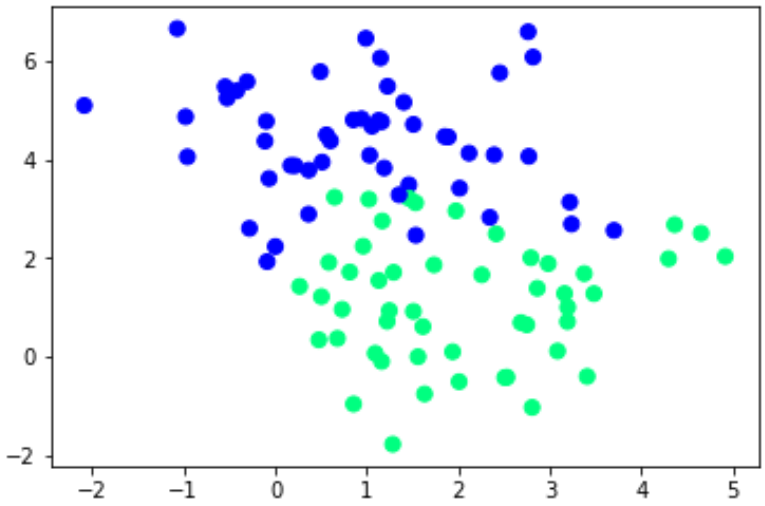


شکل ۲-۲۳: جداسازی مجموعه دادهٔ `iris` با استفاده از کرنل‌های مختلف در روش SVM [۱۹]

۲-۳-۲- تنظیم SVM: نرم کردن حاشیه‌ها[۴۹]

بحث ما تاکنون حول محور مجموعه داده‌های قابل‌تفکیک متمرکز شده است، که در آن یک مرز تصمیم کامل وجود دارد؛ اما اگر داده مقداری همپوشانی داشته باشد، چه می‌شود؟ به‌عنوان‌مثال، ممکن است داده‌هایی مانند شکل ۲-۲۴ داشته باشید که با تابع `make_blobs` تولید شده‌اند:

```
from sklearn.datasets.samples_generator import make_blobs
X,y=make_blobs(n_samples=100, centers=2,
random_state=0, cluster_std=1.2)
plt.scatter(X[:,0], X[:,1], c=y, s=50,
cmap='winter')
```



شکل ۲-۲۴: مجموعه داده‌ای که دو دسته از داده‌های آن با هم همپوشانی دارند.

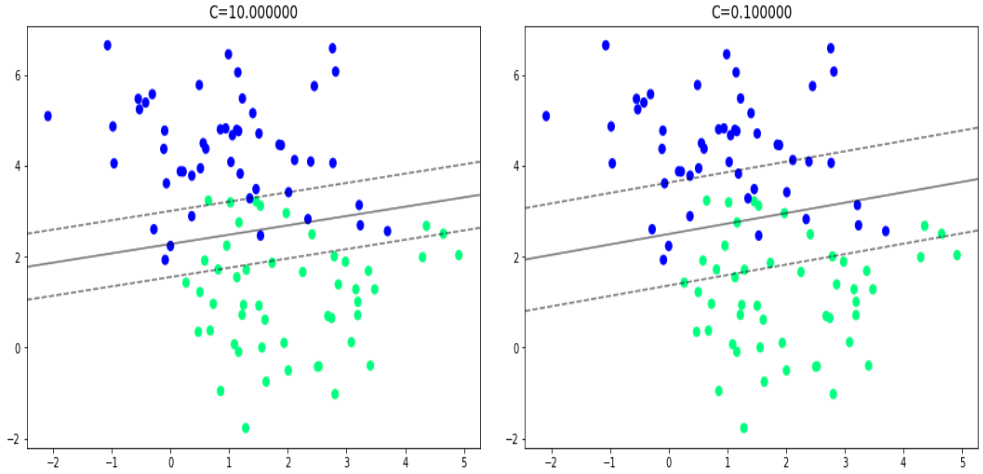
برای رسیدگی به این مورد، بهینه‌سازی مسئلهٔ SVM دارای یک فاکتور تصحیح[۵۰] است که حاشیه را «نرم»[۵۱] می‌کند؛ یعنی برای اینکه نتیجه مناسب‌تر شود این اجازه به بعضی از نقاط داده می‌شود که درون حاشیه حرکت کنند. سختی حاشیه با یک پارامتر تنظیم،[۵۲] کنترل می‌شود، که اغلب با C نمایش داده می‌شود. وقتی مقدار C بسیار بزرگ باشد، یعنی حاشیه سخت است و نقاط نمی‌توانند در آن قرار بگیرند. برای مقدار C کوچک‌تر، یعنی حاشیه نرم‌تر است و اجازه حرکت برخی نقاط درون حاشیه را می‌دهد[۱۹].

---

49. Softening Margins
50. fudge
51. softens
52. tuning parameter

طرحی که در شکل ۲-۲۵ پیاده‌سازی شده است، تصویری از چگونگی تأثیر پارامتر C بر حاشیهٔ مناسب نهایی، (از طریق نرم شدن آن) ارائه می‌دهد. با استفاده از تابع subplots در پارامترهای اول و دوم آن مشخص می‌کنیم که در fig (شکل) ما چند ردیف یا سطر وجود دارد و در هر ردیف چند نمودار وجود دارد. ما مشخص کرده‌ایم که در شکل ما در 1 ردیف 2 نمودار قرار گرفته است و سایز شکل 16X6 است. سپس تنظیمات مربوط به موقعیت نمودارها را با subplot_adjust انجام می‌دهیم. به‌عنوان‌مثال، پارامتر wspace مشخص کنندهٔ فاصلهٔ بین دو نمودار است. سپس در یک حلقهٔ for تابع تصمیم را به‌ازای دو مقدار 0.1 و 10 برای پارامتر C، در دو نمودار رسم می‌کنیم:

```
fig, ax=plt.subplots(1,2,figsize=(16,6))
fig.subplots_adjust(left=0.0625, right=0.95,
wspace=0.1)

for axi,C in zip(ax,[10,0.1]):
  model=SVC(kernel='linear',C=C).fit(X,y)
  axi.scatter(X[:,0],X[:,1],c=y,s=50,cmap='winter')
plot_svc_decision_function(model,axi)
axi.scatter(model.support_vectors_[:,0],
            model.support_vectors_[:,1],
            s=500,lw=1,facecolors='none')
axi.set_title('C={0:1f}'.format(C),size=14)
```



شکل ۲-۲۵: چگونگی تأثیر پارامتر C بر حاشیهٔ مناسب نهایی در دسته‌بندی با روش SVM خطی

۲-۴- قضیهٔ بیز[۵۳]

روشی برای دسته‌بندی پدیده‌ها، بر پایهٔ احتمال وقوع یا عدم وقوع یک پدیده است و در نظریه احتمالات بااهمیت و پرکاربرد است. اگر برای فضای نمونهٔ مفروضی با دانستن اینکه کدام‌یک از پیشامدهای افراز شده رخ داده است، بتوانیم افرازی انتخاب کنیم، بخش مهمی از عدم قطعیت تقلیل می‌یابد.

این قضیه از آن جهت مفید است که می‌توان از طریق آن، احتمال یک پیشامد را با مشروط کردن نسبت به وقوع یا عدم وقوع یک پیشامد دیگر محاسبه کرد. در بسیاری از حالت‌ها، محاسبهٔ احتمال یک پیشامد به‌صورت مستقیم کاری دشوار است. با استفاده از این قضیه و مشروط کردن پیشامد موردنظر نسبت به پیشامد دیگر، می‌توان احتمال موردنظر را محاسبه کرد [۲۰].

این رابطه به خاطر بزرگداشت توماس بیز، فیلسوف انگلیسی، به نام فرمول بیز معروف است.

در یادگیری ماشین، طبقه‌بند بیز، یک طبقه‌بند احتمالی ساده است که مبتنی بر کاربرد قضیه بیز است. مدل ویژگی مورد استفاده به‌وسیلهٔ طبقه‌بند سادهٔ بیز، فرضیات مستقل قوی[۵۴] را ایجاد می‌کند. این بدان معناست که وجود ویژگی خاص یک کلاس یا دسته، مستقل یا بی‌ارتباط با وجود هر ویژگی دیگر است [۲۱].

تعریف پیشامدهای مستقل:[۵۵]

دو پیشامد E و F مستقل هستند، اگر احتمال E و F مثبت بوده و نیز $P(E/F)=P(E)$ و $P(F/E)=P(F)$. همان‌طور که در تعریف خود بیان کردیم، طبقه‌بند بیز ساده، مبتنی بر قضیه بیز است. قضیه بیز نیز مبتنی بر احتمال شرطی است. در ادامه به آن خواهیم پرداخت [۲۲]:

تعریف احتمال شرطی:

احتمال شرطی A به شرط B با $P(A/B)$ نشان داده می‌شود:

$$P(A/B)=P(AB)/P(B) \quad (۲۷)$$

$$P(AB)=P(B)\,P(A/B) \quad (۲۸)$$

که به آن قانون ضرب احتمال‌ها می‌گوییم. به همین نحو، احتمال شرطی B به شرط A را می‌توان به‌صورت عبارت (۲۹) بیان کرد:

$$P(B/A)=P(AB)/P(A) \quad (۲۹)$$

---

53. Bayes
54. strong independence assumptions
55. independent events

که منجر به رابطهٔ $P(AB) = P(A)\,P(B|A)$ می‌شود. بنابراین قانون ضرب احتمال‌ها این تساوی را بیان می‌کند که حاصل‌ضرب احتمال شرطی یک پیشامد در احتمال شرطی پیشامد دیگر، برابر است با احتمال اشتراک آن دو پیشامد.

احتمال شرطی برای دو پیشامد مستقل

اگر دو پیشامد A و B مستقل باشند آنگاه احتمال شرطی به‌صورت زیر است:

$$P(A|B)=P(A) \quad (۳۰)$$

احتمال‌های شرطی (۳۱-۳۲)، هم‌عرض عبارت (۳۰) هستند:

$$P(B|A) = P(B) \quad (۳۱)$$

$$P(AB) = P(A)*P(B) \quad (۳۲)$$

با توجه به‌شرط استقلال اگر آزمایشی مرکب از دو قسمت فیزیکی مستقل و نامربوط به هم باشد (پیشامد A و Bبه قسمت‌های جداگانه آن آزمایش مربوط شوند)، به پیشامد AB احتمال $P(AB) = P(A)*P(B)$ را نسبت می‌دهیم.

قضیه بیز به‌صورت زیر است:

$$P(B_1|A) = P(B_1)P(A|B_1)/\sum_j P(B_j)P(A|B_j) \quad (۳۳)$$

۲-۴-۱- طبقه‌بند بیز [۵۶]

فرض کنید بردار مشاهدهٔ $\mathbf{X}$ در اختیار باشد و هدف ما انتساب آن به یکی از دو دستهٔ ω_1 و ω_2 باشد. از دیدگاه آماری ما به‌دنبال معیاری برای دسته‌بندی $\mathbf{X}$ به محتمل‌ترین دسته هستیم. با این تعبیر معیاری معقول برای انجام این دسته‌بندی، استفاده از $P(\omega_1|\underline{X})$ و $P(\omega_2|\underline{X})$ احتمالات پسین دو دستهٔ ω_1 و ω_2 است. $P(\omega_i|\underline{X})$ احتمال تعلق $\underline{X}$ به دستهٔ ω_i است، وقتی که بردار $\underline{X}$ مشاهده شده باشد. بنابراین برای انتخاب محتمل‌ترین دسته کافی است این احتمال‌ها را با یکدیگر مقایسه کنیم:

$$\begin{cases} P(w_1|\underline{X}) > P(w_2|\underline{X}) \rightarrow \underline{X} \in w_1 \\ P(w_1|\underline{X}) < P(w_2|\underline{X}) \rightarrow \underline{X} \in w_2 \end{cases} \quad (۳۴)$$

[۵۶]. Bayes Classifier

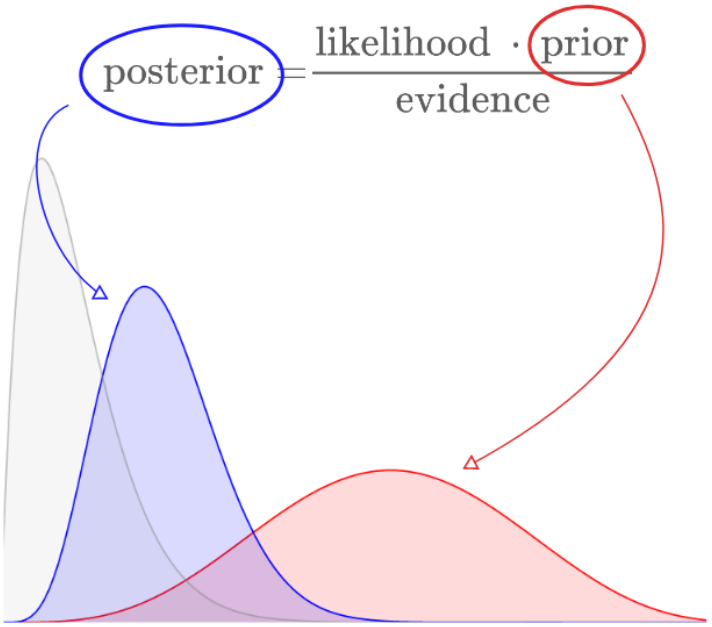
بنابر قضیهٔ بیز احتمال‌های پسین $P(\omega_1|\underline{X})$ و $P(\omega_2|\underline{X})$ را می‌توان به احتمال‌های پیشین و تابع درست‌نمایی دسته‌ها مرتبط ساخت که در شکل ۲-۲۶ نیز مشخص است:

$$P(w_i|\underline{X}) = \frac{P(w_i)P(\underline{X}|w_i)}{P(\underline{X})} \qquad (۳۵)$$

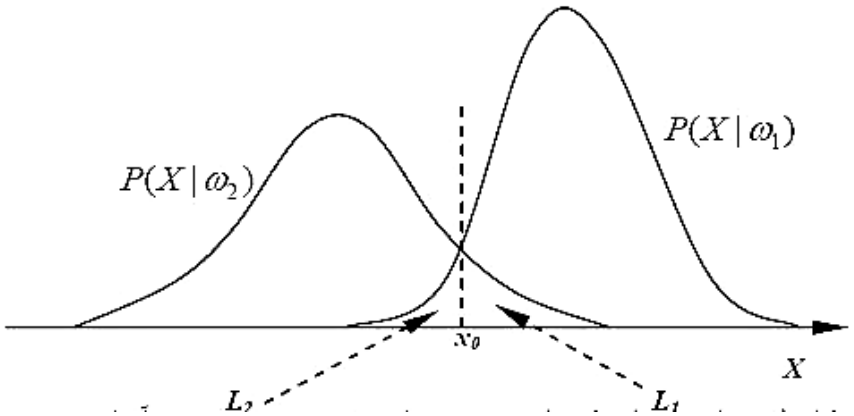
که در این عبارت $P(\omega_i)$ احتمال وقوع دستهٔ ω_i، یا احتمال پیشین دستهٔ iام، $P(\underline{X})$ احتمال وقوع بردار $\underline{X}$ و $P(\underline{X}|\omega_i)$ احتمال مشاهدهٔ بردار $\underline{X}$ است، وقتی که می‌دانیم بردار مذکور به دستهٔ ω_i تعلق دارد، که به آن اصطلاحاً تابع درست نمایی دستهٔ iام نسبت به $\underline{X}$ گفته می‌شود. حال اگر روابط ۳۴ و ۳۵ را با یکدیگر ترکیب کنیم، به معیار مقایسهٔ زیر می‌رسیم:

$$l(\underline{X}) \equiv \frac{P(\underline{X}|w_1)}{P(\underline{X}|w_2)} \quad \cdot \quad \begin{cases} l(\underline{X}) > \frac{P(w_2)}{P(w_1)} \rightarrow X \in w_1 \\ l(\underline{X}) < \frac{P(w_2)}{P(w_1)} \rightarrow X \in w_2 \end{cases} \qquad (۳۶)$$

که $l(\underline{X})$ را نسبت درست‌نمایی و نسبت $\frac{P(\omega_2)}{P(\omega_1)}$ را مرز مقایسه می‌نامند. در شکل ۲-۲۷ نمونهٔ‌یک‌بعدی از این معیار مشاهده می‌شود.



شکل ۲-۲۶: ارتباط احتمال پسین با احتمال پیشین و تابع درست‌نمایی [۲۳]



شکل ۲-۲۷: نمونه‌ای از توابع درست‌نمایی دو دسته و مرز بین آن‌ها [۲۴]

مطابق شکل۲-۲۷، x_0 مرز تساوی توابع درست‌نمایی $P(X|\omega_1)$ و $P(X|\omega_2)$ است. با توجه به اینکه اگر $X > x_0$، X جزو دستهٔ ω_1 محسوب شود و اگر $X < x_0$، X جزو دستهٔ ω_2 به حساب آید. در عمل راحت‌تر است که با منفی لگاریتم $l(\underline{X})$ کار کنیم که تابع تفکیک خوانده می‌شود:

$$h_{1,2}(\underline{X}) \equiv -\ln l(\underline{X}) = \ln P(\underline{X}|w_2) - \ln P(\underline{X}|w_1) \qquad (۳۷)$$

در ادامه خواهیم دید که این فرم از رابطه در صورت گوسی بودن توابع توزیع، ما را به یک تابع تفکیک خطی می‌رساند. طبیعتاً در مواردی که دو دستهٔ موردنظر هم احتمال باشند $P(\omega_1) = P(\omega_2)$، مرز مقایسهٔ $h_{1,2}(\underline{X})$ عدد صفر خواهد بود. بدین ترتیب $h(\underline{X})$ نمونه‌ای از یک تابع تفکیک است. روابط ۳۴، ۳۶ و ۳۷ صورت‌های مختلف معیار بیز محسوب می‌شوند و تفکیک‌کننده‌ای را که بر اساس آن طراحی می‌شود، طبقه‌بند بیز می‌گویند.

همچنین ملاحظه می‌شود که در انتساب $\underline{X}$ به یکی از دو دستهٔ ω_1 و ω_2 چهار حالت زیر متصور است:

- $\underline{X}$ متعلق به دستهٔ ω_1 باشد و ما هم آن را به ω_1 نسبت دهیم.
- $\underline{X}$ متعلق به دستهٔ ω_1 باشد و ما آن را به ω_2 نسبت دهیم.
- $\underline{X}$ متعلق به دستهٔ ω_2 باشد و ما آن را به ω_1 نسبت دهیم.
- $\underline{X}$ متعلق به دستهٔ ω_2 باشد و ما هم آن را به ω_2 نسبت دهیم.

از این میان تنها حالات اول و چهارم مطلوب ما هستند، که در آن‌ها دسته‌بندی درست انجام شده است؛ اما وقتی از معیارهای آماری همچون معیار بیز برای دسته‌بندی استفاده می‌کنیم بسته به شکل $P(\omega_1|\underline{X})$ و $P(\omega_2|\underline{X})$ و میزان همپوشانی‌شان، بروز حالات ۲ و ۳ اجتناب‌ناپذیر هستند.

۲-۴-۱-۱- طبقه‌بند بیز ساده[۵۷]

مدل‌های بیز ساده گروهی از الگوریتم‌های طبقه‌بندی بسیار سریع و ساده هستند که اغلب برای داده‌های با ابعاد بالا مناسب هستند. با توجه به سرعت بالای این روش و تعداد کم پارامترهای قابل‌تنظیم آن، به‌عنوان یک روش سریع و ارزان برای مسائل طبقه‌بندی بسیار مفید هستند. در این بخش شرحی در خصوص نحوهٔ کار طبقه‌بند بیز ساده و به‌دنبال آن چند نمونه از آن‌ها در عملکرد برخی از مجموعه داده‌ها ارائه می‌شود [۱۹].

طبقه‌بندهای بیز ساده بر اساس روش طبقه‌بندی بیزین[۵۸] ساخته شده‌اند. در طبقه‌بندی بیزی، با توجه به برخی از ویژگی‌های مشاهده شده، احتمال یک برچسب را پیدا می‌کنیم، که می‌توانیم آن را به‌صورت $\mathrm{P}(L|features)$ بنویسیم و آن را به‌صورت زیر می‌توانیم محاسبه کنیم:

$$\mathrm{P}(L|features) = \frac{P(features|L)P(L)}{P(features)} \qquad (۳۸)$$

اگر می‌خواهیم بین دو برچسب تصمیم بگیریم (بیایید آن‌ها را L_1 و L_2 بنامیم)، یکی از راه‌های تصمیم‌گیری، محاسبهٔ نسبت احتمالات پسین[۵۹] برای هر برچسب است:

$$\frac{\mathrm{P}(L_1|features)}{\mathrm{P}(L_2|features)} = \frac{P(features|L_1)}{P(features|L_2)}\frac{P(L_1)}{P(L_2)} \qquad (۳۹)$$

تنها چیزی که اکنون به آن نیاز داریم مدلی است که با استفاده از آن بتوان $\mathrm{P}(features|L)$ را برای هر برچسب محاسبه کرد. چنین مدلی یک مدل مولد[۶۰] نامیده می‌شود، زیرا فرایند تصادفی فرضی را که داده‌ها را تولید می‌کند، مشخص می‌کند. تعیین این مدل مولد به‌ازای هر برچسب، بخش اصلی آموزش طبقه‌بند بیز است. نسخهٔ کلی مرحلهٔ آموزش این روش کار بسیار سختی است، اما می‌توانیم با استفاده از برخی فرضیات ساده در مورد شکل این مدل، آن را ساده‌تر کنیم.

اینجاست که "naive" در "naive Bayes" وارد می‌شود: اگر فرضیات بسیار ساده‌ای در خصوص مدل مولد به‌ازای هر برچسب انجام دهیم، ما می‌توانیم تقریبی از مدل مولد برای هر کلاس را پیدا کنیم و سپس با طبقه‌بندی بیز ادامه دهیم. انواع مختلف طبقه‌بند سادهٔ بیز برمبنای فرضیات ساده‌ای دربارهٔ داده‌ها استوار هستند و ما در بخش‌های بعدی چند مورد از آن‌ها را بررسی خواهیم کرد.

---

57. Naive Bayes classifiers
58. Bayesian classification
59. posterior probabilities
60. generative model

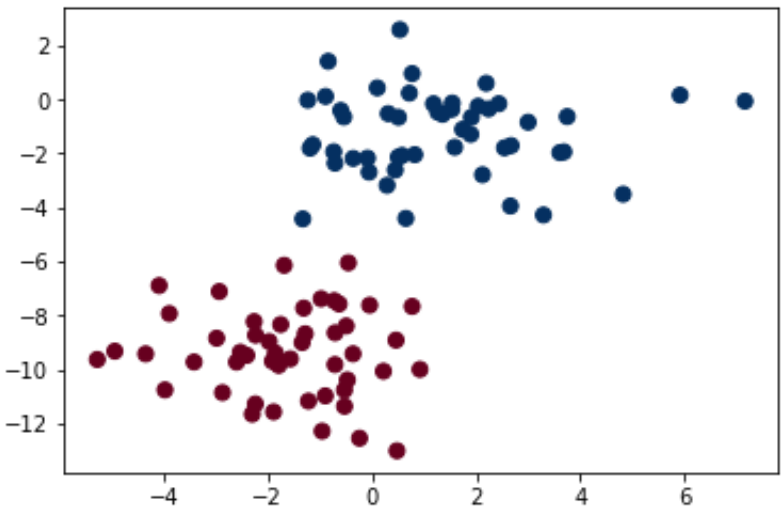
با افزودن کتابخانه‌های موردنیاز شروع می‌کنیم:

```
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap
import numpy as np
import seaborn as sns
```

۲-۴-۱-۲- بیز سادهٔ گاوسی[۶۱]

شاید آسان‌ترین راه درک طبقه‌بند سادهٔ بیز، بیز سادهٔ گاوسی باشد. در این طبقه‌بندی فرض بر این است که داده‌های هر کلاس از توزیع سادهٔ گاوسی به دست آمده است. حال فرض کنید داده‌های زیر را دارید:

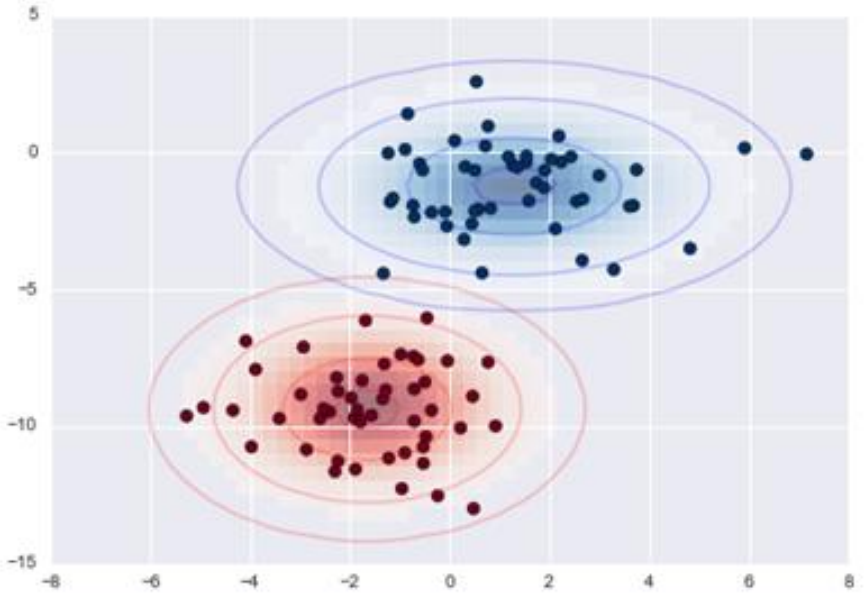
```
from sklearn.datasets import make_blobs
X,y=make_blobs(n_samples=100, centers=2,
               random_state=2, cluster_std=1.5)
plt.scatter(X[:,0],X[:,1],c=y,s=50, cmap='RdBu')
plt.show()
```



شکل ۲-۲۸: داده‌های تولید شده برای استفاده در روش دسته‌بندی بیز گاوسی

[61]. Gaussian naive Bayes

یک راه بسیار سریع برای ایجاد یک مدل ساده این است که فرض کنید داده‌ها با توزیع گاوسی و بدون همبستگی[۶۲] بین ابعاد آن توصیف شده است. این مدل می‌تواند به‌سادگی با پیدا کردن میانگین و انحراف معیار از نقاط موجود در هر کلاس نتایج مناسبی را ایجاد کند؛ یعنی تمام آنچه برای تعریف چنین توزیع نیاز دارید. نتیجهٔ این فرض گاوسی ساده در شکل ۲-۲۹ نشان داده شده است:



شکل ۲-۲۹: ایجاد مدل گاوسی از روی میانگین و انحراف معیار داده‌های هر دسته از نقاط برای جدا کردن نمونه‌ها

بیضی‌ها در اینجا نشانگر مدل مولد گاوسی برای هر کلاس، با احتمال بیشتر متمایل به مرکز آن‌ها هستند. با استفاده از این مدل مولد برای هر کلاس، دستورالعمل ساده‌ای برای محاسبهٔ احتمال $P(features|L_i)$

برای هر نقطه داریم، بنابراین می‌توانیم به‌سرعت نسبت پسین را محاسبه کرده و مشخص کنیم کدام کلاس، محتمل‌ترین گزینه برای یک نقطهٔ معین است.

