دانشگاه صنعتی خواجه نصیرالدین طوسی
دانشکدهٔ مهندسی برق

درس مبانی سیستم‌های هوشمند

استاد: دکتر مهدی علیاری

مینی پروژه سوم

نام و نام خانوادگی	محمدامین محمدیون شبستری
شمارهٔ دانشجویی	۴۰۱۲۲۵۰۳
نام و نام خانوادگی	محمدسبحان سخایی
شمارهٔ دانشجویی	۴۰۱۱۹۲۵۳
نام و نام خانوادگی	مبینا یوسفی‌مقدم
شمارهٔ دانشجویی	۴۰۱۲۴۰۹۳
تاریخ	دی ۱۴۰۴
لینک‌های مربوط به کد : کد | لینک گیت‌هاب	

فهرست مطالب

فهرست تصاویر

فهرست جداول

فهرست برنامه‌ها

۱ پرسش یک: مفاهیم پایه درخت تصمیم

۱.۱ توضیح دهید درخت تصمیم چیست و چرا می‌تواند ساختار مناسبی برای یادگیری داده‌ها باشد

Decision Tree یا درخت تصمیم، یکی از الگوریتم‌های پرکاربرد در یادگیری ماشین نظارت‌شده (Supervised Learning) است که برای هر دو مسئله‌ی طبقه‌بندی (Classification) و رگرسیون (Regression) به کار می‌رود. این مدل، داده‌ها را به صورت یک ساختار درختی سلسله‌مراتب مدل‌سازی می‌کند که در آن هر گره داخلی نشان‌دهنده یک آزمون روی یک ویژگی، هر شاخه نتیجه آن آزمون و هر گره برگ نشان‌دهنده برچسب نهایی یا مقدار پیش‌بینی شده است.

این ساختار به دلایل زیر برای یادگیری داده‌ها بسیار مناسب است:

- تفسیرپذیری (**Interpretability**): منطق تصمیم‌گیری آن کاملاً مشابه تفکر انسانی و به صورت قوانین If-Then است.
- مدل‌سازی روابط پیچیده: قادر است روابط غیرخطی بین ویژگی‌ها را بدون نیاز به پیچیدگی ریاضی زیاد استخراج کند.
- پردازش انواع داده‌ها: می‌تواند هم‌زمان با داده‌های عددی (Numerical) و دسته‌ای (Categorical) کار کند.

الگوریتم برای انتخاب بهترین ویژگی جهت تقسیم (Split)، از مفاهیم ریاضیاتی زیر استفاده می‌کند:

۱. آنتروپی (**Entropy**): معیاری برای اندازه‌گیری ناخالصی یا بی‌نظمی در داده‌ها که به صورت زیر محاسبه می‌شود:

$$H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i) \quad (۱)$$

۲. بهره اطلاعاتی (**Information Gain**): که تفاوت آنتروپی قبل و بعد از تقسیم را نشان می‌دهد و هدف آن بیشینه کردن این مقدار است:

$$IG(S, A) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v) \quad (۲)$$

۳. ناخالصی جینی (**Gini Impurity**): که در الگوریتم‌های پیشرفته‌ای مانند CART استفاده می‌شود:

$$\text{Gini}(S) = 1 - \sum_{i=1}^{c} p_i^2 \quad (۳)$$

۲.۱ نقش گره میانی، شاخه و برگ را در درخت تصمیم توضیح دهید

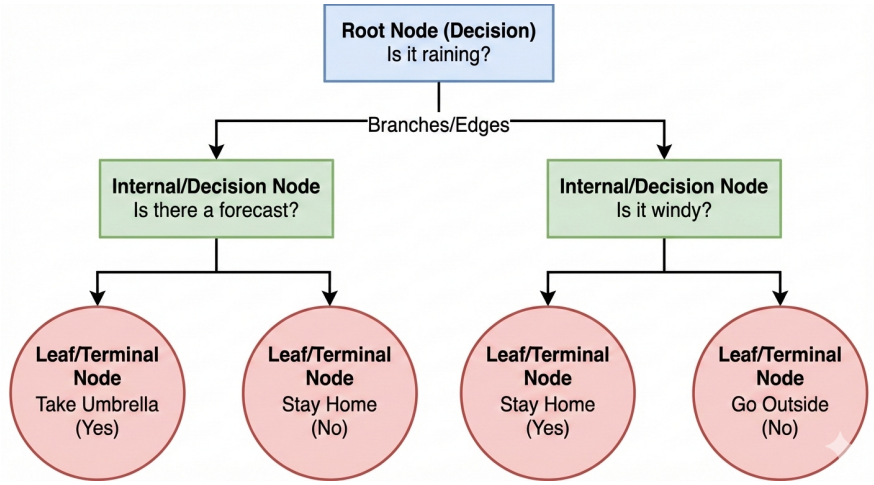
در ساختار یک Decision Tree، هر جزء نقش عملیاتی مشخصی را ایفا می‌کند:

۱. گره ریشه (**Root Node**): اولین گره در بالاترین سطح درخت است که کل مجموعه داده در آن قرار دارد. این گره بر اساس ویژگی‌ای انتخاب می‌شود که بیشترین کاهش در ناخالصی داده‌ها را ایجاد کند.

۲. گره‌های میانی یا داخلی (**Internal Nodes**): این گره‌ها نقش تصمیم‌گیرنده را دارند. هر گره میانی شامل یک سوال یا شرط روی یکی از ویژگی‌های ورودی است. با عبور داده‌ها از این گره‌ها، مجموعه داده به زیرمجموعه‌های خالص‌تر تقسیم می‌شود.

۳. شاخه‌ها (**Branches**): شاخه‌ها در واقع یال‌های متصل‌کننده گره‌ها هستند و نشان‌دهنده خروجی‌های احتمالی یک شرط (مانند بله/خیر یا دسته‌های مختلف) می‌باشند. شاخه‌ها داده‌ها را از یک سطح به سطح پایین‌تر هدایت می‌کنند.

۴. گره‌های برگ (**Leaf Nodes**): این گره‌ها نشان‌دهنده پایان مسیر تصمیم‌گیری هستند. در این گره‌ها هیچ تقسیمی صورت نمی‌گیرد و حاوی پاسخ نهایی (برچسب کلاس یا مقدار عددی) می‌باشند.



شکل ۱: نمایش ساختار اجزای درخت تصمیم: ریشه، گره میانی، شاخه و برگ

۳.۱ یک مثال واقعی از دنیای واقعی برای درخت تصمیم و بررسی ورودی‌ها و خروجی‌ها

مثال: سامانه هوشمند پیشنهاد خودرو در بازار ایران (خرید یا اجاره)

در این سناریو، یک درخت تصمیم طراحی می‌کنیم که به کاربر کمک می‌کند با توجه به نوسانات قیمت خودرو و هزینه‌های نگهداری در ایران، بهترین انتخاب را بین خرید یا اجاره‌ی خودروهای داخلی داشته باشد.

ورودی‌های مدل (ویژگی‌ها – **Features**):

- سرمایه نقدی (**Budget**): مبلغ موجود به تومان (مثلاً زیر ۳۰۰ میلیون، ۳۰۰ تا ۶۰۰ میلیون، بالای ۶۰۰ میلیون تومان).
- هدف اصلی (**Primary Purpose**): استفاده برای کار در تاکسی‌های اینترنتی، سفرهای خانوادگی، یا استفاده روزمره شهری.
- مدت زمان استفاده (**Usage Duration**): نیاز موقت (کمتر از ۱ ماه) یا نیاز دائمی و سرمایه‌گذاری.
- اولویت هزینه‌ی نگهداری (**Maintenance Priority**): چقدر هزینه استهلاک و قطعات برای کاربر اهمیت دارد (بالا یا پایین).