این روش در کلاس `sklearn.naive_bayes.GaussianNB` پیاده‌سازی شده است:

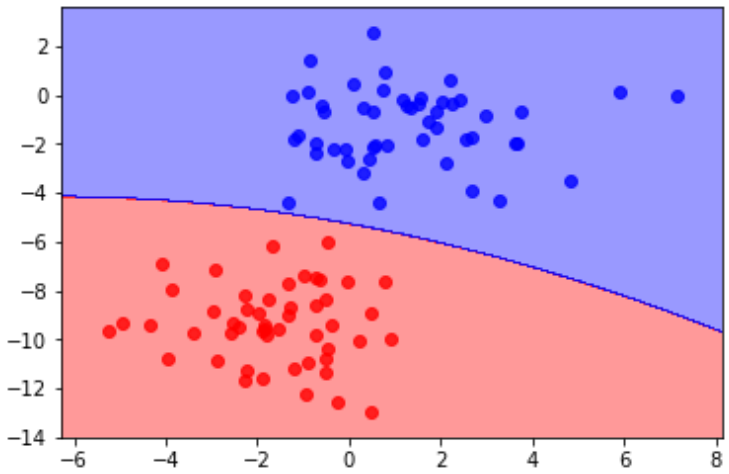
```
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(X,y)
```

---

62. covariance

اکنون با فراخوانی تابع plot_decision_regions که در بخش ۳-۵ (درخت تصمیم) به‌صورت کامل پیاده‌سازی شده است، مرز بین دو دسته که با مدل بیز سادهٔ گاوسی تعیین شده است رسم می‌شود (شکل ۲-۳۰):

```
plot_decision_regions(X,y, model)
```



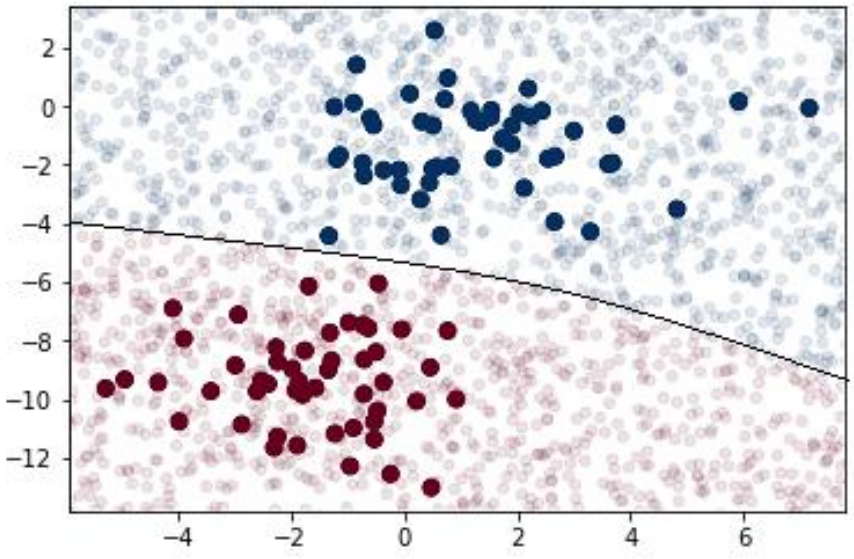
شکل ۲-۳۰: دسته‌بندی نقاط تولید شده با طبقه‌بند بیز سادهٔ گاوسی

حال بیایید داده‌های جدیدی تولید کنیم و برچسبشان را پیش‌بینی کنیم. همان‌طور که قبلاً هم عنوان شد، با ارسال عدد صحیح به RandomState می‌توانیم با هر بار اجرای برنامه همان اعداد تصادفی را تولید کنیم:

```
rng=np.random.RandomState(0)
Xnew=[-6,-14]+[14,18]*rng.rand(2000,2)
ynew=model.predict(Xnew)
```

اکنون می‌توانیم داده‌های جدید را برای تصور مرز تصمیم آن‌ها ترسیم کنیم. برای اینکه در تصویر داده‌های آزمون از وضوح کمتری نسبت به داده‌های آموزش برخوردار باشند، هنگام ترسیم داده‌های آزمون، میزان alpha را 0.1 قرار داده‌ایم. اندازهٔ قلم را هم برای ترسیم داده‌های آزمون کم کرده‌ایم (s=20):

```
plt.scatter(X[:,0],X[:,1],c=y,s=50,cmap='RdBu')
lim=plt.axis()
plt.scatter(Xnew[:,0],Xnew[:,1],c=ynew,
            s=20,cmap='RdBu',alpha=0.1)
plt.axis(lim)
```



شکل ۲-۳۱: دسته‌بندی مجموعه دادهٔ آموزش و آزمون تولید شده با طبقه‌بند بیز سادهٔ گاوسی

در اشکال ۲-۳۰ و ۲-۳۱ می‌بینید که مرز تصمیم برای این داده‌ها کمی انحنا پیدا کرده است؛ اما به‌طورکلی، مرز تصمیم در بیز سادهٔ گاوسی، درجه یک است.

نکتهٔ خوب روش بیزی این است که به‌طور طبیعی امکان طبقه‌بندی احتمالی را فراهم می‌کند، که می‌توانیم با استفاده از تابع `predict_proba` آن را محاسبه کنیم. برای سهولت در نمایش ۸ پیش‌بینی آخر را تا دو رقم اعشار گرد می‌کنیم و نمایش می‌دهیم:

```
yprob=model.predict_proba(Xnew)
yprob[-8:].round(2)
```

در خروجی خواهیم داشت:

```
array([[0.89, 0.11],
       [1., 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [0.  , 1.  ],
       [0.15, 0.85]])
```

در ماتریس بالا، ستون‌ها به ترتیب احتمال پسین برچسب اول و دوم را نشان می‌دهند. اگر به‌دنبال تخمین عدم قطعیت در طبقه‌بندی خود هستید، این‌گونه رویکردهای بیزی می‌توانند روش مفیدی باشند.

البته طبقه‌بندی نهایی تنها برمبنای درستی فرضیات مدل، منجر به نتایج خوب خواهد شد، به همین دلیل بیز سادهٔ گاوسی اغلب نتایج بسیار خوبی را به همراه ندارد. با وجود این، در بسیاری از موارد – به‌ویژه با افزایش تعداد ویژگی‌ها– بیز سادهٔ گاوسی می‌تواند روش مفیدی باشد.

۲-۴-۱-۳- بیز سادهٔ چندجمله‌ای[۶۳]

در بیز سادهٔ چندجمله‌ای فرض می‌شود ویژگی‌ها از یک توزیع چندجمله‌ای ساده ساخته شده‌اند. توزیع چندجمله‌ای احتمال مشاهدهٔ تعدادی از نقاط را در بین تعدادی از دسته‌ها توصیف می‌کند، بنابراین می‌توان گفت که بیز سادهٔ چندجمله‌ای مناسب‌ترین روش برای ویژگی‌هایی است که تعداد یا نرخ شمارش را نمایش می‌دهند. ایدهٔ این روش دقیقاً مشابه گذشته است، به‌جز اینکه به‌جای مدل‌سازی توزیع داده با روش گاوسی، آن را با توزیع چندجمله‌ای مدل می‌کنیم.

✓ **مثال:**

اغلب از روش بیز سادهٔ چندجمله‌ای در مسائل دسته‌بندی متون استفاده می‌شود، جایی که اسناد باید بر اساس ویژگی‌هایی مرتبط با تعداد یا فرکانس کلمات طبقه‌بندی شوند. در اینجا ما از ویژگی‌های شمارش تعداد کلمات پراکنده[۶۴] از ۲۰ گروه خبری استفاده خواهیم کرد تا این اسناد کوتاه را طبقه‌بندی کنیم [۱۹]. از مجموعه داده‌های کتابخانهٔ `sklearn`، مجموعه داده موردنظر را فراخوانی می‌کنیم.

```
from sklearn.datasets import fetch_20newsgroups
data=fetch_20newsgroups()
data.target_names
```

اسامی برچسب‌های این مجموعه داده در خروجی چاپ شده است:

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
```

63. Multinomial Naive Bayes
64. sparse word count

```
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

برای سادگی در اینجا ما فقط تعدادی از این دسته‌ها را انتخاب می‌کنیم و مجموعه آموزش و آزمون را بارگیری می‌کنیم:

```
categories = ['talk.religion.misc',
'soc.religion.christian',
'     sci.space', 'comp.graphics']
train=fetch_20newsgroups(subset='train',
categories=categories)
test=fetch_20newsgroups(subset='test',
categories=categories)
```

بخشی از داده‌ها در ادامه آمده است:

```
print(train.data[5])
```

در خروجی داریم:

```
From: dmcgee@uluhe.soest.hawaii.edu (Don McGee)
```

```
Subject: Federal Hearing

Originator: dmcgee@uluhe

Organization: School of Ocean and Earth Science and
Technology

Distribution: usa

Lines: 10


Fact or rumor....?  Madalyn Murray O'Hare an atheist
who eliminated the

use of the bible reading and prayer in public schools
15 years ago is now

going to appear before the FCC with a petition to stop
the reading of the

Gospel on the airways of America.  And she is also
campaigning to remove

Christmas programs, songs, etc from the public schools.
If it is true

then mail to Federal Communications Commission 1919 H
Street Washington DC

20054 expressing your opposition to her request.
Reference Petition number

2493.
```

به‌منظور استفاده از این داده‌ها برای یادگیری ماشین، باید بتوانیم محتوای هر رشته[۶۵] را به یک بردار اعداد تبدیل کنیم. برای این کار از بردار TF-IDF استفاده خواهیم کرد و پایپلاینی را ایجاد می‌کنیم که آن را به یک طبقه‌بند بیز سادهٔ چندجمله‌ای متصل می‌کند:

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.pipeline import make_pipeline


model=(TfidfVectorizer(), MultinomialNB())
```

[65]. String

حال با استفاده از این خط پایپلاین می‌توانیم مدل را روی داده‌های آموزش اعمال کرده و برچسب‌های داده‌های آزمون را پیش‌بینی کنیم:

```
model.fit(train.data, train.target)
labels=model.predict(test.data)
```

حال پس از پیش‌بینی برچسب داده‌های تست، می‌توانیم آن‌ها را ارزیابی کنیم تا از عملکرد این مدل مطلع شویم. در اینجا ماتریس درهم‌ریختگی[۶۶] برچسب‌های واقعی و پیش‌بینی شده داده‌های آزمون را با تابع `heatmap` به دست آوردیم. پارامتر `center` مقدار `float` را می‌پذیرد. این پارامتر محدودهٔ رنگ ماتریس را تغییر می‌دهد. پارامتر `annot`، درصورتی‌که `True` باشد، مقدار هر سلول را درون آن می‌نویسد. پارامتر `fmt`، فرمت کد رشته‌ای را بیان می‌کند، هنگامی‌که `annotation`ها اضافه می‌شوند. پارامتر `cbar`، درصورتی‌که `True` باشد، `colorbar` را رسم می‌کند:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
mat=confusion_matrix(test.target,labels)
sns.heatmap(mat.T, square=True, annot=True,
fmt='d', cbar=False,
xticklabels=train.target_names,
yticklabels=train.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

بعد از اجرا ماتریس درهم‌ریختگی را به‌صورت شکل ۲-۳۲ مشاهده می‌کنید.

بدیهی است، حتی این طبقه‌بندی بسیار ساده می‌تواند کلاس «مباحث فضایی»[۶۷] را با موفقیت از «مباحث کامپیوتری»[۶۸] جدا کند، اما بین «مباحث دینی»[۶۹] و «مباحث مسیحیت»[۷۰] گیج می‌شود و خطا دارد. این شاید یک بخش مورد انتظار از درهم‌ریختگی و سردرگمی باشد که در ماتریس مذکور مشهود است!

---

66. Confusion matrix
67. space talk
68. computer talk
69. Religion talk
70. Christianity talk

نکتهٔ بسیار جالب اینجاست که ما با استفاده از تابع پیش‌بینی `predict()`، ابزارهایی را برای تعیین دستهٔ هر رشته در اختیار داریم. در اینجا یک تابع مطلوبیت سریع[۷۱] وجود دارد که پیش‌بینی یک رشته را برمی‌گرداند:

```
def predict_category(s, train=train, model=model):
    pred=model.predict([s])
    return train.target_names[pred[0]]

predict_category('sending a payload to the ISS')
```

با فراخوانی تابع `predict_category` و ارسال رشتهٔ دلخواه به آن، در خروجی داریم:

```
'sci.space'
```

مجدداً رشتهٔ دلخواه دیگری به این تابع می‌فرستیم:

```
predict_category('discussing islam vs atheism')
```

در خروجی داریم:

```
'soc.religion.christian'
```

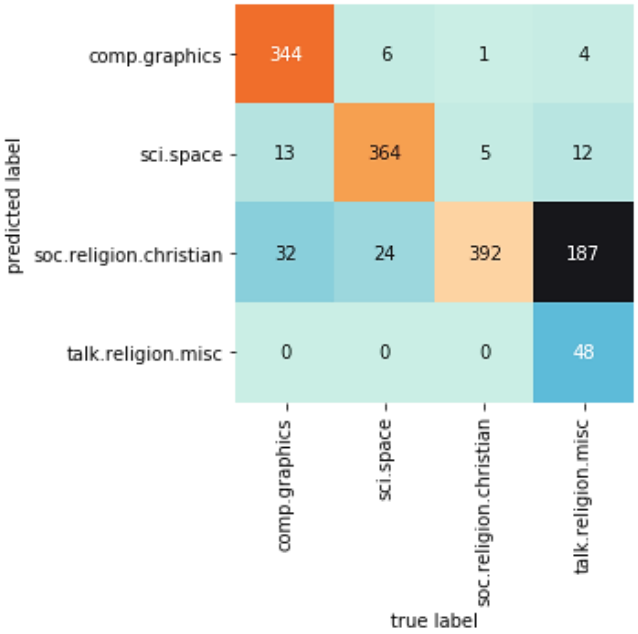
مجدداً رشتهٔ دلخواه دیگری به این تابع می‌فرستیم:

```
predict_category('determining the screen resolution')
```

در خروجی داریم:

```
'comp.graphics'
```

[71]. quick utility function



شکل ۲-۳۲: ماتریس درهم‌ریختگی دسته‌بندی متون با الگوریتم بیز سادهٔ چندجمله‌ای

به یاد داشته باشید که این چیزی پیچیده‌تر از یک مدل احتمال ساده برای (وزن) فرکانس هر کلمه در رشته نیست. بااین‌حال، نتایج به دست آمده قابل‌توجه است. حتی یک الگوریتم بسیار ساده، وقتی با دقت استفاده می‌شود و روی مجموعه بزرگی از داده‌های با ابعاد بالا آموزش می‌بیند، می‌تواند به طرز شگفت‌آوری مؤثر واقع شود.

ویژگی‌های بیز ساده:

از مزایای این روش عبارت‌اند از:

- این روش هم برای آموزش و هم پیش‌بینی بسیار سریع است.
- به‌راحتی قابل تفسیر است.
- پارامترهای قابل‌تنظیم (در صورت وجود) بسیار کمی دارد.

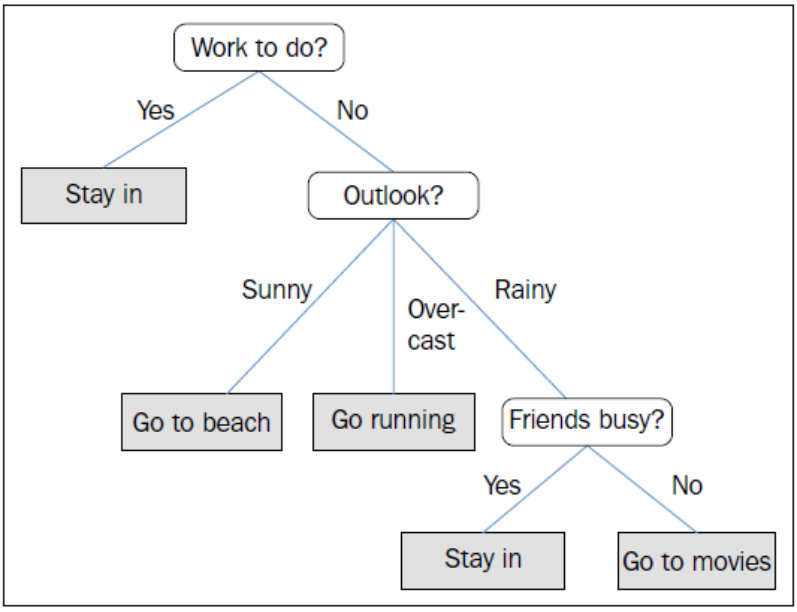
طبقه‌بند بیز ساده در یکی از موقعیت‌های زیر عملکرد خوبی دارد:

- هنگامی‌که مفروضات ساده با داده‌ها مطابقت دارند (در عمل بسیار نادر است).
- برای دسته‌های کاملاً منفک از هم، وقتی پیچیدگی مدل از اهمیت کمتری برخوردار است.
- برای داده‌های با ابعاد بالا، وقتی پیچیدگی مدل از اهمیت کمتری برخوردار است.

دو نکتهٔ آخر به نظر متفاوت می‌آیند، ولی درواقع مرتبط هستند: با افزایش ابعاد یک مجموعه داده، احتمال پیدا کردن هر دو نقطهٔ نزدیک به هم بسیار کمتر است. به‌طورکلی این بدان معناست که خوشه‌ها در داده‌های با ابعاد بالا از خوشه‌هایی با ابعاد کم، از هم جدا هستند، البته با فرض اینکه ابعاد جدید نویز نباشند و اطلاعات را اضافه کنند. به همین دلیل، طبقه‌بندهای ساده مانند بیز ساده با رشد ابعاد نتایج بهتری از طبقه‌بندهای پیچیده دارند. به‌طورکلی زمانی که اطلاعات کافی داشته باشید، حتی یک مدل ساده نیز می‌تواند بسیار قدرتمند باشد [۱۹].

۲–۵– درخت تصمیم[۷۲]

درخت تصمیم مدل طبقه‌بندی یا رگرسیون را به‌صورت ساختار یک درخت می‌سازد. به زبان ساده، درخت تصمیم بر اساس تصمیم‌گیری مبتنی بر تعدادی سؤال، داده‌ها را می‌شکند. برای درک بهتر این موضوع، شکل ۲–۳۳ را در نظر بگیرید.



شکل ۲–۳۳: مثال بسیار ساده‌ای از درخت تصمیم [۵]

بر اساس ویژگی‌های مجموعه دادهٔ آموزش، مدل درخت تصمیم یک سری از سؤالات را برای استنباط برچسب‌های دسته‌های نمونه‌ها یاد می‌گیرد. اگرچه شکل ۲–۳۳ مفهوم درخت تصمیم را بر اساس ویژگی‌های رده‌بندی‌شدهٔ کلامی نشان می‌دهد اما می‌توان ویژگی‌های مجموعه داده را اعداد واقعی مانند ویژگی‌های مجموعه داده‌های `Iris` در نظر گرفت [۵].

[72]. Decision tree

یک درخت تصمیم عمدتاً شامل یک گره ریشه،[۷۳] گره‌های داخلی[۷۴] و گره‌های برگ[۷۵] است که با شاخه‌ها به یکدیگر متصل می‌شوند.

برای ساختن درخت تصمیم در ابتدا باید ریشهٔ درخت را تعیین کنیم. ریشهٔ درخت یکی از ویژگی‌های درخت است که بیشترین اطلاعات (IG[۷۶]) را در مورد ویژگی هدف (برچسب) در اختیار ما قرار می‌دهد (نحوهٔ محاسبهٔ IG در ادامه مطرح می‌شود).

با استفاده از الگوریتم تصمیم‌گیری، ما از ریشه درخت شروع می‌کنیم و داده‌ها را روی آن تقسیم می‌کنیم و گره‌های فرزند پدیدار می‌شوند (گره‌های داخلی). سپس در یک روند تکراری، داده‌های مربوط به هر گره داخلی را روی آن تقسیم می‌کنیم تا زمانی که گره‌هایی ایجاد شوند که داده‌ها روی آن‌ها قابل‌تقسیم نباشند. این گره‌ها برگ‌های درخت هستند و داده‌ها روی برگ‌ها به یک دسته تعلق دارند.

بنابراین گره‌های برگ شامل پیش‌بینی‌های سیستم برای نمونه‌های جدید پرس‌وجو شده از مدل آموزش دیده هستند[۱۴].

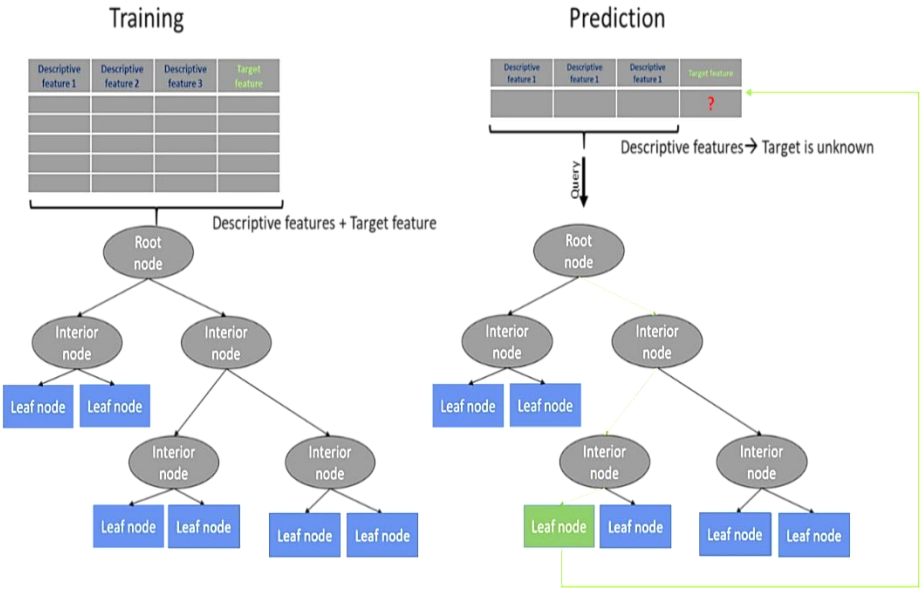
یک درخت عمیق با گره‌های زیاد، که به‌راحتی می‌تواند منجر به **آموزش بیش‌ازحد**[۷۷] شود. بنابراین، ما به‌طورمعمول با تعیین محدودیتی برای حداکثر عمق درخت، آن را هرس می‌کنیم [۵].

به زبان ساده، فرایند آموزش یک درخت تصمیم و پیش‌بینی ویژگی هدف از نمونهٔ مورد سؤال، به شرح زیر است:

- مجموعه داده‌ای را ارائه دهید که شامل تعدادی از نمونه‌های آموزشی است که با تعدادی از ویژگی‌های توصیفی و یک ویژگی هدف (برچسب) مشخص می‌شود.
- آموزش مدل با تقسیم مداوم داده‌ها در امتداد ویژگی‌هایی که بیشترین اطلاعات را در مورد ویژگی هدف ارائه می‌کنند.
- درخت را رشد دهید تا زمانی که شرایط توقف الگوریتم برقرار شود (گره‌های برگی ایجاد شوند که نشان‌دهنده پیش‌بینی‌های سیستم هستند).
- نمونه‌های پرس‌وجو شده را به درخت نشان دهید و تا زمانی که به گره‌های برگ برسید، شاخه‌های درخت را دنبال کنید.
- با رسیدن به گره برگ، ویژگی هدف برای نمونهٔ جدید پیش‌بینی شده است.

---

73. Root node
74. Interior node
75. Leaf node
76. Information gain
77. Overfitting



شکل ۲-۳۴: فرایند آموزش یک درخت تصمیم و پیش‌بینی ویژگی هدف از نمونهٔ مورد سؤال [۱۴]

۲-۵-۱- محاسبهٔ بیشترین کسب اطلاعات[۷۸] (IG)

برای پیدا کردن ویژگی که بیشترین اطلاعات را برای شکستن مجموعه داده در اختیار ما قرار می‌دهد، تابع IG را در هر نود چنین تعریف می‌کنیم:

$$IG(D_p.f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j) \qquad (۴۰)$$

در اینجا f ویژگی است که کاندیدای شکستن مجموعه داده است. D_p و D_j به ترتیب مجموعه داده‌های مربوط به گره‌های والد و اُمین فرزند هستند. I معیار اندازه‌گیری ناخالصی است. N_p تعداد نمونه‌های گره والد است. N_j تعداد نمونه‌های گره فرزند است. همان‌طور که در عبارت (۴۰) مشخص است، IG برابر است با تفاوت بین ناخالصی گره والد و مجموع ناخالصی‌های گره‌های فرزند. هرچه ناخالصی گره‌های فرزند کمتر باشد، IG گره والد بیشتر است؛ اگرچه برای سادگی و کاهش فضای جست‌وجو، بیشتر کتابخانه‌ها (مانند scikit-learn) درختان تصمیم دودویی را پیاده‌سازی کرده‌اند [۵]. به این معنا که هر گره والد به دو گره فرزند تقسیم می‌شود، D_{left} و D_{right}:

$$IG(D_p.f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right}) \qquad (۴۱)$$

78. Maximizing information gain

سه معیار ناخالصی (معیار تقسیم مجموعه داده) که در درختان تصمیم دودویی متداول است، برابر است با

- Gini impurity (I_G)
- Entropy (I_H)
- Classification error (I_E)

- **Entropy (I_H):**

$$I_H(t) = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t) \qquad (۲۸)$$

در اینجا $p(i|t)$ نسبت نمونه‌هایی است که به دستهٔ i تعلق دارند در گره t. بنابراین انتروپی برابر 0 است اگر همهٔ نمونه‌ها در یک گره درخت به یک کلاس تعلق داشته باشند و انتروپی بیشینه است اگر توزیع یکنواختی از دسته‌ها در گره t داشته باشیم [۵]. به‌عنوان‌مثال، در مسئله‌ای که با دو دسته از داده‌ها مواجهیم، انتروپی گره t است اگر $p(i=1|t)=1$ و $p(i=0|t)=0$ و انتروپی گره t، 1 است اگر $p(i=1|t)=0.5$ و $p(i=0|t)=0.5$ (اگر دسته‌ها در گره t به‌صورت یکنواخت توزیع شده باشند) [۵].

بنابراین IG گرهی که داده‌ها در آن به توزیع یکنواختی از دسته‌ها تعلق داشته باشند (انتروپی گره بیشینه باشد) و داده‌ها در گره‌های فرزند آن به یک دسته تعلق داشته باشند (کمترین انتروپی را داشته باشند)، IG آن گره، بیشینه است.

برای کسب اطلاعات بیشتر دربارهٔ روش‌های دیگر محاسبهٔ ناخالصی به [۵] مراجعه کنید.

✓ **مثال:**

به‌عنوان‌مثالی [۵] برای طبقه‌بندی درخت تصمیم، دسته‌بندی مجموعه دادهٔ iris را در نظر بگیرید. تابع train_test_split از ماژول sklearn.model_selection فراخوانی شده است و با ارسال نمونه‌ها به همراه ویژگی هدف (برچسب) و مشخص کردن اندازهٔ مجموعه آزمون برای این تابع، مجموعه دادهٔ آزمون به دست آمده است. سپس تابع plot_decision_regions برای رسم نواحی تصمیم تعریف شده است. در این تابع از کلاس ListedColormap برای رنگ‌بندی نمونه‌های هر دسته استفاده شده است (به تعداد اعداد یکتا در بردار هدف (تعداد دسته‌ها) از رنگ‌های این کلاس بهره‌برداری شده است). با استفاده از تابع contourf مرزهای مربوط به هر دسته مشخص و رنگ‌آمیزی می‌شوند. سپس در یک حلقهٔ for به‌ازای هر دسته، نمونه‌ها با رنگ مجزا با استفاده از تابع scatter رسم می‌شوند. در ادامه دستورهای مربوط به ترسیم مجموعه دادهٔ آزمون نوشته شده‌اند.

```
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
iris=datasets.load_iris()
X=iris.data[:,[2,3]]
y=iris.target


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3,
random_state=0)


def plot_decision_regions(X, y, classifier,
                test_idx=None,resolution=0.02):
    # setup marker generator and color map
    markers = ('^', '^', '^', '^', '^')
    colors = ('red', 'blue', 'lightgreen', 'gray',
    'cyan')
    cmap=ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface
    x1_min, x1_max = X[:, 0].min()-1,
    X[:,0].max()+1
    x2_min, x2_max = X[:, 1].min()-1,
    X[:,1].max()+1
    xx1,xx2=np.meshgrid(np.arange(x1_min,x1_max,res
    olution),
    np.arange(x2_min,x2_max,resolution))
```

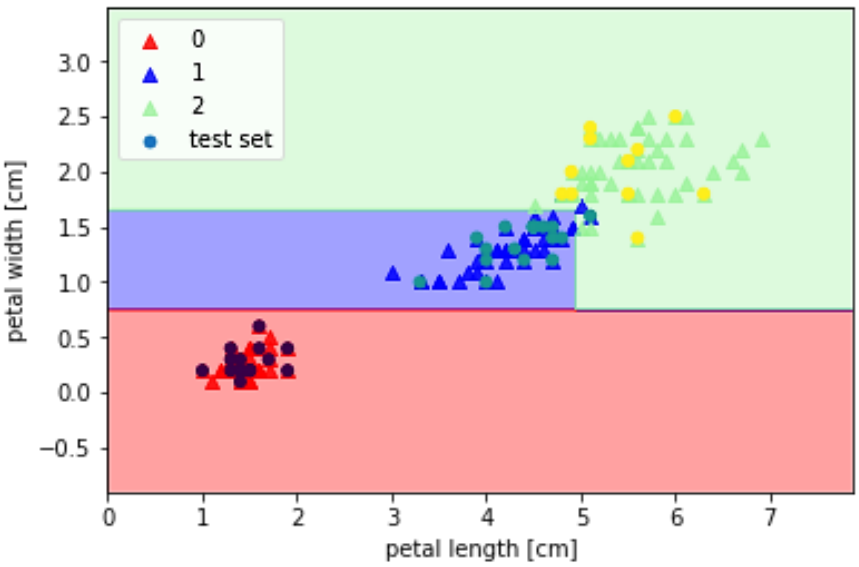
```
Z=classifier.predict(np.array([xx1.ravel(),xx2.
ravel()]).T)
Z=Z.reshape(xx1.shape)
plt.contourf(xx1,xx2,Z,alpha=0.4,cmap=cmap)
plt.xlim(xx1.min(),xx1.max())
plt.ylim(xx2.min(),xx2.max())
# plot all samples
for idx, cl in enumerate(np.unique(y)):
    plt.scatter(x=X[y==cl,0], y=X[y==cl,1],
    alpha=0.8, c=cmap(idx),
    marker=markers[idx], label=cl)

# highlight test samples
if test_idx:
    X_test,y_test=X[test_idx,:],y[test_idx]
    plt.scatter(X_test[:, 0], X_test[:, 1],
    c=y_test, alpha=1.0, linewidths=1,
    marker='o',
    s=25, label='test set')
```

در ادامه با فراخوانی کلاس `DecisionTreeClassifier` از کتابخانهٔ `sklearn`، مدل درخت تصمیم را می‌سازیم. برای جلوگیری از بیش برازش ماکزیمم عمق این درخت را برابر با 3 قرار می‌دهیم. با استفاده از تابع `fit` مدل را به مجموعه داده اعمال می‌کنیم و با فراخوانی تابع `plot_decision_regions` نواحی تصمیم‌گیری به دست آمده با مدل درخت تصمیم را به همراه نمونه‌ها رسم می‌کنیم:

```
from sklearn.tree import DecisionTreeClassifier
tree=DecisionTreeClassifier(criterion='entropy',
max_depth=3,random_state=0)
tree.fit(X_train,y_train)
```

```
X_combined=np.vstack((X_train,X_test))
y_combined=np.hstack((y_train,y_test))
plot_decision_regions(X_combined,y_combined,

classifier=tree,test_idx=range(105,150))
plt.xlabel('petal length [cm]')
plt.ylabel('petal width [cm]')
plt.legend(loc='upper left')
plt.show()
```

در شکل ۲-۳۵ نواحی مربوط به هر دسته کاملاً مشخص است. نمونه‌های مجموعه داده آموزش به‌صورت مثلث‌های کوچک با رنگ‌بندی متفاوت نمایش داده شده‌اند. نمونه‌های آزمون هم با علامت دایره در نواحی مختلف تصمیم مشخص هستند.



شکل ۲-۳۵: دسته‌بندی مجموعه دادهٔ iris با الگوریتم درخت تصمیم

فصل سوم

خوشه‌بندی داده با پایتون

PYTHON

فرض کنید مجموعه داده‌ای را در اختیار دارید که هر نمونه یا الگوی آن شامل تعدادی ویژگی است. می‌خواهیم این مجموعه داده را به یک الگوریتم یادگیری ماشین یاد بدهیم تا ماشین بتواند در صورت رویارویی با داده‌هایی از این جنس در آینده، به‌خوبی آن داده‌ها را تحلیل کند و بتواند در مورد آن‌ها تصمیم بگیرد. درصورتی‌که این مجموعه داده برچسب داشته باشد دست ما باز است. می‌توانیم از برچسب‌های مجموعه داده استفاده کنیم، به‌عبارت‌دیگر از یادگیری با ناظر بهره‌مند شویم یا می‌توانیم برچسب‌ها را در نظر نگیریم و از روش‌های بدون نیاز به ناظر مانند روش‌های خوشه‌بندی برای تصمیم‌گیری استفاده کنیم. اما زمانی که با مجموعه داده‌ای مواجه هستیم که برچسب ندارد، قطعاً از روش‌های بدون نیاز به ناظر استفاده می‌کنیم.

روش‌هایی که در این فصل مطرح می‌شوند:

- K-means
- Fuzzy c-means
- سلسله‌مراتبی

۳-۱- الگوریتم k-means

روش خوشه‌بندی k-means یک روش متداول برای خوشه‌بندی داده است و پیاده‌سازی آن بسیار آسان است. همان‌طور که از نام این روش مشخص است، در این روش k خوشه یا به عبارت بهتر k مرکز خوشه تعریف می‌شود. عدد k یک عدد صحیح است که کاربر تعیین می‌کند. پس مهم‌ترین ضعف این روش حساسیت آن به مقداردهی اولیه است و تضمینی برای یافتن بهینه سراسری وجود ندارد. در [۲۵] صورتی از روش kmeans ارائه شده است که رسیدن به بهینه سراسری را تضمین می‌کند. گفته می‌شود نمونهٔ j از مجموعه داده به خوشهٔ i متعلق است اگر فاصلهٔ آن نمونه تا مرکز خوشهٔ i نسبت به فاصلهٔ آن نمونه تا مراکز خوشه‌های دیگر کمتر باشد.

مراحل الگوریتم k-means:

- **مرحلهٔ ۱:** تعیین تعداد خوشه‌های مسئله یا به‌عبارت‌دیگر انتخاب عدد صحیح برای پارامتر k (بهتر است الگوریتم را با چندین مقدار متفاوت k اجرا کنید تا درنهایت مقداری را که برای مجموعه داده شما بهتر است پیدا کنید).
- **مرحلهٔ ۲:** مقداردهی اولیه به‌صورت تصادفی به مراکز خوشه‌ها
- **مرحلهٔ ۳:** تعیین نزدیک‌ترین مرکز خوشه به هر نمونه و درنتیجه تعیین خوشهٔ متناظر هر نمونه:

$$c_i = argmin_j \|x_i - \mu_j\|_2 \quad (۴۲)$$

در عبارت (۴۲) c_i معرف مرکز خوشهٔ متناظر نمونهٔ x_i است به‌گونه‌ای که فاصلهٔ (فاصلهٔ اقلیدسی) آن نمونه تا مرکز خوشه (μ_j) نسبت به مراکز خوشه‌های دیگر مینیمم است.

- **مرحلهٔ ۴:** محاسبهٔ مقدار میانگین نمونه‌های متعلق به هر خوشه و تعیین آن به‌عنوان مرکز جدید هر خوشه (به‌روزرسانی مراکز خوشه)
- **مرحلهٔ ۵:** تکرار الگوریتم از مرحلهٔ ۳ تا زمانی که همگرایی حاصل شود. می‌توان گفت هنگامی‌که مراکز خوشه‌ها تغییر نکنند، زمان توقف الگوریتم است.

حال اجرای این الگوریتم را با مثالی [۲۶] ادامه می‌دهیم:

✓ **مثال:**

فرض می‌کنیم که در مجموعه داده‌ای اطلاعات مشتریان شرکتی را داریم و می‌خواهیم مشتریان را بر اساس عملکردشان خوشه‌بندی کنیم، به‌گونه‌ای که مشتریانی که عملکرد مشابهی دارند در یک خوشه قرار بگیرند.

می‌توانیم مستقیم برای دانلود مجموعه دادهٔ مشتریان از پایتون استفاده کنیم. ابتدا `wget package` را نصب می‌کنیم:

اگر از `jupyter` در نرم‌افزار `anaconda` برای اجرای کدهای پایتون استفاده می‌کنید:

```
!pip install wget
```

یا در `command prompt`، ابتدا وارد فولدر `script`های پایتون نصب شده در سیستمتان شوید:

```
cd
C:\Users\APPLE\AppData\Local\Programs\Python\Python
37\Scripts
```

سپس دستور زیر را در `command prompt` وارد کنید:

```
pip install wget
```

حال `wget package` نصب شده است و می‌توانیم از طریق این بسته فایل‌های دلخواه را دانلود کنیم:

```
import wget
Cust_Segmentation=wget.download('https://s3-api.us-
geo.objectstorage.softlayer.net/cf-courses-
data/CognitiveClass/ML0101ENv3/labs/Cust_Segmentati
on.csv')
```

با استفاده از کتابخانهٔ `pandas` فایل مربوط به `dataset` را می‌خوانیم و چند سطر ابتدای مجموعه داده را با استفاده از تابع `head()`، به‌صورت یک `data frame` در خروجی مشاهده می‌کنیم. این مجموعه داده ۸۵۰ نمونه یا الگو (سطر) و ۱۰ ویژگی (ستون) دارد:

```
import pandas as pd
cust_df=pd.read_csv("Cust_Segmentation.csv")
cust_df.head()
```

جدول ۳-۱: چهار سطر از مجموعه دادهٔ مشتریان

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	Address	DebtIncomeRatio
0	1	41	2	6	19	0.124	1.073	0.0	NBA001	6.3
1	2	47	1	26	100	4.582	8.218	0.0	NBA021	12.8
2	3	33	2	10	57	6.111	5.802	1.0	NBA013	20.9
3	4	29	2	4	19	0.681	0.516	0.0	NBA009	6.3
4	5	47	1	31	253	9.308	8.908	0.0	NBA008	7.2

- **پیش‌پردازش**

همان‌طور که مشخص است، ویژگی آدرس مشتریان قابل‌محاسبه در فاصلهٔ اقلیدسی نیست بنابراین این ویژگی را حذف می‌کنیم:

```
df=cust_df.drop("Address",axis=1)
```

سپس ستون اول را که معرف شناسهٔ نمونه‌ها است، حذف می‌کنیم. مقادیر تعریف شدهٔ `nan` را با استفاده از تابع `nan_to_num` به عدد تبدیل می‌کنیم (در صورت `nan` بودن به `0` و در صورت بی‌نهایت بودن، به بزرگ‌ترین عدد تعریف شده). با توجه به اینکه مقادیر ویژگی‌های مختلف مجموعه داده در بازه‌های متفاوتی است، این مقادیر را با استفاده از کلاس `StandardScaler` از ماژول `sklearn.preprocessing` نرمال‌سازی می‌کنیم. در تابع `fit_transform()` از کلاس `StandardScaler` میانگین و انحراف از معیار همهٔ نمونه‌های آموزشی، به‌ازای هر ویژگی محاسبه می‌شود و معادل نرمال شدهٔ هر نمونه به‌صورت عبارت (۴۳) جایگزین هر نمونه می‌شود:

$$Z = \frac{x - mean}{standard\ deviation} \qquad (۴۳)$$

```
import numpy as np
from sklearn.preprocessing import StandardScaler
X=df.values[:,1:]
X=np.nan_to_num(X)
Clust_dataset=StandardScaler().fit_transform(X)
Clust_dataset
```

- **مدل‌سازی**

حال مجموعه داده برای اعمال الگوریتم kmeans آماده است. پارامتر init را با k-means++ مقداردهی اولیه می‌کنیم. در این پارامتر روش مقداردهی اولیه به مراکز خوشه‌ها مشخص می‌شود. روش "k-means++" روش هوشمندی است برای سریع‌تر رسیدن به همگرایی. پارامتر n_cluster تعداد خوشه را مشخص می‌کند و n_init تعداد دفعات اجرای الگوریتم با مقادیر اولیهٔ متفاوت برای مراکز خوشه‌ها را مشخص می‌کند و درنهایت بهترین اجرای الگوریتم به‌عنوان نتیجه در نظر گرفته می‌شود. این انتخاب برحسب مجموع مربعات فواصل نمونه‌ها از مراکز خوشه است. برای کسب اطلاعات بیشتر دربارهٔ پارامترهای روش KMeans به help(KMeans()) مراجعه کنید.

```
from sklearn.cluster import KMeans
clusterNum=3
k_means=KMeans(init="k-means++", n_clusters = clusterNum, n_init=12)
k_means.fit(X)
labels=k_means.labels_
print(labels)
```

شماره خوشه‌ها را به‌ازای هر نمونه در ادامه مشاهده می‌کنید:

```
[1 0 1 1 2 0 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 1 1 0 1
0 1 1 1 1 1 1
 1 1 0 1 0 1 2 1 0 1 1 1 0 0 1 1 0 0 1 1 1 0 1 0 1 0 0 1 1 0
1 1 1 0 0 0 1
 1 1 1 1 0 1 0 0 2 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1
1 1 1 1 1 0 1
...
1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1
1 1 1 1 0]
```

می‌توانیم برچسب خوشه‌ها را به‌عنوان ستونی به مجموعه داده اضافه کنیم:

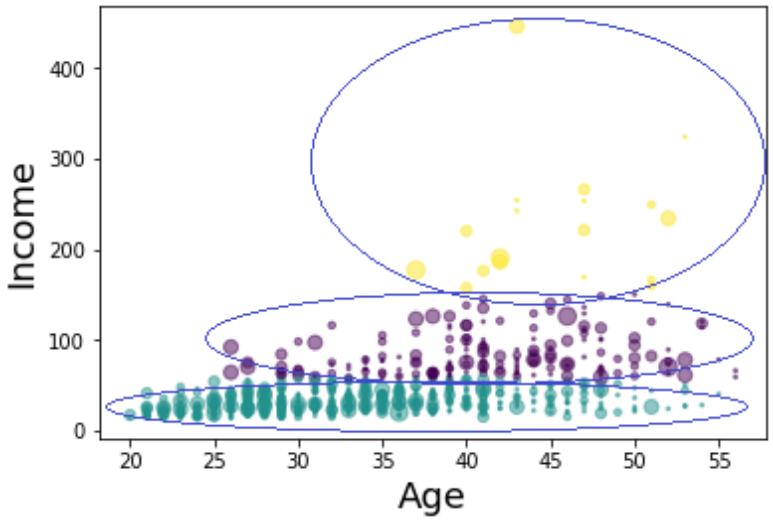
```
df["clust_km"]=labels
df.head()
```

جدول ۳-۲: مجموعه دادهٔ مشتریان بعد از افزودن برچسب خوشه‌ها

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio	Clus_km
0	1	41	2	6	19	0.124	1.073	0.0	6.3	0
1	2	47	1	26	100	4.582	8.218	0.0	12.8	2
2	3	33	2	10	57	6.111	5.802	1.0	20.9	0
3	4	29	2	4	19	0.681	0.516	0.0	6.3	0
4	5	47	1	31	253	9.308	8.908	0.0	7.2	1

می‌توان نتیجهٔ خوشه‌بندی را برحسب سه ویژگی Age و income و Edu به‌صورت دوبعدی نشان داد (شکل ۳-۱). میزان تحصیلات را به‌عنوان شعاع دایره‌ای در نظر می‌گیریم و مساحت آن را حساب می‌کنیم (تا به‌علت وجود توان دو، تغییرات Edu به‌ازای دو ویژگی Age و Income مشخص‌تر شود):

```
import matplotlib.pyplot as plt
area=np.pi*X[:,1]**2
plt.scatter(X[:,0],X[:,3],s=area,
c=labels.astype(np.float), alpha=0.5)
plt.xlabel('Age',fontsize=18)
plt.ylabel('Income',fontsize=18)
plt.show()
```



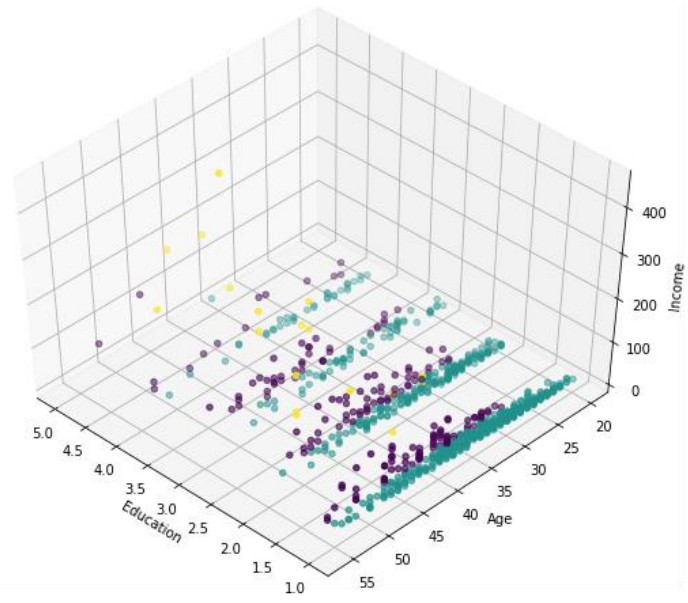
شکل ۳-۱: نتیجهٔ خوشه‌بندی k-means روی مجموعه دادهٔ مشتریان برحسب دو ویژگی Age و Income

برای نمایش نتیجهٔ خوشه‌بندی به‌صورت سه‌بعدی از کلاس Axes3D استفاده می‌کنیم. ابتدا با plt یک figure با سایز دلخواه تعریف می‌کنیم. سپس روی figure نمودار سه‌بعدی ax را در مختصات ارائه

شدهٔ پارامتر `rect` ایجاد می‌کنیم و درنهایت نمونه‌ها را برحسب ستون‌های اول، دوم و چهارم از مجموعه داده رسم می‌کنیم. از تابع `scatter` برای رسم نمونه‌ها استفاده می‌کنیم (شکل ۳-۲).