فرآیند و منطق تصمیم‌گیری در گره‌ها:

۱. گره ریشه (مدت زمان نیاز): آیا نیاز کاربر کمتر از یک ماه است؟

- بله: به سمت شاخه اجاره خودرو هدایت شود. (خروجی: اجاره روزانه پژو ۲۰۶ یا کوییک).
- خیر: به گره میانی بعدی (بررسی بودجه) برود.

۲. گره میانی اول (میزان بودجه برای خرید): آیا بودجه بالای ۵۰۰ میلیون تومان است؟

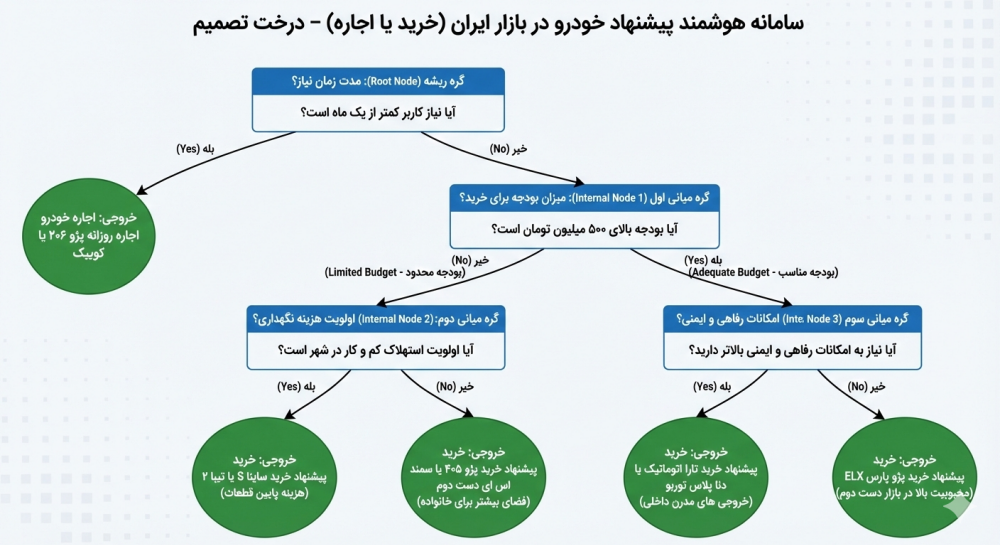
- خیر (بودجه محدود): سوال بعدی: آیا اولویت استهلاک کم و کار در شهر است؟

- بله: پیشنهاد خرید ساینا S یا تیبا ۲ (به دلیل هزینه پایین قطعات).
- خیر: پیشنهاد خرید پژو ۴۰۵ یا سمند اس‌ای دست‌دوم (فضای بیشتر برای خانواده).

- بله (بودجه مناسب): سوال بعدی: آیا نیاز به امکانات رفاهی و ایمنی بالاتر دارید؟
  - بله: پیشنهاد خرید تارا اتوماتیک یا دنا پلاس توربو (خروجی‌های مدرن داخلی).
  - خیر: پیشنهاد خرید پژو پارس **ELX** (محبوبیت بالا در بازار دست‌دوم).

خروجی‌ها (برچسب‌های نهایی در برگ‌ها):

- برگ ۱: اجاره خودروی اقتصادی برای سفرهای کوتاه‌مدت.
- برگ ۲: خرید خودروهای پلتفرم X200 (مانند کوییک و ساینا) برای کار و استفاده شهری پرفشار.
- برگ ۳: خرید خودروهای خانواده IKCO (مانند دنا و تارا) برای رفاه خانوادگی و مسافت‌های بین‌شهری.
- برگ ۴: سرمایه‌گذاری روی خودروهای با نقدشوندگی بالا (مانند پژو ۲۰۷).