```
from mpl_toolkits.mplot3d import Axes3D
fig=plt.figure(1,figsize=(8,6))
plt.clf() #clear the figure
ax=Axes3D(fig, rect=[0,0,0.95,1], elev=48,
azim=134)
plt.cla()  #clear the axes
ax.set_xlabel('Education')
ax.set_ylabel('Age')
ax.set_zlabel('Income')
ax.scatter(X[:,1],X[:,0],X[:,3],
c=labels.astype(np.float))
```

همان‌طور که به‌صورت سه‌بعدی در شکل‌های ۳-۱ و ۳-۲ مشخص است مشتریانی که سن بالا، تحصیلات و درآمد زیادی دارند، در یک خوشه قرار می‌گیرند، مشتریان مسن با درآمد متوسط در خوشهٔ دیگر قرار می‌گیرند و مشتریان با سن کم و درآمد کم در خوشهٔ دیگر جای می‌گیرند.



شکل ۳-۲: نتیجهٔ خوشه‌بندی مجموعه دادهٔ مشتریان به‌صورت سه‌بعدی برحسب سه ویژگی Age، income، Education با استفاده از روش k-means

۳-۲- الگوریتم fuzzy c-means

در خوشه‌بندی فازی، هر نمونه دارای یک مجموعه از ضرایب عضویت در خوشه‌ها است. نقاطی که به مرکز یک خوشه نزدیک‌تر هستند درجهٔ عضویت آن‌ها به آن خوشه، نسبت به نقاطی که در لبه‌های خوشه قرار گرفته‌اند، بیشتر است [۲۷]. درجهٔ عضویت یک نمونه به یک خوشه، یک مقدار عددی بین 0 و 1 است. الگوریتم خوشه‌بندی fuzzy c-means بسیار شبیه الگوریتم خوشه‌بندی k-means است. تفاوت اصلی آن با الگوریتم k-means این است که تابع هدف در الگوریتم fuzzy c-means، اجازه می‌دهد که هر نمونه مقادیر عضویت متفاوتی به خوشه‌ها داشته باشد، درحالی‌که در خوشه‌بندی k-means هر نمونه تنها به یک خوشه متعلق است.

تابع هدف الگوریتم fuzzy c-means در عبارت (۴۴) آورده شده است:

$$\min \sum_{j=1}^{k} \sum_{x_i \in C_j} u_{ij}^{m} (x_i - \mu_j)^2 \qquad (۴۴)$$

به‌گونه‌ای که u_{ij} درجهٔ عضویت نمونهٔ x_i به خوشهٔ C_j است. μ_j مرکز خوشهٔ C_j است. m هم پارامتر فازی‌ساز[1] است.

متغیر u_{ij}^{m} با عبارت (۴۵) تعیین می‌شود:

$$u_{ij}^{m} = \frac{1}{\sum_{l=1}^{k} \left(\frac{|x_i - \mu_j|}{|x_i - \mu_l|} \right)^{\frac{2}{m-1}}} \qquad (۴۵)$$

همان‌طور که از عبارت (۴۵) مشخص است، درجهٔ عضویت u_{ij}^{m} با فاصلهٔ نمونه تا مرکز خوشه رابطهٔ عکس دارد. پارامتر فازی‌ساز m یک عدد اعشاری در بازهٔ $1 < m < \infty$ است و سطح فازی‌سازی درجهٔ عضویت نمونه‌ها در خوشه‌ها را تعیین می‌کند. درصورتی‌که برابر 1 باشد، شبیه الگوریتم k-means عمل می‌کند [۲۸]. یک انتخاب مناسب برای پارامتر m عدد 2 است [۲۹].

در خوشه‌بندی فازی، مرکز یک خوشه برابر است با میانگین همهٔ نقاط بر اساس درجهٔ عضویتشان به آن خوشه:

$$\mu_j = \frac{\sum_{x \in C_j} u_{ij}^{m} x}{\sum_{x \in C_j} u_{ij}^{m}} \qquad (۴۶)$$

[1]. fuzzifier

در ادامه الگوریتم خوشه‌بندی فازی آمده است [۲۸]:

- **مرحلهٔ اول:** در ابتدا کاربر باید تعداد خوشه‌ها را تعیین کند (k).
- **مرحلهٔ دوم:** مقادیر درجهٔ عضویت نمونه‌ها به خوشه به‌صورت تصادفی تعیین شود.
- **مرحلهٔ سوم:** مرحلهٔ چهارم تا پنجم را به تعداد دفعات مشخصی که کاربر تعیین می‌کند تکرار کن یا این دو مرحله را تا زمان رسیدن به همگرایی تکرار کن (تا زمانی که تغییرات ضرایب عضویت نمونه‌ها بین دو تکرار بیشتر از ϵ نباشد).
- **مرحلهٔ چهارم:** مراکز خوشه‌ها را با عبارت (۴۶) محاسبه کن.
- **مرحلهٔ پنجم:** برای هر نمونه ضرایب عضویت در خوشه‌ها را با استفاده از عبارت (۴۵) محاسبه کن.

این الگوریتم هم مانند الگوریتم k-means اگرچه الگوریتم توانمندی است اما همانند k-means وابسته به مقداردهی اولیه و تعیین تعداد خوشه‌ها توسط کاربر است. این الگوریتم مانند الگوریتم k-means پیدا کردن مینیمم سراسری را تضمین نمی‌کند.

✓ **مثال:**

در ادامه یک مثال ساده از خوشه‌بندی fuzzy c-means را می‌بینیم [۳۰]:

اگر از jupyter در نرم‌افزار anaconda برای اجرای کدهای پایتون استفاده می‌کنید:

```
!pip install fuzzy-c-means
```

یا در command prompt، ابتدا وارد فولدر scriptهای پایتون نصب شده در سیستمتان شوید:

```
cd
C:\Users\APPLE\AppData\Local\Programs\Python\Python
37\Scripts
```

سپس دستور زیر را در command prompt وارد کنید:

```
pip install fuzzy-c-means
```

بعد از نصب کتابخانهٔ مربوط به fuzzy c-means [۳۱] نوشتن برنامهٔ خوشه‌بندی fuzzy c-means را آغاز می‌کنیم:

ابتدا کتابخانه‌های موردنیاز را فراخوانی می‌کنیم:

```
from fcmeans import FCM
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt
from seaborn import scatterplot as scatter
```

- **ایجاد داده**

سپس مجموعه داده‌ای را برای خوشه‌بندی می‌سازیم. با اکثر پارامترهای تابع make_blobs در فصل قبل در بخش پیاده‌سازی SVM خطی آشنا شدیم:

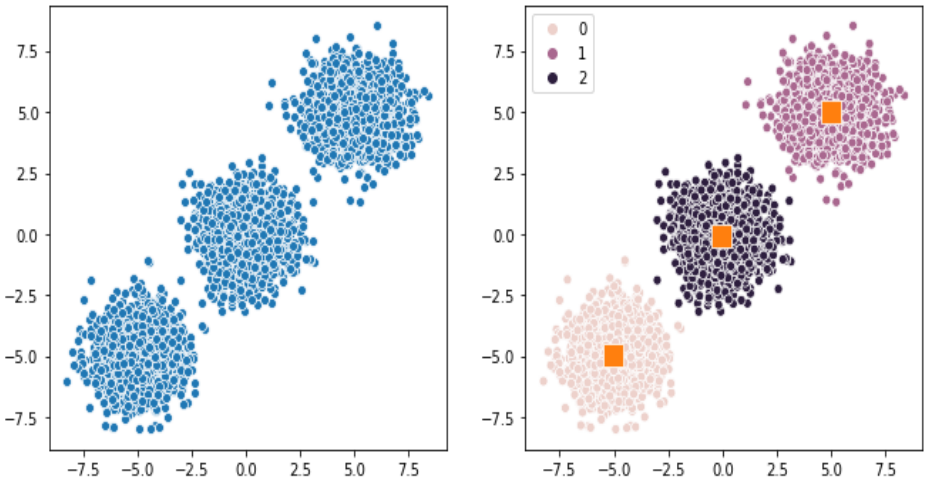
```
#create artifitial dataset
n_samples=5000
n_bins=3 # because we want to have 3 clusters
centers=[(-5,-5),(0,0),(5,5)]
X,_=make_blobs(n_samples=n_samples,n_features=2,
cluster_std=1.0, centers=centers, random_state=42)
```

- **مدل‌سازی**

```
#fit the fuzzy-c-means
fcm=FCM(n_clusters=3)
fcm.fit(X)
#outputs
fcm_centers=fcm.centers
fcm_labels=fcm.u.argmax(axis=1)
```

- **رسم مجموعه داده و نتیجهٔ خوشه‌بندی**

```
#plot result
f,axes=plt.subplots(1,2,figsize=(11,5))
scatter(X[:,0],X[:,1], ax=axes[0])
scatter(X[:,0],X[:,1], ax=axes[1], hue=fcm_labels )
scatter(fcm_centers[:,0],fcm_centers[:,1],
ax=axes[1],marker="s",s=200)
```



شکل ۳-۳: سمت راست: مجموعه دادهٔ تولید شده – سمت چپ: نتیجهٔ خوشه‌بندی `fuzzy c-means` که خوشه‌ها و مراکز خوشه‌ها مشخص است.

۳-۳- الگوریتم سلسله‌مراتبی[۲]

الگوریتم سلسله‌مراتبی یکی از روش‌های متداول و آسان خوشه‌بندی داده است. به دو صورت این روش خوشه‌بندی مطرح شده است:

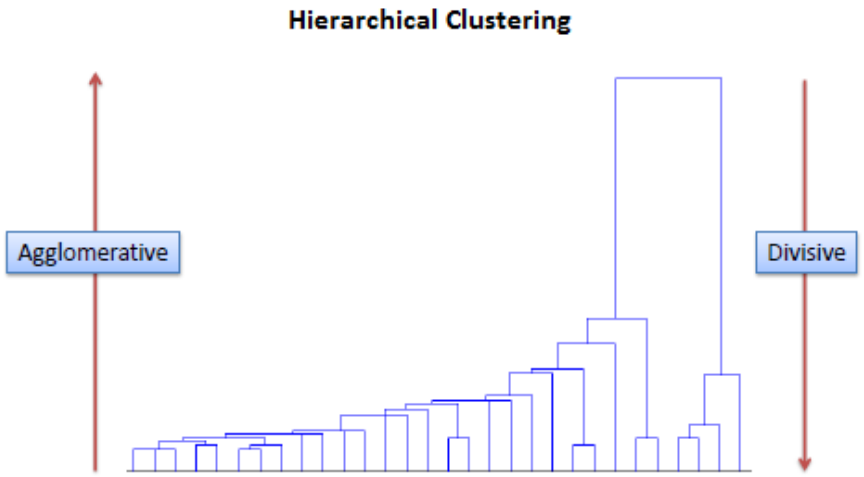
- **تجمعی:**[۳] در این روش، ابتدا هر نقطه در یک خوشه قرار می‌گیرد. سپس در هر تکرار از الگوریتم خوشه‌های نزدیک به هم (مشابه)، مرحله‌به‌مرحله به هم متصل می‌شوند، تا درنهایت یک خوشه داشته باشیم. در این بین می‌توان با دانستن این موضوع که می‌خواهیم داده‌ها به چند خوشه تقسیم‌بندی شوند، قبل از رسیدن به تک خوشه با اعمال یک آستانه الگوریتم را متوقف کنیم. روش خوشه‌بندی سلسله‌مراتبی، رویکردی حریصانه را در پی می‌گیرد و خوشه‌هایی را در هر مرحله با هم ترکیب می‌کند که بیشترین مینیمم‌سازی را در تابع هدف انجام می‌دهند و کاربر می‌تواند در هر مرحله‌ای درخت حاصل از ترکیب خوشه‌ها را قطع کند تا به تعداد خوشه‌ی دلخواه دست یابد. اما مشکل اینجاست که اگر خطایی در مراحل اولیه الگوریتم رخ دهد، این خطا هرچه الگوریتم پیش می‌رود، منتشر می‌شود و قابل اصلاح نیست [۳۲].

- **تقسیم‌کننده:**[۴] ازآنجایی‌که این روش کاربرد چندانی در دسته‌بندی مجموعه داده‌های واقعی ندارد، به توضیح کوتاهی از آن بسنده می‌کنیم؛ به عبارت خیلی ساده می‌توان گفت که این روش برعکس روش تجمعی است. در ابتدا همهٔ نقاط (نمونه‌ها) به‌صورت یک خوشه در نظر گرفته می‌شوند. سپس در هر تکرار

---

2. Hierarchical
3. Agglomerative
4. Divisive

نقاطی را که به نقاط خوشه شبیه نیستند جدا می‌کنیم. هر نقطه‌ای که جدا می‌شود به‌صورت یک خوشه در نظر گرفته می‌شود [۳۳].



شکل۳-۴: خوشه‌بندی سلسله‌مراتبی [۳۴]

مراحل الگوریتم تجمعی سلسله‌مراتبی به شرح زیر است:

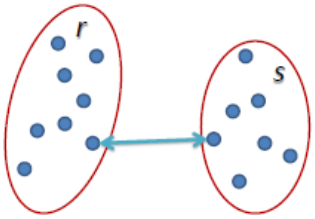
- **مرحلهٔ اول:** ماتریس فاصله را بین تمامی نقاط (نمونه‌های) مجموعه داده محاسبه کن.
- **مرحلهٔ دوم:** هر نقطه را به‌عنوان یک خوشه در نظر بگیر.
- **مرحلهٔ سوم:** دو خوشهٔ نزدیک را با هم ادغام کن.
- **مرحلهٔ چهارم:** ماتریس فاصله را به‌روز کن.
- **مرحلهٔ پنجم:** از مرحلهٔ سوم تکرار را آغاز کن تا زمانی که یک خوشه (یا k خوشه) باقی بماند.

نتیجهٔ خوشه‌بندی با این روش را به‌صورت دندروگرام (شبیه درخت) در شکل ۳-۴ مشاهده می‌کنید.

محاسبهٔ ماتریس فاصله به سه روش انجام می‌شود:

- **Single-linkage** (نام‌های دیگر این روش، `connectedness` یا `minimum` است).

در این روش فاصلهٔ بین دو خوشه با کوتاه‌ترین فاصلهٔ بین دو نقطه از این دو خوشه برابر است (شکل ۳-۵).

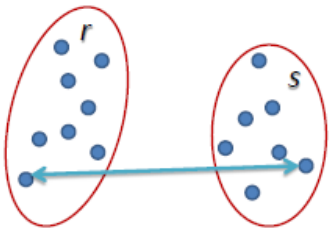


$$L(r,s) = \min(D(x_{ri}, x_{sj}))$$

شکل ۳-۵: محاسبهٔ فاصلهٔ بین دو خوشه با روش `Single-linkage` [۳۴]

- **`Complete-linkage`** (نام‌های دیگر این روش، `diameter` یا `maximum` است):

در این روش فاصلهٔ دو خوشه با بیشترین فاصلهٔ بین دو نقطهٔ این دو خوشه برابر است (شکل ۳-۶).

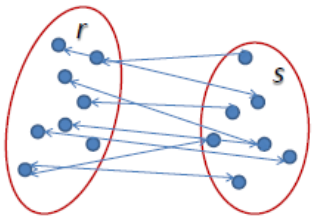


$$L(r,s) = \max(D(x_{ri}, x_{sj}))$$

شکل ۳-۶: محاسبهٔ فاصلهٔ بین دو خوشه با روش `complete-linkage` [۳۴]

- **`Average-linkage`:**

در این روش فاصلهٔ بین دو خوشه با میانگین فواصل هر نقطه از یک خوشه تا همهٔ نقاط خوشه دیگر برابر است (شکل ۳-۷).



$$L(r,s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} D(x_{ri}, x_{sj})$$

شکل ۳-۷: محاسبهٔ فاصلهٔ بین دو خوشه با روش `average-linkage` [۳۴]

✓ **مثال:**

حال اجرای الگوریتم تجمعی سلسله‌مراتبی را با مثالی [۳۰] ادامه می‌دهیم:

برای خوشه‌بندی سلسله‌مراتبی مجموعه داده متداول `iris` را که پیشتر با آن آشنا شدیم، در نظر می‌گیریم.

```
from sklearn import datasets
iris=datasets.load_iris()
feat=iris.feature_names
X=iris.data
Y=iris.target
Y_names= ['Setosa', 'Versicolour', 'Virginica']
```

- **پیش‌پردازش**

```
from sklearn import preprocessing
X=preprocessing.MinMaxScaler().fit_transform(X)
```

تابع `MinMaxScaler()` مقدارهای هر ویژگی را به بازهٔ `(min,max)` منتقل می‌کند:

$$std = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (۲۰)$$

$$X_{scaled} = std * (max - min) + min \quad (۲۱)$$

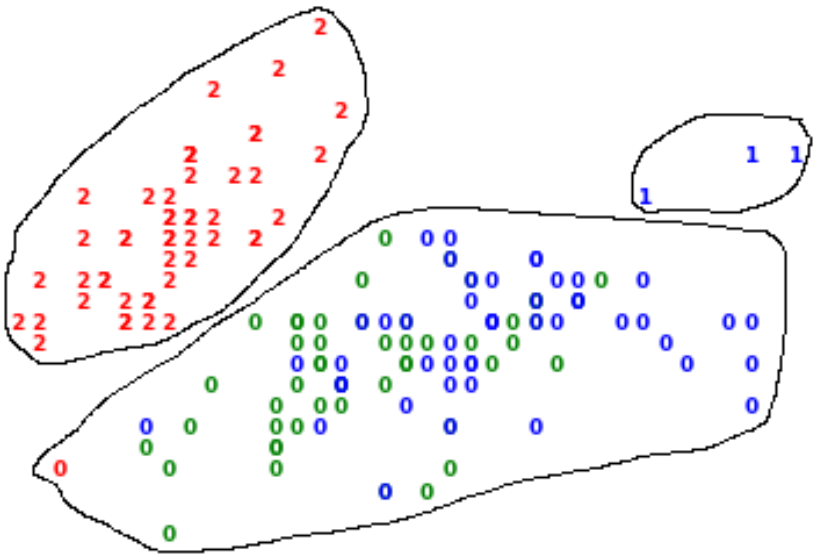
- **مدل‌سازی**

از ماژول `sklearn.cluster` کلاس `AgglomerativeClustering` را فراخوانی می‌کنیم. یک شیء به نام `clustering` از این کلاس می‌سازیم. مقدار پارامتر `linkage` و تعداد خوشه‌ها را مشخص می‌کنیم. پارامتر `linkage` مشخص‌کنندهٔ روش محاسبهٔ ماتریس فاصله است. در ادامه با تابع `fit` از شیء `clustering`، الگوریتم را به نمونه‌ها اعمال می‌کنیم. سپس با `plt.text` نتیجهٔ خوشه‌بندی را در دو بعد نمایش می‌دهیم. `Plt.text` این امکان را به ما می‌دهد که مقدار رشته‌ای را در مختصات موردنظر چاپ کنیم. ما در اینجا برچسب نمونه‌ها را چاپ کردیم.

```
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
```

```
clustering=AgglomerativeClustering(linkage="average",n_
clusters=3)
clustering.fit(X)


colours='rgb'
for i in range(X.shape[0]):
     plt.text(X[i,0],X[i,1],
     str(clustering.labels_[i]),
     color=colours[Y[i]],
     fontdict={'weight':'bold','size':9})


plt.xticks([])
plt.yticks([])
plt.axis('off')
plt.show()
```

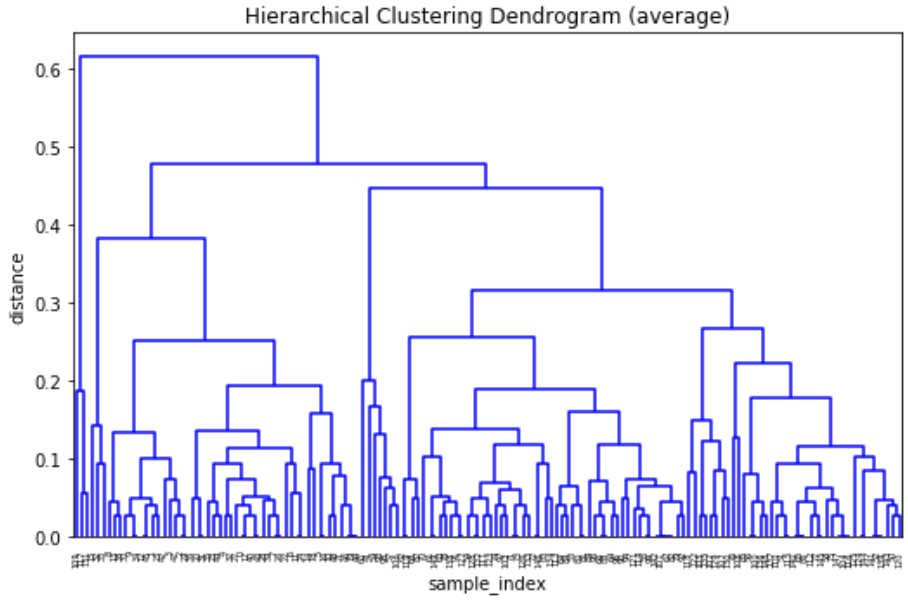
نتیجهٔ خوشه‌بندی با الگوریتم خوشه‌بندی سلسله‌مراتبی را در شکل ۳-۸ مشاهده می‌کنید. در این شکل نمونه‌های مربوط به هر خوشه در یک منحنی نشان داده شده است (منحنی‌ها به‌علت خاکستری بودن نمونه‌ها در تصاویر کتاب، برای درک بهتر خوانندگان به‌صورت دستی رسم شده است).



شکل ۳-۸: الگوریتم خوشه‌بندی تجمعی روی مجموعه دادهٔ iris

حال به رسم دندروگرام می‌پردازیم که نمایش‌دهندهٔ مراحل خوشه‌بندی سلسله‌مراتبی است:

```
from scipy.cluster.hierarchy import dendrogram,linkage
linkage_matrix=linkage(X,'average')
figure=plt.figure(figsize=(7.5,5))
dendrogram(linkage_matrix, color_threshold=0,)
plt.title('Hierarchical Clustering Dendrogram (average)')
plt.xlabel('sample_index')
plt.ylabel('distance')
# Automatically adjust subplot parameters to give #specified padding.
plt.tight_layout()
plt.show()
```



شکل ۳-۹: دندروگرام حاصل از خوشه‌بندی تجمعی سلسله‌مراتبی روی مجموعه دادهٔ iri

فصل چهارم

پیش‌پردازش داده[1]

PYTHON

[1]. Data preprocessing

کیفیت اطلاعات مجموعه داده‌هایی که برای پردازش استفاده می‌شوند، نقش بسیار مهمی در یادگیری یک سیستم یادگیری ماشین دارد. بنابراین پیش‌پردازش مجموعه داده و آماده‌سازی آن برای پردازش، قبل از اعمال الگوریتم‌های یادگیری ماشین اهمیت بسیار دارد. در این فصل با روش‌های اصلی آماده‌سازی و پیش‌پردازش اطلاعات آشنا می‌شویم. برنامه‌های این بخش برگرفته از فصل چهارم کتاب [۵] است.

آنچه در این فصل مطرح می‌شود:

- مدیریت «اطلاعات گم شده» در مجموعه داده
- مدیریت داده‌های دسته‌ای
- تقسیم یک مجموعه داده به بخش‌های آموزش و آزمون
- یکسان کردن مقیاس ویژگی‌ها

۴-۱- مدیریت «اطلاعات گم شده»[۲] در مجموعه داده

گاهی در مجموعه داده‌ها با فیلدهای خالی یا فیلدهای حاوی NaN (Not a Number) مواجه می‌شویم. علت آن می‌تواند نداشتن ابزار اندازه‌گیری مناسب یا هر نوع خطایی در جمع‌آوری اطلاعات باشد. ممکن است کاربر فراموش کرده باشد فیلد (فیلدهای) موردنظر را پر کند یا اینکه هنوز برخی از اندازه‌گیری‌ها انجام نشده باشد [۵].

قبل از اینکه به نحوهٔ مدیریت اطلاعات گم شده بپردازیم، اجازه بدهید مجموعه دادهٔ کوچکی را به همراه اطلاعات گم شده بسازیم. داده‌های جداسازی شده با کاما[۳] (فرمت CSV) را با تابع read_csv از کتابخانهٔ pandas به‌صورت یک دیتافریم می‌خوانیم و در df ذخیره می‌کنیم:

```
import pandas as pd
from io import StringIO
csv_data='''A,B,C,D
1.0,2.0,3.0,4.0
5.0,6.0,,8.0
10.0,11.0,12.0,'''
df=pd.read_csv(StringIO(csv_data))
```

[2]. Missing Data
[3]. Comma-Separated Values

حال می‌توان تعیین کرد که هر کدام از فیلدهای مجموعه داده دارای مقدار هستند یا نه (تابع `inull()`)، همچنین می‌توان تعداد این فیلدها را با تابع `sum()` محاسبه کرد:

```
df.isnull()
```

	A	B	C	D
0	False	False	False	False
1	False	False	True	False
2	False	False	False	True

حذف نمونه‌ها و ویژگی‌هایی که دارای مقادیر گم شده هستند:

برای حذف سطرهایی که دارای مقدار `Nan` هستند از تابع `dropna()` استفاده می‌کنیم:

```
df.dropna()
```

برای حذف ستون‌هایی که دارای مقدار `Nan` هستند از تابع `dropna()` استفاده می‌کنیم، درحالی‌که مقدار پارامتر `axis` را برابر `1` قرار می‌دهیم:

```
df.dropna(axis=1)
```

تابع `dropna` پارامترهای مختلفی دارد که با تنظیم آن‌ها، سطرها و ستون‌ها با در نظر گرفتن برخی شرایط حذف می‌شوند:

```
# only drop rows where all columns are NaN
df.dropna(how='all')
# drop rows that have not at least 4 non-NaN values
df.dropna(thresh=4)
# only drop rows where NaN appear in specific
# columns (here: 'C')
df.dropna(subset=['C'])
```

البته همیشه حذف کردن ستون‌ها و سطرها راه‌حل ما نیست. گاهی ممکن است یک ویژگی مهم در دسته‌بندی را حذف کنیم یا گاهی ممکن است تعداد نمونه‌های زیادی حذف شوند. در بخش بعدی با روش‌های درون‌یابی[۴] سعی می‌کنیم به این مشکل غلبه کنیم.

تخمین مقادیر گم شده:

در این قسمت با استفاده از نمونه‌های مجموعه داده، از روش‌های مختلف درون‌یابی برای تخمین مقادیر گم شده استفاده می‌کنیم. متداول‌ترین روش درون‌یابی، جایگزینی مقدار گم شده با میانگین کل مقادیر آن ویژگی است. برای این کار از کلاس `Imputer` از کتابخانهٔ `sklearn.preprocessing` استفاده می‌کنیم:

```
from sklearn.preprocessing import Imputer
imr= Imputer(missing_values='NaN',
strategy='mean',axis=0)
imr=imr.fit(df)
imputed_data=imr.transform(df.values)
imputed_data
```

در خروجی می‌بینیم:

```
array([[ 1. ,  2. ,  3. ,  4. ],
       [ 5. ,  6. ,  7.5,  8. ],
       [10. , 11. , 12. ,  6. ]])
```

کلاس `Imputer` متعلق به کلاس‌های `transformer` در ماژول `sklearn` است. دو متد اساسی این کلاس‌ها، `fit()` و `transform()` هستند. متد `fit` برای یاد گرفتن پارامترها با استفاده از مجموعه داده استفاده می‌شود. سپس متد `transform()` از این پارامترها برای انتقال داده‌ها استفاده می‌کند. دسته‌بندهای مختلفی که در فصل دوم از کتابخانهٔ `sklearn` استفاده کردیم، نیز دارای دو متد `transform()` و `fit()` هستند. به‌طوری‌که متد `fit()`، پارامترهای مدل ساخته شده از دسته‌بند را یاد می‌گیرد [۵].

---

[4]. Interpolation

۴-۲- مدیریت داده‌های دسته‌ای[5]

در بسیاری از مجموعه داده‌هایی که با آن‌ها مواجه می‌شویم، برخی از ویژگی‌های آن‌ها به‌صورت غیرعددی و دسته‌ای هستند. برخی از آن‌ها ترتیبی[6] هستند مانند سایز لباس (XL>L>M) و برخی دیگر اسمی[7] هستند مانند رنگ‌های لباس. قبل از اینکه به روش‌های مختلف مدیریت این داده‌ها بپردازیم ابتدا اجازه دهید یک دیتا فریم جدید بسازیم:

```
import pandas as pd
pd.DataFrame([['green','M',10.1,'class1'],
              ['red','L',13.5,'class2'],
              ['blue','XL',15.3,'class1']])
df.columns=['color','size','price','classlabel']
df
```

در خروجی می‌بینیم:

	color	size	price	classlabel
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	10.0	11.0	12.0	NaN

نگاشت ویژگی‌های ترتیبی:

نگاشت‌های ترتیبی را باید به‌صورت دستی تعریف کنیم، زیرا روند ترتیبی آن را می‌دانیم به‌عنوان‌مثال:

```
XL=L+1=M+2
size_mapping = {'XL':3,'L':2,'M':1}
df['size']=df['size'].map(size_mapping)
df
```

5. Categorical data
6. Ordinal
7. Nominal

	color	size	price	classlabel
0	green	1	10.1	class1
1	red	2	13.5	class2
2	blue	3	15.3	class1

درصورتی‌که بخواهیم مقادیر عددی به دست آمده را بعداً به همان مقادیر رشته‌ای تبدیل کنیم، یک دیکشنری reverse_mapping ایجاد می‌کنیم:

```
Inv_size_mapping = {v : k for k, v in
size_mapping.items()}
df['size'] = df['size'].map(inv_size_mapping)
```

تبدیل برچسب‌های دسته‌ها:

بسیاری از کتابخانه‌های الگوریتم‌های یادگیری ماشین برای برچسب‌های دسته‌ها، نیازمند مقادیر عددی هستند. اگرچه بسیاری از آن‌ها به‌صورت داخلی این تبدیل را انجام می‌دهند اما بهتر است برای جلوگیری از برخی مشکلات احتمالی این تبدیل در مرحلهٔ پیش‌پردازش انجام شود:

```
import numpy as np
class_mapping = {label:idx for idx,label in
enumerate(np.unique(df['classlabel']))}
df['classlabel']=df['classlabel'].map(class_mapping)
df
```

	color	size	price	classlabel
0	green	M	10.1	0
1	red	L	13.5	1
2	blue	XL	15.3	0

اگرچه در کتابخانهٔ sklearn، کلاس LabelEncoder کار را راحت‌تر کرده است (متد fit_transform راه‌حل مختصری است برای استفاده از متدهای fit و transform به‌صورت جداگانه):

```
from sklearn.preprocessing import LabelEncoder
class_le = LabelEncoder()
df['classlabel']=class_le.fit_transform(df['classlabel'].values)
df[‘classlabel’]
```

در خروجی داریم:

```
0    0
1    1
2    0
Name: classlabel, dtype: int32
```

برای برگرداندن مقادیر قبلی برچسب‌های کلاس از متد inverse_transform استفاده می‌کنیم:

```
df['classlabel']=class_le.inverse_transform(df['classlabel'].values)
df['classlabel']
```

در خروجی داریم:

```
0    class1
1    class2
2    class1
Name: classlabel, dtype: object
```

تبدیل one-hot برای ویژگی‌های اسمی:

قبل از اینکه به تبدیل one-hot بپردازیم، به نظر می‌رسد که می‌توانیم از کلاس labelEncoder برای کدگذاری ویژگی color استفاده کنیم:

```
X=df[['color','size','price']].values
color_le=LabelEncoder()
```

```
X[:,0]=color_le.fit_transform(X[:,0])
X
```

در خروجی داریم:

```
array([[1, 'M', 10.1],
       [2, 'L', 13.5],
       [0, 'XL', 15.3]], dtype=object)
```

اگرچه مقادیر رنگ، ترتیبی ندارند اما الگوریتم یادگیری ماشین فرض می‌کند رنگی که متناظر است با مقدار بیشتر، بزرگ‌تر از رنگ‌های دیگر است. با وجودی که این فرض اشتباه است، الگوریتم کار می‌کند اما آنچه به دست می‌آید بهینه نیست.

برای حل این مشکل، روشی به نام `one-hot` وجود دارد:

ایدهٔ اصلی این روش این است که به‌ازای مقادیر یکتا در ویژگی موردنظر، یک ویژگی باینری ایجاد کنیم. به‌عنوان‌مثال، نمونهٔ آبی به‌صورت `blue=1, green=0, red=0` کدگذاری می‌شود. برای ایجاد این تبدیل از کلاس `OneHotEncoder` استفاده می‌کنیم:

```
from sklearn.preprocessing import OneHotEncoder
ohe=OneHotEncoder(categorical_features=([0]))
X_new=ohe.fit_transform(X).toarray()
print(X_new)
```

در خروجی داریم:

```
[[ 0.   1.   0.   1.  10.1]
 [ 0.   0.   1.   2.  13.5]
 [ 1.   0.   0.   3.  15.3]]
```

همان‌طور که از خروجی مشخص است، به‌ازای سه رنگ، سه ستون (سه ویژگی) پدیدار شد که درصورتی‌که مقدار رنگ موردنظر `True` باشد، مقدار متناظر آن 1 است.

زمانی که کلاس `OneHotEncoder` را مقداردهی می‌کنیم، موقعیت ستونی (ویژگی) را که می‌خواهیم به `one-hot` شود، در پارامتر `categorical_features` مشخص می‌کنیم: به‌صورت پیش‌فرض، کلاس `OneHotEncoder` بعد از استفاده از متد `transform` یک ماتریس `sparse` را برمی‌گرداند که ما برای سادگی در نمایش از متد `()toarray` استفاده می‌کنیم. البته برای

اینکه از متد toarray() استفاده نکنیم، می‌توانیم پارامتر sparse کلاس OneHotEncoder را False قرار دهیم [۵].

یک راه آسان‌تر برای ایجاد این ستون‌های باینری برای انواع متفاوت ویژگی موردنظر، استفاده از متد get_dummies برای تبدیل ویژگی‌های رشته‌ای است:

```
pd.get_dummies(df[['price','color','size']])
```

در خروجی داریم:

	price	size	color_blue	color_green	color_red
0	10.1	1	0	1	0
1	13.5	2	0	0	1
2	15.3	3	1	0	0

۴-۳- تقسیم یک مجموعه داده به بخش‌های آموزش و آزمون

با استفاده از مجموعه داده‌های کتابخانهٔ sklearn، مجموعه دادهٔ breast cancer را لود می‌کنیم:

```
from sklearn.datasets import load_breast_cancer
cancer=load_breast_cancer()
cancer_df=pd.DataFrame(data=cancer.data[0:,0:],
            columns=cancer.feature_names)
print('calss labels', np.unique(cancer.target))
```

در خروجی داریم:

```
calss labels [0 1]
```

سپس با انجام دستور cancer_df.head() چهار نمونهٔ اول مجموعه دادهٔ سرطان را در جدول ۴-۱ می‌بینیم:

```
cancer_df.head()
```

جدول ۴-۱: چهارنمونهٔ اول مجموعه دادهٔ سرطان سینه

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	wors
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.162	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.123	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.144	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.209	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.137	

مقادیر ویژگی‌های این مجموعه داده از تصاویر توده‌های سینه محاسبه شده‌اند. این مجموعه داده 596 نمونه و 30 ویژگی دارد که از بین این نمونه‌ها، ۲۱۲ نمونه بدخیم و بقیه خوش‌خیم هستند. پس متغیر هدف دارای دو مقدار 0 و 1 است.

آسان‌ترین راه برای تقسیم این مجموعه داده به مجموعه داده‌های مجزای آموزش و آزمون، استفاده از تابع train_test_split از کتابخانهٔ sklearn است. ستون‌های 1 تا 30 را به‌عنوان X و برچسب‌های کلاس‌ها را به‌عنوان y در نظر می‌گیریم. سپس X و y را به‌عنوان پارامترهای تابع train_test_split در نظر می‌گیریم. با قرار دادن test_size=0.3، 30درصد از مجموعه داده را برای آزمون و 70درصد از آن را برای آموزش در نظر می‌گیریم:

```
from sklearn.model_selection import train_test_split
X, y = cancer_df.iloc[:, 1:].values, cancer.target
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3,
random_state=0)
```

البته باید توجه داشته باشیم اندازه‌ای که برای test_size در نظر می‌گیریم باید به‌گونه‌ای باشد که بسته به‌اندازهٔ مجموعه داده، نه خیلی زیاد باشد که سیستم یادگیری ماشین به‌خوبی آموزش نبیند و نه خیلی کم باشد که خطای تعمیم الگوریتم را نتوانیم به‌خوبی اندازه‌گیری کنیم. معمولاً 40:60، 30:70 یا 20:80 تقسیم‌هایی است که بر اساس اندازهٔ مجموعه داده متداول است. برای مجموعه داده‌های خیلی بزرگ 10:90 هم می‌تواند تقسیم‌بندی مناسبی باشد.

۴-۴- یکسان کردن مقیاس ویژگی‌ها

درختان تصمیم و جنگل‌های تصادفی[8] جزو معدود الگوریتم‌های یادگیری ماشین هستند که نیازی به یکسان کردن مقیاس ویژگی ندارند، درحالی‌که اکثر الگوریتم‌های یادگیری ماشین با یکسان کردن مقیاس الگوریتم‌ها، بسیار بهتر عمل می‌کنند [۵]. فرض کنید مقیاس یکی از ویژگی‌ها در بازهٔ 0 تا 10 باشد درحالی‌که مقیاس ویژگی دیگر در بازهٔ 1 تا 100000 باشد. به‌عنوان‌مثال در الگوریتم KNN، هنگام محاسبهٔ فاصلهٔ اقلیدسی، الگوریتم نمی‌تواند درک درستی از فواصل نمونه‌ها داشته باشد.

دو روش کلی برای مقیاس کردن ویژگی‌ها وجود دارد: استانداردسازی[9] و نرمال‌سازی.[10]

8. Random forests
9. Standardization
10. Normalization

نرمال‌سازی حالت خاصی از مقیاس کردن min-max است، به‌طوری‌که مقادیر ویژگی را به بازهٔ [0,1] منتقل می‌کند:

$$x_{norm}^{i} = \frac{x^{i} - x_{min}}{x_{max} - x_{min}} \quad (۱)$$

درحالی‌که x^{i} نمونهٔ iام مجموعه داده و x_{min} کمترین مقدار ویژگی موردنظر است. x_{max} هم بیشترین مقدار ویژگی موردنظر است. فرایند نرمال‌سازی در کتابخانهٔ sklearn پیاده‌سازی شده است و به این شرح قابل‌استفاده است:

```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)
X_test_norm_df=pd.DataFrame(data=X_test_norm[0:,0:])
X_test_norm_df.head()
```

در خروجی بخشی از داده‌های نرمال شده را به‌صورت جدول ۴-۲ می‌بینیم:

جدول ۴-۲: بخشی از داده‌های نرمال شدهٔ مجموعه دادهٔ سرطان سینه

	0	1	2	3	4	5	6
0	0.448548	0.309930	0.175270	0.629630	0.436682	0.338566	0.406163
1	0.644813	0.278557	0.167296	0.383187	0.111876	0.064948	0.102783
2	0.246888	0.316495	0.196394	0.293581	0.124752	0.048899	0.131809
3	0.351037	0.327759	0.207678	0.142609	0.112390	0.057990	0.068290
4	0.361411	0.268261	0.161315	0.404040	0.062598	0.060028	0.145278

اگرچه نرمال‌سازی با روش مقیاس min-max روش متداول و مفیدی است و مقادیر را محدود به یک بازهٔ بسته می‌کند اما استانداردسازی برای بسیاری از الگوریتم‌های یادگیری ماشین بسیار پرکاربردتر است. دلیل این موضوع این است که بسیاری از مدل‌های خطی مانند خطی‌سازی لجستیک و SVM، مقادیر وزن‌ها را با 0 یا مقادیر تصادفی کوچکی مقداردهی اولیه می‌کنند. بنابراین با استفاده از استانداردسازی می‌توانیم مقدار ویژگی‌ها را با میانگین 0 و انحراف معیار 1 مقیاس کنیم. بنابراین مقادیر ویژگی‌ها شکل یک توزیع نرمال را به خود می‌گیرند که باعث می‌شود الگوریتم، وزن‌ها را به‌خوبی یاد بگیرد. علاوه بر این استانداردسازی، اطلاعات مفیدی را دربارهٔ نویزها حفظ می‌کند و باعث می‌شود الگوریتم به نویزها کمتر حساس باشد [۵]. استانداردسازی را می‌توان با عبارت (۴۷) بیان کرد:

$$x_{std}^{i} = \frac{x^{i} - \mu_{x}}{\sigma_{x}} \quad (۴۷)$$

به‌گونه‌ای که x^i نمونهٔ i ام مجموعه داده و μ_x میانگین مقادیر ویژگی موردنظر است. σ_x نیز انحراف از معیار ویژگی موردنظر است. فرایند استانداردسازی در کتابخانهٔ `sklearn` پیاده‌سازی شده است و به این شرح قابل‌استفاده است:

```
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)
X_test_std_df=pd.DataFrame(data=X_test_std[0:,0:])
X_test_std_df.head()
```

در خروجی بخشی از داده‌های استاندارد شده را به‌صورت جدول ۴-۳ می‌بینیم:

جدول ۴-۳: بخشی از داده‌های استاندارد شدهٔ مجموعه دادهٔ سرطان سینه

	0	1	2	3	4	5	6
0	0.317107	-0.149384	-0.287243	1.016289	0.840148	0.713667	0.823176
1	1.447278	-0.332906	-0.338596	-0.616243	-1.020853	-0.782447	-0.735274
2	-0.844125	-0.110982	-0.151214	-1.209826	-0.947080	-0.870205	-0.586169
3	-0.244394	-0.045092	-0.078556	-2.209923	-1.017909	-0.820497	-0.912463
4	-0.184660	-0.393137	-0.377110	-0.478100	-1.303191	-0.809351	-0.516978

فصل پنجم

کاهش ابعاد داده[1]

PYTHON

[1]. Data preprocessing

در این فصل روش‌های مهم و کاربردی در دو حوزهٔ انتخاب ویژگی و استخراج ویژگی برای کاهش ابعاد داده مطرح می‌شود. در روش‌های انتخاب ویژگی، همان‌طور که از نام آن مشخص است، تعدادی ویژگی از بین ویژگی‌ها انتخاب می‌شود. این در حالی است که ماهیت ویژگی‌ها تمام و کمال حفظ می‌شود، درحالی‌که در روش‌های استخراج ویژگی به‌دنبال تبدیلی هستیم که فضای ویژگی‌ها را به فضایی جدید و با بعد کمتر ببرد. بدین ترتیب، این تبدیل‌ها به‌نوعی باعث فشرده‌سازی داده‌ها می‌شوند و عمده تبدیل‌های به کار رفته در این حوزه خطی هستند.

۵-۱- مشکل بیش برازش[۲]

از نشانه‌های بیش برازش این است که یک مدل روی یک مجموعه داده آموزش بسیار بهتر از مجموعه داده آزمون نتیجه بدهد. بیش برازش به این معنی است که مدل، پارامترها را بر اساس نمونه‌های مشخصی در مجموعه داده آموزش یاد بگیرد و نتواند تعمیم خوبی برای داده‌های واقعی داشته باشد. در این صورت می‌گوییم مدل واریانس زیادی دارد. راه‌حل‌های متداول برای غلبه بر این مشکل به این شرح است [۵]:

- جمع‌آوری داده‌های بیشتر برای آموزش
- انتخاب یک مدل ساده‌تر با پارامترهای کمتر
- کاهش ابعاد داده

جمع‌آوری داده‌های بیشتر خیلی از مواقع امکان‌پذیر نیست. در بخش‌های بعدی تلاش می‌کنیم با معرفی چند روش از روش‌های انتخاب ویژگی و استخراج ویژگی، کاهش ابعاد داده را بررسی کنیم.

۵-۲- روش‌های انتخاب ویژگی[۳]

برای مجموعه داده‌ای که N ویژگی دارد، 2^N زیرمجموعه برای انتخاب وجود دارد. با استفاده از روش‌های انتخاب ویژگی، زیرمجموعه‌ای از ویژگی‌ها انتخاب می‌شوند که کمترین تعداد ویژگی را داشته باشند، درحالی‌که بالاترین دقت را در الگوریتم‌های یادگیری ماشین ارائه دهند. با افزایش تعداد ویژگی‌های مجموعه داده، تعداد زیرمجموعه‌ها افزایش پیدا می‌کند و محاسبهٔ بهترین زیرمجموعه، پیچیدگی محاسباتی زیادی دارد. روش‌های مختلفی در این زمینه ارائه شده است که اگرچه موفقیت‌هایی را کسب کرده‌اند اما اغلب آن‌ها دچار مشکلاتی چون زمان همگرایی زیاد، ناپایداری و گیر کردن در بهینهٔ محلی هستند [۳۵]. به‌طورکلی الگوریتم‌های انتخاب ویژگی به چهار دسته تعلق دارند [۳۵]: روش‌های فیلتر،[۴] روش‌های پوششی،[۵] روش‌های ترکیبی[۶] و روش‌های تعبیه شده.[۷]

---

2. Overfitting problem
3. Feature selection
4. Filter methods
5. Wrapper methods
6. Hybrid methods
7. Embeded methods

روش‌های فیلتری:

در این روش‌ها برای انتخاب زیرمجموعه ویژگی مناسب، از مشخصات ذاتی و آماری ویژگی‌ها استفاده می‌کنند و مستقل از هرگونه الگوریتم یادگیری هستند. در این الگوریتم‌ها بر اساس میزان ارتباط و وابستگی[۸] ویژگی‌ها با برچسب کلاس، به هر ویژگی، وزنی اختصاص می‌یابد؛ معمولاً از معیارهای همبستگی[۹] و معیارهای مبتنی بر تئوری اطلاعات،[۱۰] برای وزن‌دهی ویژگی‌ها استفاده می‌شود. روش‌های فیلتری به‌دلیل نیاز به محاسبات کمتر، برای مجموعه داده‌های با ابعاد بالا[۱۱] مؤثر هستند ولی دقت مناسبی را ندارند [۳۵].

روش‌های پوششی:

از این روش‌ها برای یافتن زیرمجموعه ویژگی‌ها، از یک الگوریتم‌های یادگیری و یک دسته‌بند[۱۲] استفاده می‌کنند. در این روش‌ها مدل جست‌وجو، وظیفه جست‌وجو در فضای ویژگی‌های اولیه و انتخاب زیرمجموعه ویژگی‌های کاندیدا را دارد؛ همچنین از دسته‌بند برای تخمین کارایی زیرمجموعه کاندیدای انتخاب شده استفاده می‌شود. در مقایسه با الگوریتم‌های فیلتری، الگوریتم‌های پوششی دارای هزینه‌های محاسباتی بالاتری بوده و برای مجموعه داده‌های با ابعاد بالا مناسب نیستند؛ ولی در یافتن زیرمجموعه ویژگی‌های مؤثر موفق‌تر عمل می‌کنند و دقت زیرمجموعه ویژگی انتخاب شده با این روش بالاست.

در بسیاری از الگوریتم‌های پوششی از روش‌های جست‌وجوی تکاملی[۱۳](مرحله‌ای) برای یافتن زیرمجموعه ویژگی‌ها استفاده شده است. این الگوریتم‌ها از یک راه‌حل اولیه، که به‌صورت تصادفی تولید شده، شروع کرده و در هر تکرار یک قدم به بهترین زیرمجموعه جواب نزدیک می‌شوند. الگوریتم‌های تکاملی استفاده شده در روش‌های پوششی شامل الگوریتم ژنتیک، الگوریتم شبیه‌سازی ذوب فلزات، الگوریتم بهینه‌سازی مورچگان، الگوریتم جهش قورباغه، الگوریتم پرندگان و... است.

روش‌های ترکیبی:

در این روش‌ها، روش‌های فیلتری و پوششی با هم ترکیب می‌شوند؛ به‌طوری‌که در مرحلهٔ اول بر اساس یک روش فیلتر تعدادی ویژگی‌ها بر اساس اهمیت انتخاب می‌شوند. سپس در فضای ویژگی‌های انتخاب شده، یک روش پوششی، برای انتخاب ویژگی‌های مؤثر اعمال می‌شود.

---

8. Relevency
9. Dependence Measures
10. Information Theory
11. High Dimantional Dataset
12. Classifier
13. Evolutionary search

روش‌های تعبیه شده:

انتخاب زیرمجموعه ویژگی به‌عنوان بخشی از ساخت مدل در نظر گرفته می‌شود. این روش‌ها را می‌توان جست‌وجو در فضای ویژگی و مدل در نظر گرفت. از جمله این روش‌ها می‌توان به Adaboost، جنگل تصادفی و درخت تصمیم اشاره کرد.

در این بخش دو روش انتخاب ویژگی را به همراه پیاده‌سازی آن‌ها در زبان پایتون، معرفی می‌کنیم. روش‌های انتخاب ویژگی زیادی در کتابخانهٔ sklearn پیاده‌سازی شده‌اند. برای اطلاعات بیشتر در این زمینه به [۳۶] مراجعه بفرمایید.

۵-۲-۱- انتخاب ویژگی با جنگل‌های تصادفی[۱۴]

جنگل‌های تصادفی یکی از روش‌های انتخاب ویژگی‌های مرتبط است. این روش، یک روش دسته‌ای[۱۵] است. با استفاده از یک جنگل تصادفی اهمیت ویژگی را با اندازه‌گیری کاهش ناخالصی میانگین درخت‌های تصمیم جنگل محاسبه می‌کنیم (محاسبهٔ ناخالصی در فصل دوم قسمت درخت تصمیم شرح داده شده است). این محاسبه بدون در نظر گرفتن فرض تفکیک‌پذیر بودن داده‌ها به‌صورت خطی انجام می‌شود. پیاده‌سازی جنگل تصادفی در کتابخانهٔ sklearn، اهمیت ویژگی‌ها را برای ما جمع‌آوری می‌کند. در ادامه ما می‌توانیم با استفاده از feature_importance_ بعد از اعمال RandomForestClassifier به اهمیت ویژگی‌ها دسترسی داشته باشیم [۵].

✓ **مثال**

با اجرای برنامه‌ای [۵] که در ادامه مطرح می‌شود، یک جنگل با 10000 درخت روی مجموعه دادهٔ سرطان سینه آموزش داده می‌شود و 30 ویژگی بر اساس اهمیتشان مرتب می‌شوند (یادآوری: روش‌های مبتنی بر درخت نیازی به نرمال‌سازی و استانداردسازی ندارند).