شکل ۲: درخت تصمیم برای بازار خودروی ایران

۲ پرسش دو

جدول داده زیر را در نظر بگیرید که وضعیت بازی را بر اساس شرایط آب و هوایی نشان می‌دهد:

جدول ۱: داده‌های آموزشی برای مسئله پیش‌بینی بازی

Outlook	Temp	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Mild	High	Strong	No
Overcast	Cool	Normal	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Strong	Yes

۱.۲ پاسخ الف: محاسبه آنتروپی کل

مفهوم آنتروپی (**Entropy**):
آنتروپی در نظریه اطلاعات و یادگیری ماشین، معیاری برای سنجش میزان «ناخالصی» (Impurity) یا «عدم قطعیت» (Uncertainty) در یک مجموعه داده است.

- اگر آنتروپی صفر باشد، به این معنی است که نمونه‌ها کاملاً خالص هستند (یعنی همه متعلق به یک کلاس، مثلاً همه Yes یا همه No هستند). در این حالت قطعیت کامل وجود دارد.
- اگر آنتروپی یک باشد (در حالت دو کلاسه)، یعنی داده‌ها کاملاً تصادفی و با نسبت برابر (۵۰-۵۰) توزیع شده‌اند و بیشترین میزان بی‌نظمی وجود دارد.

بنابراین، هدف ما در درخت تصمیم، کاهش آنتروپی و رسیدن به گره‌هایی با آنتروپی صفر (برگ‌های خالص) است.
رابطه ریاضی آنتروپی شانون (Shannon Entropy) برای مجموعه داده S نسبت به متغیر هدف با c کلاس به صورت زیر تعریف می‌شود:

$$H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i) \tag{۴}$$

که در آن p_i احتمال (یا نسبت) وقوع کلاس iام در مجموعه داده است.
محاسبات:
در اینجا متغیر هدف ما Play است. بیایید تعداد نمونه‌ها را بشماریم:

- تعداد کل نمونه‌ها (N): ۵
- تعداد نمونه‌های Yes (مثبت): ۳
- تعداد نمونه‌های No (منفی): ۲

بنابراین احتمالات به صورت زیر هستند:

$$p_+ = P(\text{Yes}) = \frac{3}{5} = 0.6$$

$$p_- = P(\text{No}) = \frac{2}{5} = 0.4$$

حال مقادیر را در فرمول جایگذاری می‌کنیم:

$$H(S) = -\left(p_+ \log_2(p_+) + p_- \log_2(p_-)\right)$$

$$H(S) = -\left(0.6 \log_2(0.6) + 0.4 \log_2(0.4)\right)$$

با محاسبه مقادیر لگاریتم:

$$\log_2(0.6) \approx -0.737$$

$$\log_2(0.4) \approx -1.322$$

ادامه محاسبه:

$$H(S) = -\left(0.6 \times (-0.737) + 0.4 \times (-1.322)\right)$$

$$H(S) = -\left(-0.4422 - 0.5288\right)$$

$$H(S) = -(-0.971)$$

$$\boxed{H(S) \approx 0.971}$$

تحلیل نتیجه: مقدار آنتروپی به دست آمده (0.971) بسیار به عدد ۱ نزدیک است. این نشان می‌دهد که مجموعه داده اولیه دارای ناخالصی و عدم قطعیت بالایی است (توزیع کلاس‌ها تقریباً برابر است).

۲.۲ پاسخ ب: محاسبه Information Gain برای ویژگی Outlook

مفهوم بهره اطلاعات (**Information Gain**):
بهره اطلاعات (IG) معیاری است که نشان می‌دهد اگر ما داده‌ها را بر اساس یک ویژگی خاص (مانند Outlook) تقسیم کنیم، چقدر از ناخالصی (آنتروپی) کل سیستم کاسته می‌شود.

- **IG** زیاد: یعنی ویژگی مورد نظر توانسته داده‌ها را به خوبی تفکیک کند و گروه‌های خالص‌تری بسازد (ویژگی ارزشمندی است).
- **IG** کم: یعنی تقسیم‌بندی بر اساس آن ویژگی تغییر زیادی در خلوص داده‌ها ایجاد نکرده است.