```
from sklearn.ensemble import RandomForestClassifier
feat_labels = cancer.feature_names
forest = RandomForestClassifier(n_estimators=10000,
                                random_state=0,
                                n_jobs=-1)
forest.fit(X_train, y_train)
importances = forest.feature_importances_
```

14. Random Forests
15. Ensemble

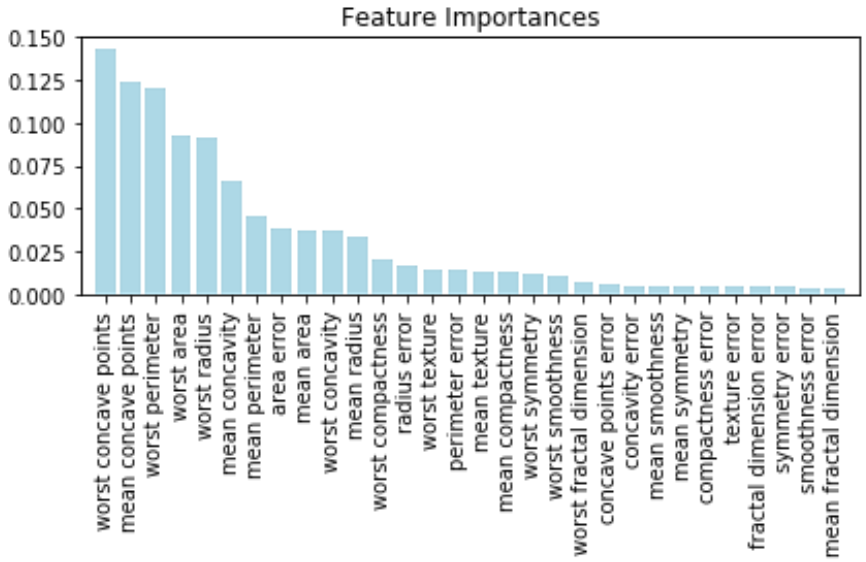
```
indices = np.argsort(importances)[::-1]
for f in range(X_train.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30,
                            feat_labels[indices[f]],
                           importances[indices[f]]))
```

در خروجی نام ویژگی‌ها را بر اساس میزان اهمیتشان داریم:

```
 1) worst concave points           0.142459
 2) mean concave points            0.124031
 3) worst perimeter                0.120270
 4) worst area                     0.092017
 5) worst radius                   0.091058
 6) mean concavity                 0.066016
 7) mean perimeter                 0.045964
 8) area error                     0.038880
 9) mean area                      0.037878
10) worst concavity                0.036824
11) mean radius                    0.033914
12) worst compactness              0.020317
13) radius error                   0.016455
14) worst texture                  0.014988
15) perimeter error                0.014198
16) mean texture                   0.013735
17) mean compactness               0.013019
18) worst symmetry                 0.012067
19) worst smoothness               0.010654
20) worst fractal dimension        0.007528
21) concave points error           0.005631
```

```
22) concavity error                 0.005532
23) mean smoothness                 0.005361
24) mean symmetry                   0.004779
25) compactness error               0.004768
26) texture error                   0.004760
27) fractal dimension error         0.004758
28) symmetry error                  0.004479
29) smoothness error                0.003994
30) mean fractal dimension          0.003667

plt.title('Feature Importances')
plt.bar(range(X_train.shape[1]),
             importances[indices],
             color='lightblue',
             align='center')
plt.xticks(range(X_train.shape[1]),
               feat_labels[indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.tight_layout()
plt.show()
```

بعد از اجرای برنامه نموداری رسم می‌شود که اهمیت ویژگی‌ها را به‌صورت نرمال شده نشان می‌دهد:



شکل۵-۱: اهمیت ویژگی‌های مجموعه دادهٔ سرطان سینه

با توجه به شکل ۵-۱ می‌توانیم نتیجه بگیریم که ویژگی `worst concave points`، بیشترین جداکنندگی را بر اساس کاهش ناخالصی میانگین `10000` درخت تصمیم دارد.

۵-۲-۲- انتخاب ویژگی با روش انتخاب پشت سرهم (SBS[16])

الگوریتم‌های انتخاب ویژگی ترتیبی از خانوادهٔ الگوریتم‌های حریصانه[17] هستند. الگوریتم SBS تلاش می‌کند تا ابعاد فضای ویژگی اولیه را کاهش دهد؛ به‌گونه‌ای که کارایی محاسبات افزایش می‌یابد، درحالی‌که با کاهش کمی در کارایی دسته‌بند مواجه هستیم. البته گاهی که مدل دچار بیش برازش شده است، الگوریتم SBS قدرت پیش‌بینی مدل را افزایش می‌دهد [۵]. در این الگوریتم ویژگی به ترتیب از فضای ویژگی اولیه حذف می‌شود تا زمانی که زیرفضای ویژگی جدیدی به دست آید که تعداد ویژگی دلخواه را دارد. برای اینکه مشخص کنیم کدام ویژگی باید حذف شود، تابع معیار J را تعریف می‌کنیم. این تابع می‌تواند تفاوت کارایی دسته‌بند را قبل و بعد از حذف ویژگی محاسبه کند، بنابراین ویژگی که باید حذف شود، این تفاوت را بیشینه می‌کند. به‌عبارت‌دیگر، در هر مرحله، ما ویژگی را حذف می‌کنیم که بعد از حذف آن کمترین کاهش کارایی را داشته باشیم.

---

16. Sequential Backward Selection

17. Greedy الگوریتم‌های حریصانه برعکس الگوریتم‌های جست‌وجوی همه‌جانبه، که تمام ترکیب‌های ممکن را ارزیابی می‌کنند، در هر مرحله انتخاب‌های بهینهٔ محلی ارائه می‌کنند. اجرای الگوریتم جست‌وجوی همه‌جانبه اکثر اوقات به‌علت پیچیدگی محاسباتی زیاد امکان‌پذیر نیست.

مراحل این الگوریتم به این شرح است:

- **مرحلهٔ اول:** k (بعد فضای جدید) را برابر d (بعد فضای اولیهٔ ویژگی) قرار بده.
- **مرحلهٔ دوم:** ویژگی $\bar{x}$ را به‌گونه‌ای مشخص کن که

$$\bar{x} = argmax\, J(X_k - x) \quad (۴۸)$$

به‌گونه‌ای که $x \epsilon X_k$

- ویژگی $\bar{x}$ را از مجموعهٔ ویژگی‌ها حذف کن:

$$X_{k-1} = X_k - \bar{x}\,; k := k - 1 \quad (۴۹)$$

- اگر k برابر است با تعداد ویژگی‌های دلخواه، به الگوریتم خاتمه بده. اگر نه به مرحلهٔ دوم برو.

متأسفانه الگوریتم SBS در کتابخانهٔ sklearn پیاده‌سازی نشده است. ازآنجایی‌که الگوریتم ساده‌ای است از ابتدا با پایتون آن را پیاده‌سازی [۵] می‌کنیم:

ابتدا کلاس SBS را تعریف می‌کنیم و در تابع __init__ پارامترها را مقداردهی اولیه می‌کنیم. پارامتر k_features را برای تعیین تعداد ویژگی‌های دلخواهی که می‌خواهیم انتخاب شود تنظیم می‌کنیم. به‌صورت پیش‌فرض از accuracy_score از کتابخانهٔ sklearn استفاده می‌کنیم تا کارایی یک مدل دسته‌بند را در زیرمجموعه ویژگی‌ها ارزیابی کنیم. داخل حلقهٔ while متد fit، زیرمجموعه ویژگی‌های ساخته شدهٔ تابع itertools.combinations ارزیابی می‌شوند و کاهش داده می‌شوند تا وقتی که زیرمجموعه ویژگی اندازهٔ دلخواه را پیدا کند. در هر تکرار، accuracy score بهترین زیرمجموعه در لیست self.scores_ برمبنای مجموعه دادهٔ آزمون (X_test) جمع‌آوری می‌شود. اندیس‌های ستونی زیرمجموعه ویژگی نهایی در self.indices_ قرار داده می‌شود که می‌توان با استفاده از آن و متد transform، آرایه‌ای از داده‌های جدید را با ویژگی‌های انتخاب شده به دست آورد. توجه شود که به‌جای محاسبهٔ معیار ویژگی نامناسب در متد fit، ویژگی که در زیرمجموعهٔ بهترین ویژگی‌ها وجود ندارد، حذف می‌شود [۵].

```
from sklearn.base import clone
from itertools import combinations
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
class SBS():
  def __init__(self, estimator, k_features,
```

```
                scoring=accuracy_score,
                test_size=0.25, random_state=1):
        self.scoring = scoring
        self.estimator = clone(estimator)
        self.k_features = k_features
        self.test_size = test_size
        self.random_state = random_state

def fit(self, X, y):
 X_train, X_test, y_train, y_test = \
 train_test_split(X, y, test_size=self.test_size,
                  random_state=self.random_state)
 dim = X_train.shape[1]
 self.indices_ = tuple(range(dim))
 self.subsets_ = [self.indices_]
 score = self._calc_score(X_train, y_train,
                  X_test, y_test, self.indices_)
 self.scores_ = [score]
 while dim > self.k_features:
   scores = []
   subsets = []
   for p in combinations(self.indices_, r=dim-1):
     score = self._calc_score(X_train, y_train,
                              X_test, y_test, p)
     scores.append(score)
     subsets.append(p)
   best = np.argmax(scores)
   self.indices_ = subsets[best]
```

```
            self.subsets_.append(self.indices_)
            dim -= 1
            self.scores_.append(scores[best])
        self.k_score_ = self.scores_[-1]
        return self

    def transform(self, X):
        return X[:, self.indices_]

    def _calc_score(self, X_train, y_train,
                    X_test, y_test, indices):
        self.estimator.fit(X_train[:, indices], y_train)
        y_pred = self.estimator.predict(X_test[:,
        indices])
        score = self.scoring(y_test, y_pred)
        return score
```

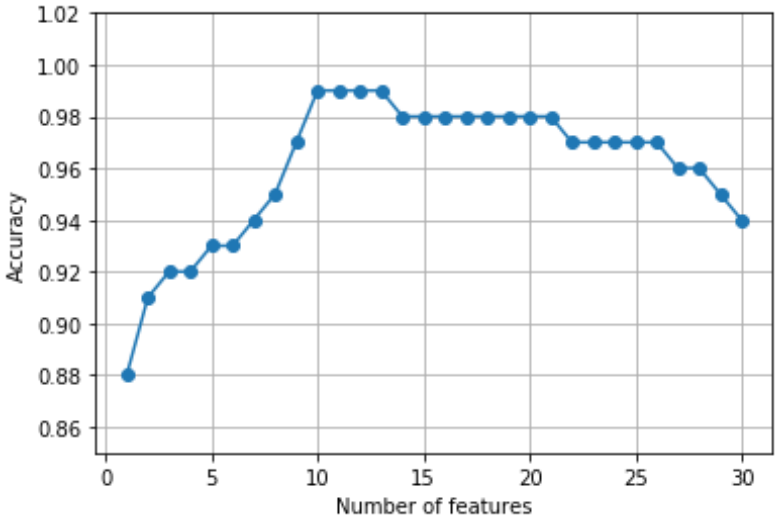
حال ببینیم الگوریتم SBS با دسته‌بند KNN روی مجموعه دادهٔ سرطان سینه چگونه عمل می‌کند. ابتدا مجموعه داده را از کتابخانهٔ sklearn وارد می‌کنیم. سپس آن را به دو مجموعه دادهٔ آموزش و آزمون تقسیم می‌کنیم:

```
from sklearn.datasets import load_breast_cancer
import pandas as pd
cancer=load_breast_cancer()
cancer_df=pd.DataFrame(data=cancer.data[0:,0:],
            columns=cancer.feature_names)

from sklearn.model_selection import train_test_split
X, y = cancer_df.iloc[:, 0:].values, cancer.target
```

```
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3,
random_state=0)
```

داده‌ها را با استفاده از کلاس StandardScaler استاندارد می‌کنیم. سپس مدل KNeigboursClassifier را می‌سازیم. در ادامه از طریق الگوریتم SBS این مدل را به داده‌های استاندارد شده اعمال می‌کنیم و درنهایت دقت دسته‌بند را برحسب تعداد ویژگی‌ها رسم می‌کنیم:

```
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
knn = KNeighborsClassifier(n_neighbors=2)
sbs = SBS(knn, k_features=1)
sbs.fit(X_train_std, y_train)

k_feat = [len(k) for k in sbs.subsets_]
plt.plot(k_feat, sbs.scores_, marker='o')
plt.ylim([0.85, 1.02])
plt.ylabel('Accuracy')
plt.xlabel('Number of features')
plt.grid()
plt.show()
```



شکل ۵-۲: نمودار دقت دسته‌بند KNN برحسب تعداد ویژگی‌هایی که با الگوریتم SBS انتخاب شده‌اند.

۵-۳- استخراج ویژگی[18]

اگرچه مهم‌ترین دلیل استفاده از روش‌های استخراج ویژگی، کارایی محاسبات است، اما این روش‌ها نقش مهمی در کاهش «نفرین ابعاد»[19] دارند [۵]. نفرین ابعاد، مشکلی است که الگوریتم‌های یادگیری ماشین هنگام مواجه شدن با مجموعه داده‌هایی با ابعاد بالا به آن دچار می‌شوند. درک داده‌ها مشکل می‌شود و داده‌ها فاصلهٔ زیادی از هم می‌گیرند. همین مسئله باعث می‌شود الگوریتم‌های یادگیری به‌سختی بتوانند داده‌ها را در یک خوشه یا دسته تشخیص دهند.

۵-۳-۱- تحلیل مؤلفهٔ اصلی (PCA[20])

PCA یک روش تبدیل خطی است که در زمینه‌های مختلفی مانند حذف نویز در سیگنال‌های بازار بورس و تحلیل داده‌های بیان ژن در حوزهٔ بیوانفورماتیک بسیار کاربرد دارد [۵]. PCA برای انتقال داده‌ها به فضای جدید، از وابستگی[21] بین ویژگی‌ها استفاده می‌کند. PCA به ما کمک می‌کند تا جهت‌های واریانس بیشینه را در مجموعه داده با ابعاد بالا بیابیم و مجموعه داده را در فضایی جدید، به‌گونه‌ای که محورهای آن عمود بر هم هستند (به عبارت روشن‌تر، ویژگی‌هایش ناهمبسته است)، منتقل کنیم. این محورها در جهت‌های واریانس بیشینه است (شکل۵-۳).

18. Feature Extraction
19. Curse of Dimensionality
20. Principle Component Analysis
21. correlation

برای انتقال مجموعه داده d بعدی $x = \{x_1. x_2. x_3 \dots . x_d\}$ به فضای k بعدی $z = \{z_1. z_2. z_3 \dots . z_k\}$ به ماتریس انتقال W نیاز است به‌طوری‌که ابعاد آن dxk باشد:

$$z = xw \quad (۵۰)$$

مراحل الگوریتم PCA به این شرح است [۵]:

- **مرحلهٔ اول**: استانداردسازی مجموعه داده d بعدی
- **مرحلهٔ دوم**: ایجاد ماتریس کواریانس[۲۲] ویژگی‌ها

به‌عنوان‌مثال کواریانس دو ویژگی x_j و x_k به‌صورت عبارت (۵۱) است:

$$\sigma_{jk} = \frac{1}{n}\sum_{i=1}^{n}(x_j^i - \mu_j)(x_k^i - \mu_k) \quad (۵۱)$$

به‌گونه‌ای که μ_j میانگین مقادیر ویژگی x_j است و μ_k میانگین مقادیر ویژگی x_k است. n تعداد نمونه‌های مجموعه داده است. کواریانس مثبت به این معنی است که دو ویژگی با هم کاهش یا افزایش می‌یابند. ماتریس کواریانس سه ویژگی به‌صورت عبارت (۵۲) است:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_2^2 & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_3^2 \end{bmatrix} \quad (۵۲)$$

- **مرحلهٔ سوم**: به دست آوردن eigenvectors و eigenvalueها از ماتریس کواریانس (عبارت ()).

Eigenvectorsهای ماتریس کواریانس همان مؤلفه‌های اصلی (جهت‌های واریانس ماکزیمم) است. eigenvalueهای متناظر این بردارها، اندازهٔ این بردارها است.

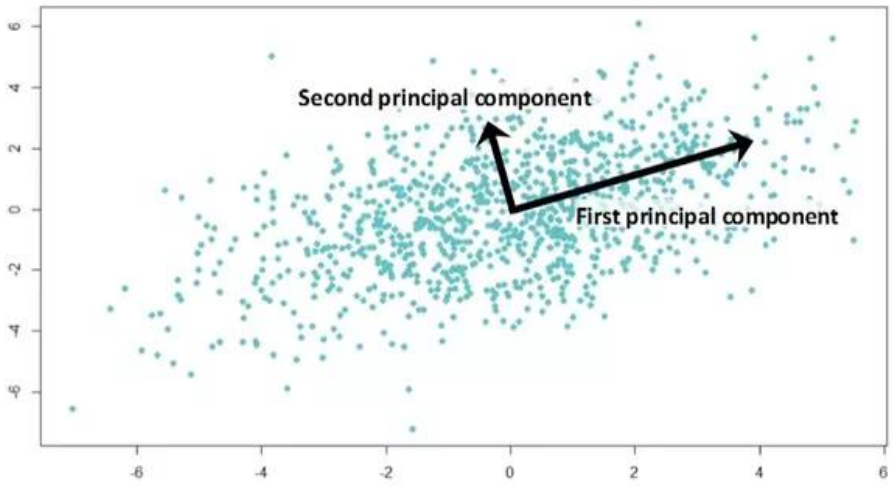
$$\Sigma v = \lambda v \quad (۵۳)$$

v معرف یک eigenvector است. λ یک عدد است و معرف eigenvalue متناظر v است.

- **مرحلهٔ چهارم**: پیدا کردن k تا eigenvectors متناظر با k تا از بزرگ‌ترین eigenvalueها (k ابعاد فضای ویژگی جدید است).
- **مرحلهٔ پنجم**: ساختن ماتریس w با k تا eigenvector یافته شده در مرحلهٔ چهارم.
- **مرحلهٔ ششم**: انتقال مجموعه دادهٔ d بعدی x با ماتریس انتقال w به فضای جدید k بعدی (عبارت (اول)).

---

[22]. covariance



شکل ۵-۳: دو مؤلفه اصلی به دست آمده با روش PCA [۳۷]

✓ **مثال:**

در این مثال می‌خواهیم الگوریتم PCA را روی مجموعه دادهٔ مربوط به سرطان سینه اعمال کنیم [۳۸]:

```
from sklearn.datasets import load_breast_cancer
cancer=load_breast_cancer()
```

برای اینکه کمی در مورد مجموعه داده‌ای که قرار است روی آن کار کنیم بدانیم:

```
print(cancer.DESCR)
```

مقادیر ویژگی‌های این مجموعه داده از تصاویر توده‌های سینه محاسبه شده‌اند. این مجموعه داده 596 نمونه و 30 ویژگی دارد که از بین این نمونه‌ها، ۲۱۲ نمونه بدخیم و بقیه خوش‌خیم هستند. پس متغیر هدف دارای دو مقدار 0 و 1 است. فقط برای اینکه مطمئن بشویم 0 نمایش‌دهندهٔ بدخیم است می‌توانیم با یک دستور ساده این اطمینان را به دست آوریم:

```
print(len(cancer.data[cancer.target==0]))
```

در خروجی خواهیم داشت: 212

قبل از اعمال PCA به مجموعه داده خوب است کمی دربارهٔ تأثیر ویژگی‌ها در مقدار ویژگی هدف بدانیم:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# we will have a figure which has 3 columns each has 10 features
fig,axes=plt.subplots(10,3, figsize=(12,9))

malignant=cancer.data[cancer.target==0]
benign=cancer.data[cancer.target==1]

# flat axes with numpy ravel
ax=axes.ravel()

for i in range(cancer.data.shape[1]):
    _,bins=np.histogram(cancer.data[:,i],bins=40)
    # red color for malignant class

     ax[i].hist(malignant[:,i],bins=bins,color='r',
     alpha=0.5)
    # green color for malignant class
    # alpha is for transparency in the overlapped
    #region

     ax[i].hist(benign[:,i],bins=bins,color='g',alpha
     =0.3)

    ax[i].set_title(cancer.feature_names[i],fontsiz
    e=9)
    #the x-axis co-ordinates are not so useful
    #here.
    ax[i].axes.get_xaxis().set_visible(False)
    ax[i].set_yticks(())
```

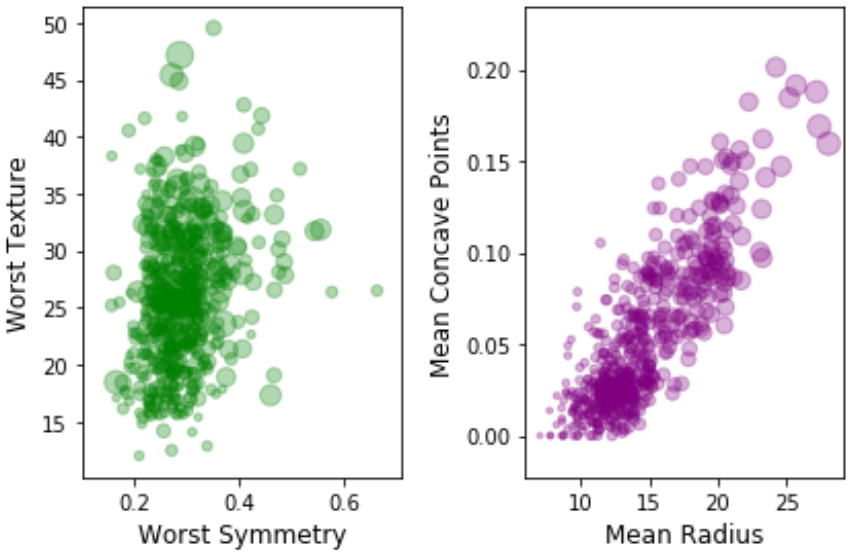
```
ax[0].legend(['malignant','benign'],loc='best',
fontsize=8)
# let's make good plots
plt.tight_layout()
plt.show()
```

همان‌طور که از هیستوگرام‌های حاصل از برنامه در شکل ۵-۵ مشخص است، ویژگی‌هایی مانند fractal dimension نقش کمی در جداسازی دو دسته توده دارند، درحالی‌که ویژگی‌هایی مانند worst concave points یا worst perimeter ویژگی‌های مفیدی هستند که در جداسازی دسته‌ها بسیار مؤثرند. از طرف دیگر مقادیر بسیاری از ویژگی‌ها به هم وابسته است:

```
import pandas as pd


# just convert the scikit learn data-set to pandas
data-frame:
cancer_df=pd.DataFrame(cancer.data,columns=cancer.f
eature_names)
#fisrt plot
plt.subplot(1,2,1)
plt.scatter(cancer_df['worst
symmetry'],cancer_df['worst
texture'],s=cancer_df['worst
area']*0.05,color='g',label='check',alpha=0.3)


plt.xlabel('Worst Symmetry',fontsize=12)
plt.ylabel('Worst Texture',fontsize=12)


#second plot
plt.subplot(1,2,2)
plt.scatter(cancer_df['mean
radius'],cancer_df['mean concave
```

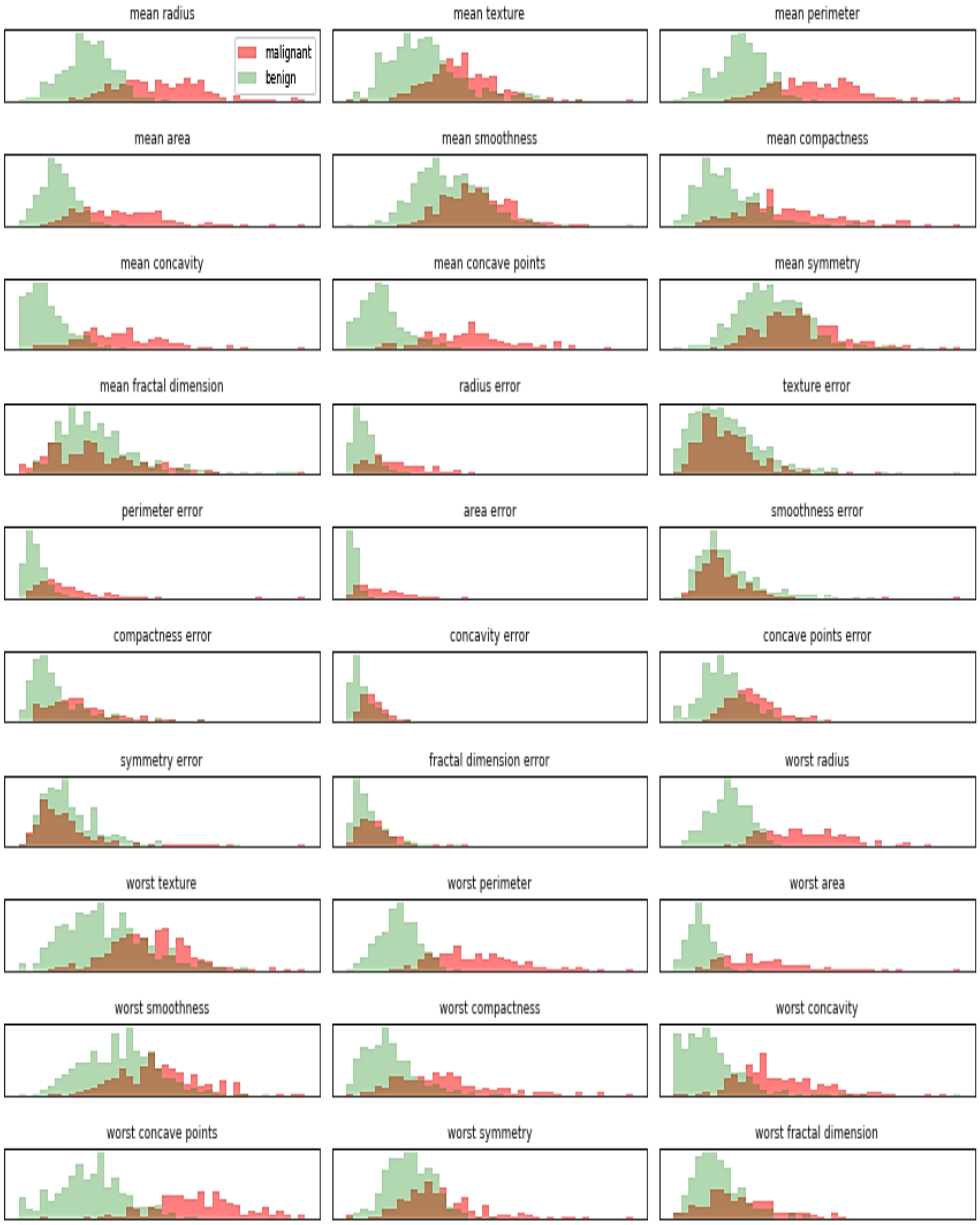
```
points'],s=cancer_df['mean area']*0.05,
color='purple', label='check', alpha=0.3)


plt.xlabel('Mean Radius',fontsize=12)
plt.ylabel('Mean Concave Points',fontsize=12)
plt.tight_layout()
plt.show()
```

وابستگی ویژگی‌ها را می‌توان به‌سادگی در خروجی (شکل ۵-۴) مشاهده کرد. همان‌طور که دیدید در شکل اول اندازهٔ قلم را ضریبی از ویژگی `worst area` قرار دادیم و کاملاً مشخص است که با افزایش مقدار ویژگی `Worst Symmetry` اندازهٔ قلم (اندازهٔ دایره‌ها) کاهش می‌یابد. در شکل دوم نیز اندازهٔ قلم را ضریبی از مقدار ویژگی `mean area'` قرار دادیم. همان‌طور که مشهود است با افزایش مقادیر ویژگی‌های `Mean Radius` و `Mean Concave Points` اندازهٔ قلم (اندازهٔ دایره‌ها) افزایش می‌یابد.



شکل ۵-۴: مجموعه دادهٔ سرطان سینه بر اساس تعداد کمی از ویژگی‌ها

PCA روشی است که ابعاد فضای ویژگی را به‌گونه‌ای کاهش می‌دهد که ابعاد جدید حاصل شده بر هم عمود هستند (به هم وابسته نیستند و از هم مستقل هستند).