رابطه ریاضی بهره اطلاعات برای ویژگی A روی مجموعه S به صورت زیر است:

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v) \qquad (۵)$$

که در آن:

- $H(S)$: آنتروپی کل (محاسبه شده در قسمت الف).
- S_v: زیرمجموعه‌ای از داده‌ها که در آن ویژگی A دارای مقدار v است.
- $H(S_v)$: آنتروپی آن زیرمجموعه.

محاسبات برای ویژگی **Outlook**:

مقادیر ممکن برای Outlook عبارتند از: Sunny، Overcast، Rain. باید آنتروپی هر دسته را جداگانه حساب کنیم.

۱. حالت **Outlook = Sunny**:

نمونه‌های مربوطه (سطر ۱ و ۲):

- سطر ۱: Play = No
- سطر ۲: Play = No

توزیع: ۰ مثبت، ۲ منفی.

$$H(Sunny) = -(\frac{0}{2}\log_2 0 + \frac{2}{2}\log_2 1) = 0$$

(چون تمام نمونه‌ها No هستند، خلوص کامل است).

۲. حالت **Outlook = Overcast**:

نمونه مربوطه (سطر ۳):

- سطر ۳: Play = Yes

توزیع: ۱ مثبت، ۰ منفی.

$$H(Overcast) = 0$$

(خلوص کامل).

۳. حالت **Outlook = Rain**:

نمونه‌های مربوطه (سطر ۴ و ۵):

- سطر ۴: Play = Yes
- سطر ۵: Play = Yes

توزیع: ۲ مثبت، ۰ منفی.

$$H(Rain) = 0$$

(خلوص کامل).

محاسبه نهایی **Information Gain**:

$$IG(S, Outlook) = H(S) - \left[\frac{2}{5}H(Sunny) + \frac{1}{5}H(Overcast) + \frac{2}{5}H(Rain)\right]$$

$$IG(S, Outlook) = 0.971 - \left[\frac{2}{5}(0) + \frac{1}{5}(0) + \frac{2}{5}(0)\right]$$

$$IG(S, Outlook) = 0.971 - 0$$

$$\boxed{IG(S, Outlook) = 0.971}$$

تحلیل نتیجه: مقدار بهره اطلاعات دقیقاً برابر با آنتروپی اولیه شد. این یعنی با دانستن ویژگی Outlook، آنتروپی باقی‌مانده به صفر می‌رسد و ما به قطعیت کامل می‌رسیم. این ویژگی بهترین ویژگی ممکن برای تفکیک این داده‌هاست.

۳.۲ پاسخ ج: تحلیل انتخاب ویژگی در الگوریتم ID3

الگوریتم ID3 (Iterative Dichotomiser 3) یک الگوریتم حریصانه (Greedy) برای ساخت درخت تصمیم است. این الگوریتم در هر گره از درخت، ویژگی‌ای را برای انشعاب انتخاب می‌کند که بیشترین Information Gain را داشته باشد. دلایل و تحلیل این انتخاب به شرح زیر است:

۱. ماکسیمم کردن کاهش عدم قطعیت: همانطور که در فرمول مشاهده کردیم، IG برابر است با اختلاف آنتروپی قبل از انشعاب و میانگین وزنی آنتروپی بعد از انشعاب. از آنجا که آنتروپی قبل از انشعاب ($H(S)$) ثابت است، ماکسیمم کردن IG معادل است با مینیمم کردن آنتروپی فرزندان.

$$\max(IG) \equiv \min(\text{آنتروپی باقی‌مانده})$$

انتخاب ویژگی با بیشترین IG تضمین می‌کند که زیرمجموعه‌های ایجاد شده (فرزندان)، خالص‌ترین (همگن‌ترین) حالت ممکن را داشته باشند.