شکل ۵-۵: هیستوگرام دو دسته توده‌های خوش‌خیم و بدخیم مبتنی بر ۳۰ ویژگی از مجموعه دادهٔ سرطان سینه

قبل از اعمال الگوریتم PCA برای استانداردسازی مقادیر ویژگی‌ها از کلاس StandardScalar استفاده می‌کنیم. در این کلاس مقادیر ویژگی از میانگین آن ویژگی کم می‌شود و بر اساس واریانس آن ویژگی، مقیاس می‌شود:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

```
scaler.fit(cancer.data)
X_scaled=scaler.transform(cancer.data)
```

اکنون مجموعه داده آماده است تا الگوریتم PCA به آن اعمال شود. هنگام استفاده از کلاس PCA می‌توانیم مشخص کنیم به چه تعداد از مؤلفه‌های اصلی نیازمندیم:

```
from sklearn.decomposition import PCA
pca=PCA(n_components=3)
u=pca.fit(X_scaled)
X_pca=pca.transform(X_scaled)
```

می‌توانیم میزان نقش مؤلفه‌های اصلی را در واریانس کل با به دست آوردن نرخ واریانس به دست آوریم:

```
ex_variance=np.var(X_pca,axis=0)
ex_variance_ratio=ex_variance/np.sum(ex_variance)
print(ex_variance_ratio)
```

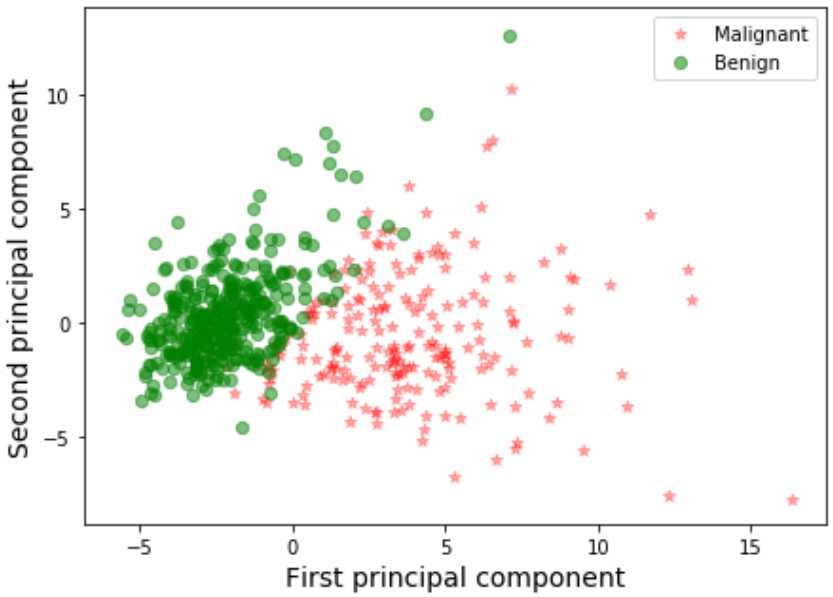
در خروجی داریم:

```
[0.60950217 0.2611802  0.12931763]
```

همان‌طور که از خروجی برنامه مشخص است، دو مؤلفهٔ اول در ۸۷درصد از کل واریانس نقش دارند. بنابراین انتخاب دو مؤلفهٔ اول کافی است. ازآنجایی‌که مؤلفه‌های PCA بر هم عمود هستند و به هم مرتبط نیستند، می‌توانیم این انتظار را داشته باشیم که دو دسته توده‌های خوش‌خیم و بدخیم را به‌صورت مجزا ببینیم:

```
Xax=X_pca[:,0]
Yax=X_pca[:,1]
cdict={0:'red',1:'green'}
marker={0:'*',1:'o'}
labl={0:'Malignant',1:'Benign'}
alpha={0:0.3,1:0.5}


fig,ax=plt.subplots(figsize=(7,5))
fig.patch.set_facecolor('white')
```

```
for l in np.unique(cancer.target):
    ix=np.where(cancer.target==l)
    plt.scatter(Xax[ix],Yax[ix],c=cdict[l],s=40,

label=labl[l],marker=marker[l],alpha=alpha[l])
plt.xlabel("First principal component",fontsize=14)
plt.ylabel("Second principal
component",fontsize=14)
# Place a legend on the axes:
plt.legend()
plt.show()
```



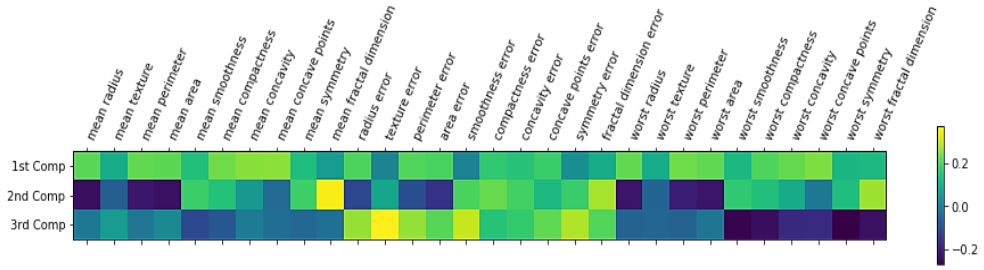
شکل ۵-۶: نمودار دو دسته از داده‌های سرطان سینه بر اساس دو مؤلفهٔ اصلی ویژگی‌های توده‌های سرطانی

همان‌طور که در شکل ۵-۶ مشخص است، نمونه‌های دو دستهٔ مجموعه داده به‌خوبی قابل‌تفکیک هستند. PCA در اینجا شبیه یک طبقه‌بند خطی عمل کرده است.

مؤلفه‌های اصلی فقط با استفاده از اطلاعات ویژگی‌ها به دست آمده است و در محاسبهٔ آن‌ها برچسب‌ها نقشی نداشته‌اند. بنابراین روش PCA، از جمله روش‌های بدون ناظر است.

می‌توان برای نمایش میزان وابستگی سه مؤلفهٔ اصلی به ویژگی‌های اصلی از نمودار heat استفاده کرد:

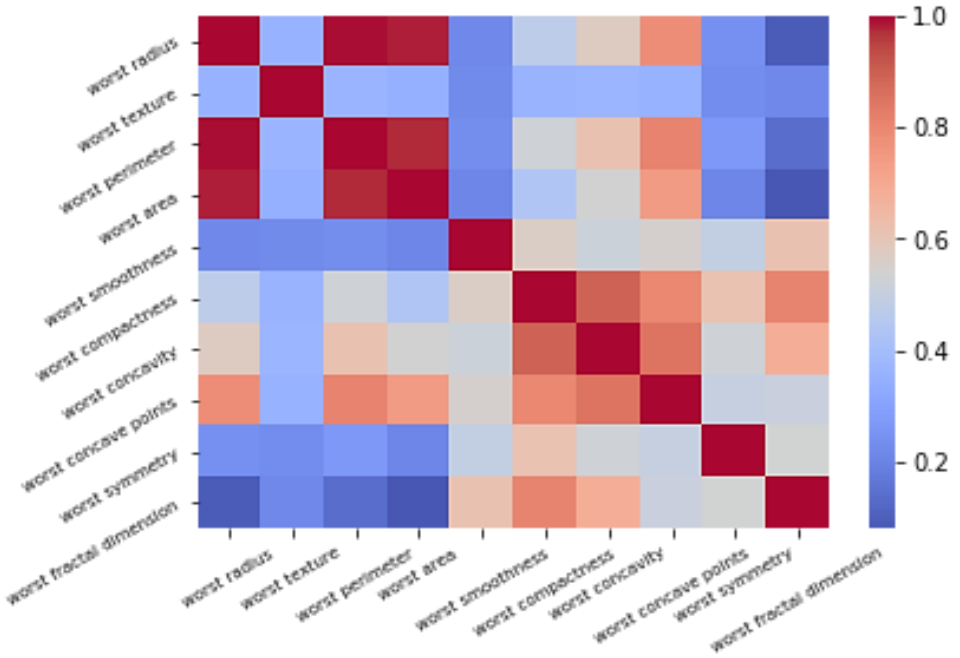
```
plt.matshow(pca.components_,cmap='viridis')
plt.yticks([0,1,2],['1st Comp','2nd Comp','3rd Comp'],fontsize=10)
plt.colorbar()
plt.xticks(range(len(cancer.feature_names)),cancer.feature_names,rotation=65,ha='left')
plt.tight_layout()
plt.show()
```



شکل ۵-۷: میزان وابستگی سه مؤلفهٔ اصلی به ویژگی‌های اصلی

می‌توان میزان همبستگی ویژگی‌ها را با نمودار heatmap از کتابخانهٔ seaborn رسم کرد. در اینجا ما همبستگی ویژگی‌های 'worst' را به هم بررسی کرده‌ایم:

```
feature_worst=list(cancer_df.columns[20:31])
s=sns.heatmap(cancer_df[feature_worst].corr(),cmap='coolwarm')
s.set_yticklabels(s.get_yticklabels(),rotation=30,fontsize=7)
s.set_xticklabels(s.get_xticklabels(),rotation=30,fontsize=7)
plt.show()
```



شکل ۵-۸: نمایش میزان همبستگی تعدادی از ویژگی‌های مجموعه دادهٔ سرطان

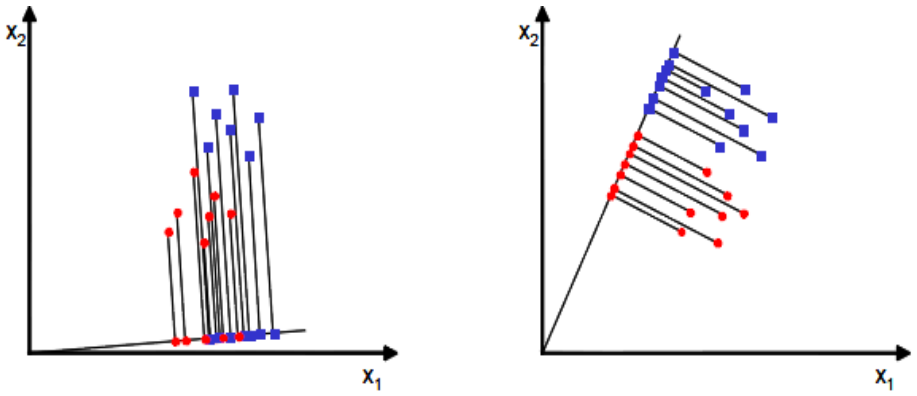
۵-۳-۲- تحلیل تفکیک‌کنندهٔ خطی (LDA[۲۳])

در این بخش هدف از اعمال LDA، کاهش ابعاد است که بر اساس فاصلهٔ بین داده‌های هر دسته از میانگین آن و فاصلهٔ بین دسته‌ها استوار شده است. فرض کنید ما یک مجموعهٔ D بعدی مانند $\{x^1, x^2, \ldots, x^d\}$ داشته باشیم. هر x^i معرف یکی از ستون‌های (ویژگی‌های) ماتریس X است. در این روش نگاشت نمونه‌ها از فضای X به فضای Y انجام می‌شود؛ به‌گونه‌ای که ابعاد نمونه‌ها در فضای Y به $C-1$ بعد کاهش می‌یابد. C مشخص‌کنندهٔ تعداد دسته‌های مجموعه داده است. معادلهٔ خط این نگاشت به‌صورت عبارت (۵۴) است:

$$Y = W^T X \qquad (۵۴)$$

توضیحات بالا برای دو بعد در شکل ۵-۹ نمایش داده شده است. از بین همهٔ خطهایی که می‌توان انتخاب کرد، باید یک خط را که بیشترین جداکننده برای بردارهاست، انتخاب کنیم.

[23]. Linear Discriminant Analysis



شکل ۵-۹: انتخاب یک خط برای دو بعد [۳۹]

در این روش، برای انتخاب خطی با بیشترین جداکنندگی برای بردارها، سعی بر آن است تا نسبت فاصلهٔ بین دسته‌ها بر فاصلهٔ بین داده‌های هر دسته از میانگین آن دسته بیشینه شود. در زیر مراحل به دست آوردن LDA برای C دسته توضیح داده شده است [۴۰]:

- **مرحلهٔ اول:** میانگین نمونه‌های هر دسته از رابطهٔ () به دست می‌آید (به‌طوری‌که ω_i معرف دستهٔ i است، N_i تعداد نمونه‌های دستهٔ i است):

$$\mu_i = \frac{1}{N_i}\sum_{x\in\omega_i} x \;\; and \;\; \tilde{\mu}_i = \frac{1}{N_i}\sum_{y\in\omega_i} y = \frac{1}{N_i}\sum_{x\in\omega_i} w^T x = w^T\mu_i \qquad (۵۵)$$

- **مرحلهٔ دوم:** محاسبهٔ دو مفهوم فاصلهٔ درون کلاسی[۲۴] و فاصلهٔ بین کلاس‌ها[۲۵] که به ترتیب در رابطه‌های (۵۶ و ۵۷) بیان شده است:

$$S_i = \sum_{x\in\omega_i}(x-\mu_i)(x-\mu_i)^T \qquad (۵۶)$$

$$S_w = \sum_{i=1}^{c} S_i \times N_i \qquad (۵۷)$$

به‌گونه‌ای که S_i ماتریس کواریانس کلاس ω_i را تشکیل می‌دهد.

$$S_B = \sum_{i=1}^{c} N_i(\mu_i-\mu)(\mu_i-\mu)^T \qquad (۵۸)$$

درحالی‌که

$$\mu = \frac{1}{N}\sum_{\forall x} x = \frac{1}{N}\sum_{x\in\omega_i} N_i\mu_i \qquad (۵۹)$$

---

۲۴. Within-class scatter
۲۵. Between-class sctter

در LDA می‌خواهیم نسبت فاصلهٔ بین کلاس‌ها به فاصلهٔ درون کلاسی در فضای جدید (Y) را به بیشترین مقدار برسانیم. این دو مقدار به ترتیب عبارت‌اند از: S_B و S_w. به همین علت، از این نسبت (عبارت (۶۰)) مشتق می‌گیریم و برابر صفر قرار می‌دهیم که درنهایت عبارت (۶۱) به دست می‌آید:

$$J(w) = \frac{|S_B|}{|S_w|} \qquad (۶۰)$$

$$S_w^{-1} S_B w = Jw \qquad (۶۱)$$

- **مرحلهٔ سوم:** به دست آوردن مقادیر ویژه و بردارهای ویژه در عبارت (۶۱) و انتخاب k بردار ویژه (ماتریس w) ی متناظر با k مقدار ویژه (بردار J).

  در اینجا به‌سادگی و با حل مسئلهٔ مقدار ویژهٔ معادلهٔ (۶۱)، بردار J به دست می‌آید و با جایگذاری J در عبارت ۶۱ ماتریس W به دست می‌آید. این ماتریس امکان به دست آوردن خطی با بیشترین قابلیت تفکیک را به دست می‌دهد.
- **مرحلهٔ چهارم:** انتقال فضای ویژگی d بعدی به فضای جدید با ماتریس انتقال w.

✓ **مثال:**

اجازه بدهید درک الگوریتم LDA را با یک مثال [۴۱] ادامه بدهیم:

مجموعه داده‌ای که در این مثال استفاده می‌کنیم مجموعه دادهٔ مربوط به «اصالت چک‌های بانکی (یا اسکناس)» است. تصاویر این مجموعه داده از دوربینی که برای بازرسی استفاده می‌شود، تهیه شده است. تصاویر دارای سطوح خاکستری و ابعاد 400x400 هستند. از انتقال Wavelet برای استخراج ویژگی تصاویر استفاده شده است.

ویژگی‌های این مجموعه داده به شرح زیر است:

- واریانس تصاویر تبدیل شده با Wavelet
- Curtosis تصاویر تبدیل شده با Wavelet (یک پارامتر آماری است)
- Skewness تصاویر تبدیل شده با Wavelet (یک پارامتر آماری است)
- انتروپی تصاویر

دستهٔ 0، مربوط به تصاویر چک‌های بانکی جعلی است و دستهٔ 1، مربوط به تصاویر چک‌های بانکی معتبر است. این مجموعه داده دارای ۱۳۷۲ نمونه است که ۷۶۲ عدد از تصاویر آن مربوط به تصاویر جعلی است و ۶۱۰ عدد از تصاویر معتبر هستند.

ابتدا کتابخانه‌های موردنیاز را فراخوانی می‌کنیم:

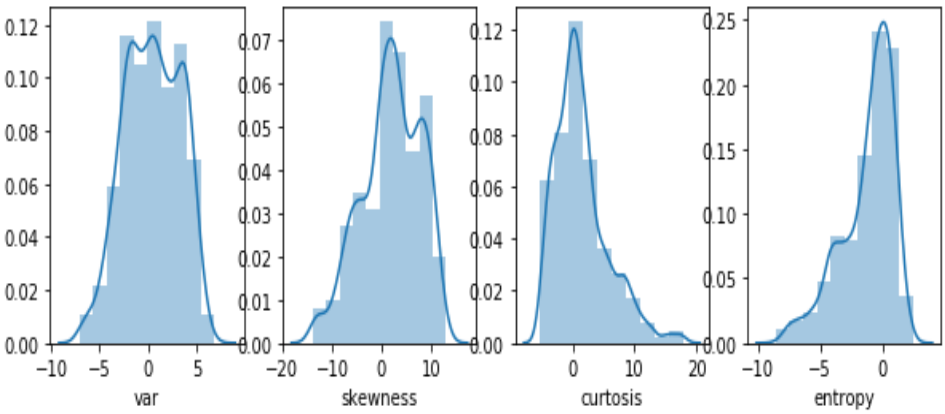
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

سپس مجموعه داده را لود می‌کنیم و با استفاده از تابع read_csv از کتابخانهٔ pandas آن را به یک DataFrame تبدیل می‌کنیم. در تابع read_csv وقتی مقدار پارامتر index_col را false قرار می‌دهیم، pandas را مجبور می‌کنیم ستون اول را که حاوی اسم ویژگی‌ها است، در نظر نگیرد:

```
columns=["var","skewness","curtosis","entropy","cla
ss"]
df=pd.read_csv("http://archive.ics.uci.edu/ml/machi
ne-learning-
databases/00267/\data_banknote_authentication.txt",
index_col=False,names=columns)
```

توزیع داده برحسب ویژگی‌ها به‌صورت شکل ۵-۱۰ است:

```
f,ax=plt.subplots(1,4,figsize=(10,3))
vis1=sns.distplot(df["var"],bins=10,ax=ax[0])
vis2=sns.distplot(df["skewness"],bins=10,ax=ax[1])
vis3=sns.distplot(df["curtosis"],bins=10,ax=ax[2])
vis4=sns.distplot(df["entropy"],bins=10,ax=ax[3])
f.savefig('sublot.png')
```



شکل ۵-۱۰: توزیع نمونه‌های مجموعه داده برحسب ویژگی‌هایش

حال با استفاده از یک دستور ساده، توزیع داده‌ها را برحسب ویژگی‌ها دوبه‌دو نسبت به هم می‌سنجیم. وقتی پارامتر hue از تابع pairplot را برابر متغیر "class" قرار می‌دهیم، داده‌های هر دسته قابل تفکیک با نمونه‌های دسته دیگر نمایش داده می‌شوند (شکل ۵-۱۱):

```
sns.pairplot(df,hue="class")
```

حال بردارهای 4 بعدی میانگین را برای هر دو کلاس محاسبه می‌کنیم. در اینجا برعکس PCA نیاز به استانداردسازی داده‌ها نیست، زیرا در LDA ما از نسبت ماتریس‌های کواریانس بین‌کلاسی و درون‌کلاسی استفاده می‌کنیم. درحالی‌که در PCA از خود ماتریس کواریانس استفاده می‌کنیم.

```
mean_vec=[]
for i in df["class"].unique():
  mean_vec.append(df[df["class"]==i].mean()[:4])
```

مرحلهٔ بعد محاسبهٔ ماتریس‌های کواریانس درون‌کلاسی و بین‌کلاسی است:

```
SW=np.zeros((4,4))
for i in range(2):#2 is the number of classes
  per_class_sc_mat=np.zeros((4,4))
  for j in range(df[df["class"]==i].shape[0]):
    row, mv=df.loc[j][:4].T,mean_vec[i].T
    per_class_sc_mat += (row-mv).dot((row-mv).T)
  SW += per_class_sc_mat
print('within-class Scatter Matrix:\n', SW)
```

```
overall_mean=np.array(df.drop("class",axis=1).mean(
))


SB=np.zeros((4,4))
for i in range(2):#2 is the number of classes
  SB=np.zeros((4,4))
  n = df[df["class"]==i].shape[0]
  mv=mean_vec[i].T
  overall_mean = overall_mean.T # make column vector
   SB += n * (mv - overall_mean).dot((mv -
  overall_mean).T)
print('between-class Scatter Matrix:\n', SB)
```

در خروجی داریم:

```
within-class Scatter Matrix:
 [[ 12894.02618964  12388.17949578  -5025.6093967
1110.90896782]
 [ 12388.17949578  34177.22271176 -12505.24344685
-4576.31987213]
 [ -5025.6093967  -12505.24344685   7858.82866205
1824.14383828]
 [  1110.90896782  -4576.31987213   1824.14383828
2876.82304762]]
between-class Scatter Matrix:
 [[ 5.82112206e+03  7.37300999e+03 -1.89802442e+03
1.39029178e+02]
 [ 7.37300999e+03  9.33862504e+03 -2.40403016e+03
1.76093803e+02]
 [-1.89802442e+03 -2.40403016e+03  6.18866372e+02 -
4.53315997e+01]
```

```
 [ 1.39029178e+02  1.76093803e+02 -4.53315997e+01
3.32051315e+00]]
```

حال باید مقادیر ویژه به همراه بردارهای ویژهٔ ماتریس معکوس SW ضرب در ماتریس SB را به دست آوریم:

```
e_vals, e_vecs = np.linalg.eig(np.linalg.inv(SW).dot(SB))
print('Eigenvectors \n%s' %e_vecs)
print('\nEigenvalues \n%s' %e_vals)
```

در خروجی داریم:

```
Eigenvectors
[[ 0.74375324 -0.27164236  0.12880279 -0.13376905]
 [ 0.34589513  0.21717358 -0.08720775 -0.00480773]
 [ 0.57194183 -0.05802911  0.12801313 -0.36134456]
 [-0.00837995 -0.93577705  0.97949849  0.92277453]]

Eigenvalues
[ 5.27056121e-01  1.38917912e-16  5.71158160e-17 -
5.14733112e-17]
```

اکنون باید k تا بردار ویژهٔ متناظر با k تا بیشترین مقادیر ویژه را انتخاب کنیم. در اینجا برای سادگی نمایش باید 2 بردار ویژهٔ متناظر با 2 مقدار ویژهٔ بزرگ‌تر را انتخاب کنیم. ماتریس انتقال W برابر است با:

```
W=e_vecs[:,:2]
```

حال داده‌ها را (بدون در نظر گرفتن ستون مربوط به برچسب‌ها) با ماتریس انتقال W به فضای جدید منتقل می‌کنیم:

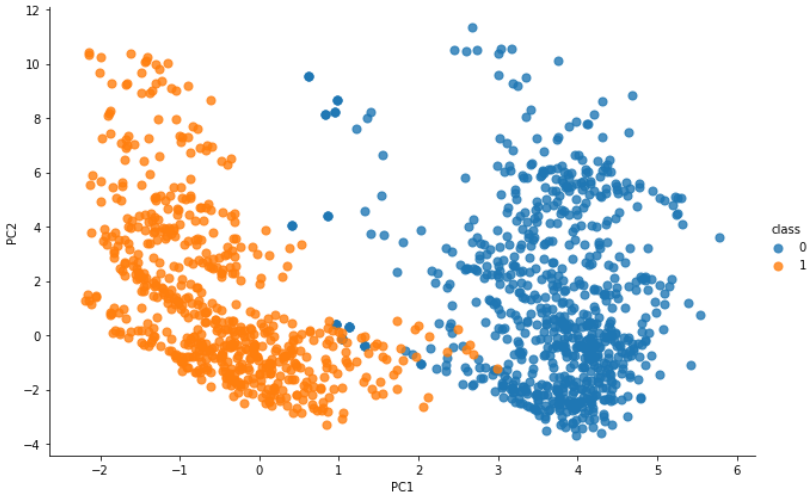
```
X_lda=df.iloc[:,:4].dot(W)
```

دو ستون X_lda (داده‌ها در فضای جدید دوبعدی) را به DataFrame اضافه می‌کنیم و نام ستون‌ها را "PC1" و "PC2" نام‌گذاری می‌کنیم:

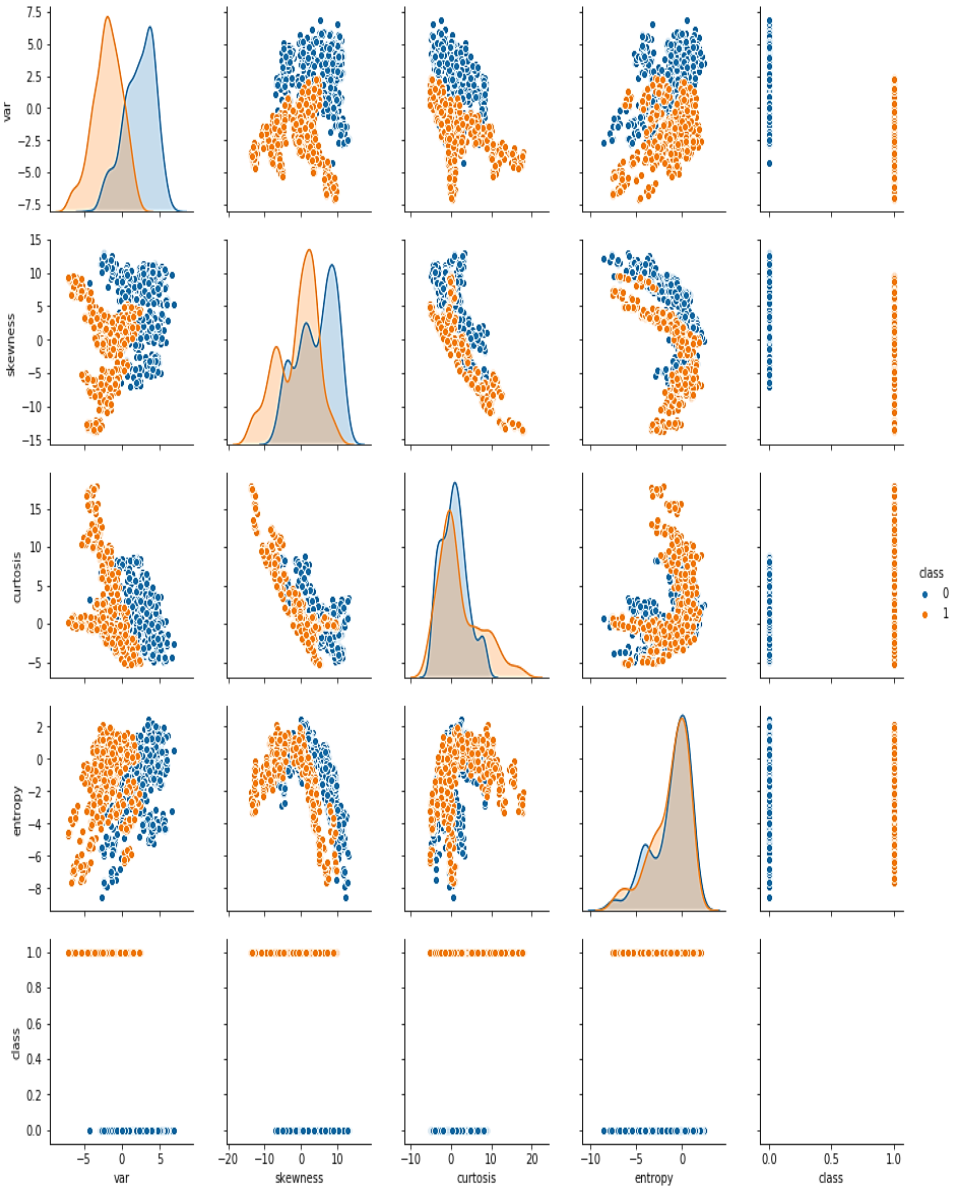
```
df["PC1"]=X_lda.iloc[:,0]
df["PC2"]=X_lda.iloc[:,1]
```

حال داده‌ها را در فضای جدید دوبعدی رسم می‌کنیم:

```
vis=sns.lmplot(data=df[["PC1","PC2","class"]],x="PC
1",
y="PC2",fit_reg=False,hue="class",size=6,aspect=1.5
,scatter_kws={'s':50})

vis.savefig("lda.png")
```



شکل ۵-۱۱: ترسیم مجموعه داده مبتنی بر دو مؤلفهٔ اصلی به دست آمده با روش LDA

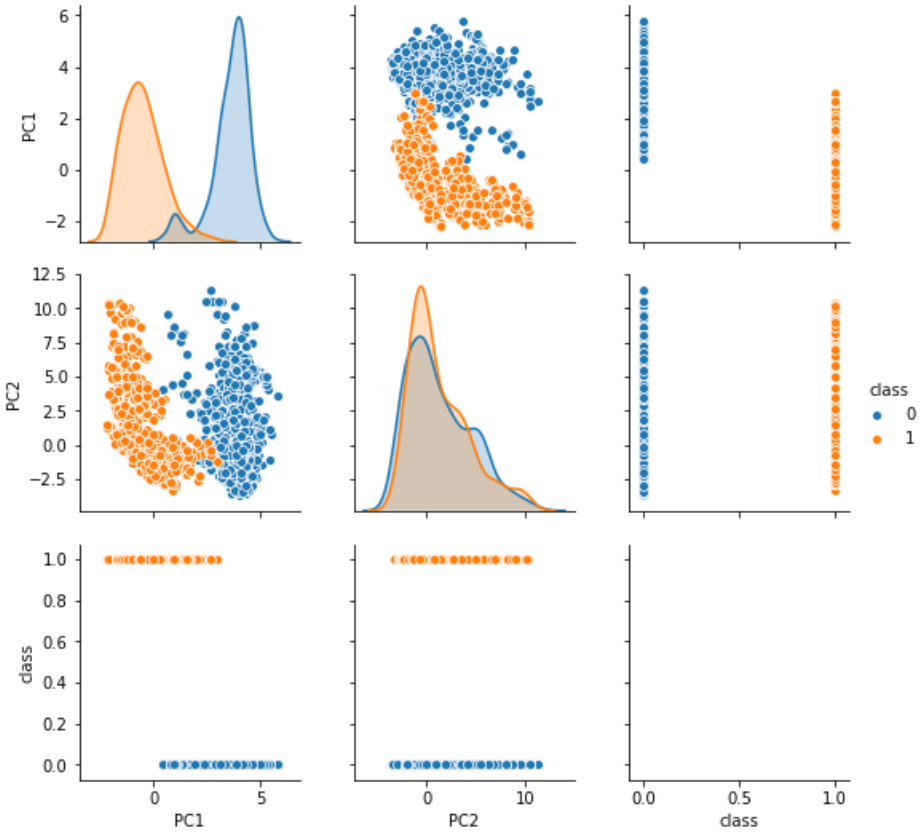


شکل ۵-۱۲: توزیع داده‌ها برحسب ویژگی‌های مجموعه داده دوبه‌دو نسبت به هم رسم شده‌اند.

حال با استفادهٔ مجدد از تابع `pairplot`، توزیع داده‌ها را برحسب ویژگی‌های جدید (`PC1` و `PC2`) دوبه‌دو نسبت به هم رسم می‌کنیم:

```
sns.pairplot(df[["PC1","PC2","class"]],hue="class")
```

خروجی در شکل ۵-۱۳ نشان داده شده است:



شکل ۵-۱۳: توزیع داده برحسب ویژگی‌های جدید

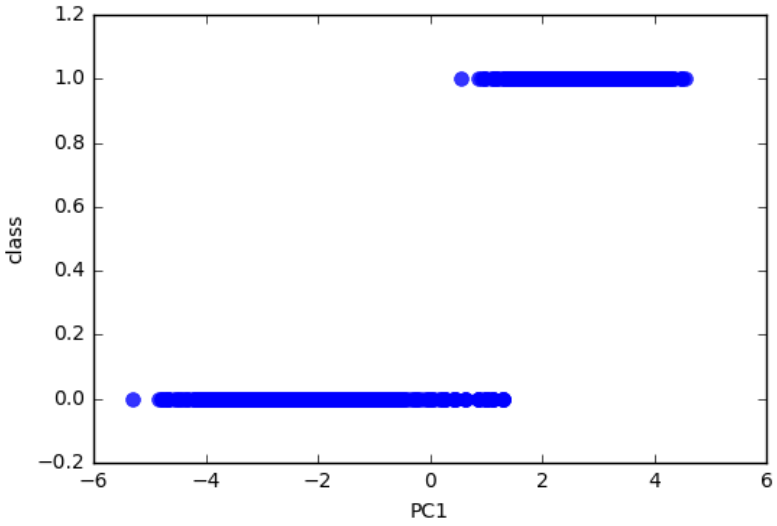
با توجه به شکل‌های ۵-۱۱ و ۵-۱۳مؤلفهٔ اصلی اول (PC1) برای جداسازی دو کلاس کافی است.

می‌توان از کتابخانهٔ `scikit learn` نیز برای پیاده‌سازی هرچه سریع‌تر روش LDA استفاده کرد:

```
from sklearn.discriminant_analysis import \
LinearDiscriminantAnalysis as LDA
model=LDA(n_components=3)
X=df.iloc[:,:4]
y=df["class"]
X_lda=model.fit(X,y)
X_lda=model.transform(X)
df["PC1"]=X_lda[:,0]
```

```
sns.regplot(data=df[["PC1","class"]], x="PC1",
y="class", fit_reg=False, scatter_kws={'s':50})
```

همان‌طور که در خروجی (شکل ۵-۱۴) می‌بینیم، داده‌ها برحسب اولین مؤلفهٔ اصلی به‌خوبی قابل جداسازی هستند.



شکل ۵-۱۴: ترسیم داده‌ها برحسب اولین مؤلفهٔ اصلی و دسته‌ای که به آن تعلق دارند.

فصل ششم

ارزیابی نهایی مدل

PYTHON

در یک سیستم طبقه‌بندی، ابتدا دسته‌بند با داده‌های برچسب‌داری که معرف مجموعه داده آموزش است، آموزش داده می‌شود و سپس با داده‌های برچسب‌داری که مجموعه داده آزمون نامیده می‌شود، عملکرد آن ارزیابی می‌شود.

به‌طورکلی در سیستم‌های دسته‌بندی و تشخیص، برای بررسی میزان موفقیت و کارایی سیستم‌ها از معیارهایی برای ارزیابی بهره‌برداری می‌شود که برای مثال می‌توان از معیارهایی مثل صحت[۱]، حساسیت[۲] و اختصاصی[۳] نام برد، که با یک مثال ساده به تشریح آن‌ها خواهیم پرداخت (این معیارها به‌تفصیل در فصل دو در قسمت مربوط به شبکه‌های عصبی مطرح شده است).

به‌عنوان‌مثال در سیستم طبقه‌بندی یک بیماری خاص داریم:

- صحت (یا درصد درستی) برابر است با نسبت تمام مواردی که به‌صورت صحیح طبقه‌بندی شده‌اند (اعم از بیمار و سالم) بر تمام موارد. این معیار دارای دید و دامنه‌ای جامع‌تر از عملکرد سیستم‌های طبقه‌بندی‌کنندهٔ بیماری‌ها است. درواقع صحت، درجهٔ انطباق یک آزمون با واقعیت است.
- حساسیت عبارت است از درصد موارد مثبت در جمعیت بیمار.
- اختصاصی عبارت است از درصد موارد منفی در جمعیت سالم تحت مطالعه.

قبل از محاسبهٔ این معیارها، ما به استراتژی مناسبی برای تقسیم مجموعه داده به بخش‌های آموزش و آزمون نیازمندیم تا بتوانیم ارزیابی مناسبی از عملکرد سیستم داشته باشیم که به آن روش اعتبارسنجی می‌گویند.

آنچه در این فصل مطرح می‌شود:

- `Kfold`
- `Leave-one-out`
- `Hold out`
- جایگزینی مجدد
- جایگشت تصادفی
- `pipeline` در پایتون
- پیاده‌سازی `pipeline` به همراه `KFold`

---

1. Accuracy
2. Sensitivity
3. Specificity

۶-۱- روش‌های اعتبارسنجی

تاکنون مراحل مختلف استخراج و انتخاب، نرمالیزه کردن، کاهش بعد و طبقه‌بندی ویژگی‌ها و نیز معیارهای ارزیابی را بررسی کردیم. حال به روش‌هایی برای ارزیابی عملکرد یک طبقه‌بند نیازمندیم. از بین داده‌های برچسب‌دار، روش‌های مختلفی وجود دارد که بخشی را به‌عنوان دادهٔ آموزش و بخشی را به‌عنوان دادهٔ آزمون انتخاب می‌کنند. برای این منظور روش‌های متعددی پیشنهاد شده است که در اینجا مرور کوتاهی بر تعدادی از آن‌ها خواهیم داشت.

۶-۱-۱- روش K-Fold

در این روش مجموعه دادهٔ **آزمون** به‌صورت تصادفی به K گروه افراز می‌شود. سپس K-1 گروه از داده‌ها برای آموزش و یک گروه برای آزمون الگوریتم استفاده می‌شوند. تعداد دفعات اجرای این فرایند K است و K مدل با تخمین‌های مختلف به دست می‌آید و درنهایت از کارایی مدل‌ها میانگین‌گیری می‌شود. به‌طورمعمول از روش K-fold برای تنظیم مدل یادگیری استفاده می‌شود. به‌عبارت‌دیگر، از این روش برای یافتن مقادیر بهینه برای ابرپارامترها[4] استفاده می‌شود تا کارایی مناسبی برای مدل به دست آید. به‌محض پیدا کردن مقادیر راضی‌کننده برای ابرپارامترها، مدلی را که با آموزش مجموعه دادهٔ آموزش به دست آمده حفظ می‌کنیم و با استفاده از یک مجموعه دادهٔ آزمون مستقل، کارایی نهایی مدل را تخمین می‌زنیم [۵].

۶-۱-۲- روش Leave-One-Out

در این روش یک داده از مجموعه داده‌های موجود کنار گذاشته می‌شود و با مابقی داده‌ها طبقه‌بند آموزش داده می‌شود. سپس دادهٔ کنار گذاشته شده را به‌عنوان آزمون به طبقه‌بند می‌دهند. این عمل به ازای تک‌تک داده‌های موجود تکرار می‌شود و نهایتاً درصد صحت طبقه‌بند محاسبه می‌شود. این روش حالت خاصی از روش K-Fold است.

۶-۱-۳- روش Hold-out

دادهٔ آموزش در دست به دو قسمت تقسیم می‌شود یکی به‌عنوان مجموعه دادهٔ آموزش و دیگری برای آزمون. اشکال اصلی این روش آن است که تعداد دادهٔ مربوط به آموزش و آزمون طبقه‌بندی را کاهش می‌دهد. مشکل دیگر آن است که نمی‌دانیم نسبت این دو زیرمجموعه باید چگونه باشد. دراین‌باره معیار، میزان داده‌ها است. اگر داده‌ها زیاد باشند ۵۰درصد آموزش و ۵۰درصد آزمون مناسب است. اغلب اوقات نسبت ۸۰درصد (یا ۶۶درصد) آموزش و ۲۰درصد (یا ۳۳درصد) آزمون، استفاده می‌شود. یک راه بهتر برای استفاده از این روش این است که مجموعه داده به سه قسمت آموزش، اعتبارسنجی و آزمون تقسیم‌بندی شود. بعد از آموزش الگوریتم با مجموعه

[4]. Hyper Parameters

دادهٔ آزمون، تنظیم پارامترهای الگوریتم بعد از آزمون الگوریتم با مجموعه دادهٔ اعتبارسنجی انجام می‌شود و مجدداً الگوریتم آموزش می‌بیند تا جایی که تنظیمات کاملاً انجام شود و مدل نهایی به دست آید. سپس مجموعه دادهٔ آزمون برای آزمون مدل نهایی استفاده می‌شود تا عملکرد سیستم، در برخورد با داده‌هایی که در آینده با آن‌ها مواجه می‌شود، سنجیده شود.

۶-۱-۴- روش جایگزینی مجدد[5]

در این روش از مجموعه داده‌های موجود هم به‌منظور آموزش طبقه‌بند و هم به‌منظور آزمون آن استفاده می‌شود. ناگفته پیداست که وقتی مجموعه داده‌های آموزش و آزمون یکی باشند نتایج حاصل از آن خوش‌بینانه خواهند بود.

۶-۱-۵- روش جایگشت تصادفی[6]

در این روش K بار مجموعه‌های آزمون و آموزش به نسبت مشخص به‌طور تصادفی تشکیل و از آن‌ها برای ساخت و آموزش الگوریتم استفاده می‌شود. مقدار K باید به‌گونه‌ای باشد تا اطمینان یابیم که به‌طور یکنواختی از تمام داده برای آزمون و آموزش استفاده شده است. درواقع این روش، همان روش Hold-out است که برای اطمینان بیشتر چند بار و هر بار با تقسیم‌بندی تصادفی اجرا می‌شود. سپس روی تمام نتایج میانگین‌گیری می‌شود.

۶-۲- استفاده از `Pipeline` در پایتون

همان‌طور که در بخش‌های قبل دیدیم، قبل از اعمال مدل به مجموعه دادهٔ آموزش، داده‌ها را استانداردسازی یا نرمال‌سازی می‌کنیم سپس با استفاده از روش‌های مختلف انتخاب یا استخراج ویژگی، ویژگی‌های مرتبط را به دست می‌آوریم و درنهایت مدل را با داده‌ها در فضای ویژگی جدید آموزش می‌دهیم و کارایی مدل را ارزیابی می‌کنیم. حال می‌خواهیم کلاس `Pipeline` در پایتون را به شما معرفی کنیم که با استفاده از این کلاس می‌توانیم به‌جای استفاده از عملیات `fit` و `transform`، که تاکنون می‌دیدیم، از زنجیره‌ای از عملیات استانداردسازی، استخراج ویژگی و آموزش دسته‌بند به‌صورت فشرده استفاده کنیم. نکتهٔ مهمی که دراین‌باره وجود دارد این است که هیچ محدودیتی در انجام تعداد این مراحل میانی در `pipeline` وجود ندارد.

---

5. Resubstitution
6. Randomized Permutation

✓ **مثال:**

می‌خواهیم زنجیره‌ای از عملیات استانداردسازی، PCA و خطی‌سازی لجستیک را در یک pipeline روی مجموعه دادهٔ سرطان سینه اعمال کنیم و درنهایت با استفاده از KFold اعتبارسنجی آموزش را انجام دهیم [۵]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

pipe_lr = Pipeline([('scl', StandardScaler()),
('pca', PCA(n_components=2)),
('clf',LogisticRegression(ranom_state=1))])

pipe_lr.fit(X_train, y_train)
print('Test Accuracy: %.3f' % pipe_lr.score(X_test,
y_test))
```

در خروجی داریم:

```
Test Accuracy: 0.936
```

حال می‌خواهیم با استفاده از کلاس StratifiedKFold از کتابخانهٔ sklearn نحوهٔ پیاده‌سازی اعتبارسنجی KFold را نشان دهیم [۵]. در پارامتر n_splits مقدار K را مشخص می‌کنیم. سپس در یک حلقهٔ for، با استفاده از متد split، مجموعه دادهٔ آموزش را K بار تقسیم می‌کنیم و در هر بار تقسیم، آموزش را با استفاده از pipline به مجموعه داده‌ای که در این تقسیم برای آموزش در نظر گرفته شده است، اعمال می‌کنیم. متد bincount محاسبهٔ تعداد تکرار اعداد را انجام می‌دهد و در اینجا مشخص می‌کند که در هر تقسیم انجام شده، چه تعداد برچسب 0 و چه تعداد برچسب 1 داریم:

```
import numpy as np
from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=10,random_state=1)
scores = []
```

```
for k, (train, test) in
enumerate(kfold.split(X_train,y_train)):
    pipe_lr.fit(X_train[train], y_train[train])
    score = pipe_lr.score(X_train[test],
    y_train[test])
    scores.append(score)
    print('Fold: %s, Class dist.: %s, Acc: %.3f' %
    (k+1,
    np.bincount(y_train[train]), score))
```

در خروجی داریم:

```
Fold: 1, Class dist.: [134 224], Acc: 0.975
Fold: 2, Class dist.: [134 224], Acc: 0.975
Fold: 3, Class dist.: [134 224], Acc: 0.950
Fold: 4, Class dist.: [134 224], Acc: 0.875
Fold: 5, Class dist.: [134 224], Acc: 0.950
Fold: 6, Class dist.: [134 224], Acc: 0.950
Fold: 7, Class dist.: [134 224], Acc: 0.950
Fold: 8, Class dist.: [134 224], Acc: 0.975
Fold: 9, Class dist.: [134 224], Acc: 1.000
Fold: 10, Class dist.: [135 225], Acc: 0.974
```

برای نمایش میانگین دقت‌های به دست آمده:

```
print('CV accuracy: %.3f +/- %.3f' %
(np.mean(scores),          np.std(scores)))
```

در خروجی داریم:

```
CV accuracy: 0.957 +/- 0.032
```

اگرچه مثال قبلی برای نمایش چگونگی عملکرد KFold مفید است، اما لازم است cross_val_score را نیز از کتابخانهٔ sklearn معرفی کنیم که به ما این امکان را می‌دهد تا مدلمان را با استفاده از StratifiedKFold به‌صورت کاراتری ارزیابی کنیم:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator=pipe_lr,
                         X=X_train,
                         y=y_train,
                         cv=10,
                         n_jobs=1)

print('CV accuracy scores: %s' % scores)
```

در خروجی داریم:

```
CV accuracy scores: [0.975      0.975      0.95
0.875      0.95       0.95  0.95       0.975
1.         0.97368421]
```

و برای به دست آوردن میانگین دقت داریم:

```
print('CV accuracy: %.3f +/- %.3f' %
(np.mean(scores),np.std(scores)))
```

در خروجی داریم:

```
CV accuracy: 0.957 +/- 0.032
```

نکتهٔ مهم قابل‌توجه این است که در همهٔ زبان‌های برنامه‌نویسی اعم از پایتون، در هر نسخه‌ای که از آن‌ها ارائه می‌شود، اصلاحاتی بنا به صلاحدید معماران این زبان‌ها انجام می‌شود و ممکن است در برخی از کلاس‌های کتابخانه‌های پایتون جابه‌جایی جزئی رخ دهد یا نام تعداد اندکی از پارامترها یا متدها در نسخه‌های بعدی تغییر یابد که به‌سادگی با استفاده از دستور help پایتون و جست‌وجو در اینترنت این مشکل مرتفع خواهد شد.

منابع

.1 Koza, J.R., et al., *Automated design of both the topology and sizing of analog electrical circuits using genetic programming*, in *Artificial Intelligence in Design'96*. 1996, Springer. p. 151-170.

.2 Bishop, C., *Pattern Recognition and Machine Learning, Springer55*. Chapter.

.3 Mitchell, T.M., *Machine learning*. 1997, McGraw hill.

.4 ; Available from: http://microarray.princeton.edu/oncology).

.5 Raschka, S., *Python machine learning*. 2015: Packt Publishing Ltd.

.6 Heckard. Available from: https://newonlinecourses.science.psu.edu/stat462/node/101/.

.7 Kazor, K., et al., *Comparison of linear and nonlinear dimension reduction techniques for automated process monitoring of a decentralized wastewater treatment facility.* Stochastic environmental research and risk assessment, 2016. **30**(5): p. 1527-1544.

.8 Zaitseva, E., et al. *Introduction to knowledge discovery in medical databases and use of reliability analysis in data mining*. in *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*. .2015IEEE.

.9 *machine learning*. Available from: https://www.capgemini.com/wp-content/uploads/2017/07/machinelearning_v2.png.

.10 Han, J., J. Pei, and M. Kamber, *Data mining: concepts and techniques*. 2011: Elsevier.

.11 Shah, T. 2017; Available from: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

.12 Brownlee, J. 2017; Available from: https://machinelearningmastery.com/difference-test-validation-datasets/.

.13 Kaplan, A., *The conduct of inquiry: Methodology for behavioural science*. 2017: Routledge.

.14 Klein, B.; Available from: https://www.python-course.eu.

.15 Kohl, N.; Available from: https://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks.

.16 Jarosz, Q.; Available from: https://en.wikipedia.org/wiki/Neuron.

17. منهاج، د.م.ب. (۱۳۹۱). *مبانی شبکه‌های عصبی*، تهران: دانشگاه صنعتی امیرکبیر.

.18 Rojas, R., *The backpropagation algorithm*, in *Neural networks*. 1996, Springer. p. 149-182.

.19 VanderPlas, J., *Python data science handbook: essential tools for working with data*. 2016: " O'Reilly Media, Inc.".

.20 Sheldon, R., *A first course in probability*. 2002: Pearson Education India.

.21 Drake, A.W., *Fundamentals of applied probability theory*. 1967.

.22 Papoulis, A., *Probability & statistics*. Vol. 2. 1990: Prentice-Hall Englewood Cliffs.

.23 MacKay, D.J. and D.J. Mac Kay, *Information theory, inference and learning algorithms*. 2003: Cambridge university press.

.24 Raizada, R.D. and Y.-S. Lee, *Smoothness without smoothing: why Gaussian naive Bayes is not naive for multi-subject searchlight studies.* PloS one, 2013. **8**(7): p. e69566.

.25 Esmaeili Abharian, T. and M.B. Menhaj, *Modified Convex Data Clustering Algorithm Based on Alternating Direction Method of Multipliers.* Journal of Computer & Robotics, 2015. **8**(2): p. 3341.-

.26 .-Available from: https://www.datacamp.com/courses/customer-segmentation-in-python.

.27 Ergin, E.K. 2018; Available from: http://eneskemalergin.github.io/blog//blog/Fuzzy_Clustering/.

.28 ;2018Available from: https://www.datanovia.com/en/lessons/fuzzy-clustering-essentials/fuzzy-c-means-clustering-algorithm/.

.29 Hathaway, R.J. and J.C. Bezdek, *Fuzzy c-means clustering of incomplete data.* IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2001. **31**(5): p. 735-744.

.30 ..Available from: https://pypi.org/project/fuzzy-c-means/.

.31 Dias, M.L.D., *fuzzy-c-means: An implementation of Fuzzy C-means clustering algorithm.* 2019: Federal University of Cear\'{a}, Department of Computer Science.

.32 Tahereh E.Abharian, M.B.M., *Distributed Data Clustering using Alternating Direction Method of Multipliers*, in *Faculty of Electronic, Computer & IT – Department of Computer*. 2015, Qazvin Islamic Azad University

p. 138.

.33 Reddy, C. 2018; Available from: https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec.

.34 Sayad, S.; Available from: https://www.saedsayad.com/clustering_hierarchical.htm.

.35 Jamshid Pirgazi, M.A., Tahereh Esmaeili Abharian, *An Efficient hybrid filter-wrapper metaheuristic-based gene selection method for high dimensional datasets.* Nature, 2019.

.36 ..*feature selection*. Available from: http://scikitlearn.org/stable/modules/feature_selection.html.

.37 Nazrul, S.S. 2018; Available from: https://medium.com/@sadatnazrul/the-dos-and-donts-of-principal-component-analysis-7c2e9dc8cc48.

.38 Saptashwa. 2018; Available from: https://towardsdatascience.com/dive-into-pca-principal-component-analysis-with-python-43ded13ead21.

.39 Alimoradi, M., *MEG signal processing for decoding Long-Term memory*, in *Faculty of Electronic, Computer & IT* 2014, Islamic Azad University of Qazvin

.40 Fukunaga, K., *Introduction to statistical pattern recognition*. 2013: Elsevier.

.41 Mahapatra, S. 2018; Available from: https://medium.com/journey-2-artificial-intelligence/lda-linear-discriminant-analysis-using-python-2155cf5b6398.

.42 https://www.gettyimages.com

Machine learning with Python

Tahereh Esmaeili Abharian

Mohsen Alimoradi