۲. تولید درخت‌های کوچکتر : با انتخاب ویژگی‌ای که بیشترین اطلاعات را فراهم می‌کند، ما سریع‌تر به گره‌های خالص (برگ‌ها) می‌رسیم. این کار باعث می‌شود عمق درخت کاهش یابد. طبق اصل تیغ اوکام (Occam's Razor)، مدل‌های ساده‌تر (درخت‌های کوتاه‌تر) معمولاً تعمیم‌پذیری (Generalization) بهتری دارند و کمتر دچار بیش‌برازش (Overfitting) می‌شوند.

۳. بهبود تفکیک کلاس‌ها: در مثال بالا دیدیم که IG برای Outlook ماکسیمم مقدار ممکن بود. این انتخاب باعث شد تمام زیرشاخه‌ها (Sunny, Overcast, Rain) بلافاصله به برگ‌های خالص (Yes یا No) منتهی شوند و کلاس‌ها کاملاً از هم تفکیک شوند. اگر ویژگی دیگری با IG کمتر انتخاب می‌شد، احتمالاً نیاز به انشعابات بعدی و پیچیده‌تر شدن درخت بود.

بسیار عالی. در ادامه کد LaTeX مربوط به پرسش سوم را با جزئیات کامل، تحلیل‌های فنی و رعایت دقیق دستورالعمل‌های نگارشی شما (استفاده از (آماده کرده‌ام.

در این بخش تلاش شده است تا دلایل ریاضی و منطقی پشت هر مزیت یا عیب (مانند مرزهای تصمیم متعامد یا واریانس بالا) تشریح شود.

کد زیر را به فایل خود اضافه کنید:

snippet Code

۳ پرسش سه: مزایا و معایب درخت تصمیم

درخت تصمیم (Decision Tree) یکی از محبوب‌ترین الگوریتم‌های یادگیری ماشین است، اما مانند هر روش دیگری، نقاط قوت و ضعف خاص خود را دارد که ناشی از ساختار سلسله‌مرتبی و روش یادگیری حریصانه (Greedy) آن است.

۱.۳ بخش الف: مشکلات و معایب

درخت‌های تصمیم، به ویژه زمانی که محدود نشوند، مستعد خطاها و رفتارهای خاصی هستند. چهار مورد از مهم‌ترین معایب عبارتند از:

۱. بیش‌برازش (**Overfitting**):
یکی از شایع‌ترین مشکلات درخت تصمیم، تمایل شدید آن به بیش‌برازش است. اگر شرط توقف مناسبی (مانند حداکثر عمق درخت) تعیین نشود، الگوریتم سعی می‌کند تک‌تک نمونه‌های آموزشی را با ایجاد قوانین بسیار خاص و پیچیده دسته‌بندی کند.

- تحلیل: درخت ممکن است آنقدر رشد کند که هر برگ تنها شامل یک نمونه باشد. در این حالت، درخت نویزها (Noise) و داده‌های پرت (Outliers) موجود در داده‌های آموزشی را به عنوان الگو یاد می‌گیرد. این مسئله باعث می‌شود واریانس مدل بالا برود و روی داده‌های تست عملکرد ضعیفی داشته باشد.
- راهکار: استفاده از تکنیک‌های هرس کردن (Pruning) یا محدود کردن عمق درخت.

۲. ناپایداری (**Instability**):
درخت‌های تصمیم نسبت به تغییرات کوچک در داده‌های آموزشی بسیار حساس هستند. تغییری اندک در داده‌ها می‌تواند منجر به ایجاد درختی کاملاً متفاوت شود.

- دلیل: این ناپایداری به دلیل ساختار سلسله‌مرتبی درخت است. اگر یک تغییر کوچک در داده‌ها باعث شود که در ریشه درخت، ویژگی متفاوتی بیشترین Information Gain را کسب کند، کل ساختار پایین‌دست درخت تغییر خواهد کرد. این ویژگی نشان‌دهنده «واریانس بالا» (High Variance) در این مدل‌هاست.

۳. ناتوانی در مدل‌سازی روابط خطی مورب (مرزهای تصمیم متعامد):
درخت‌های تصمیم استاندارد، فضا را با خطوط (یا ابرصفحات) موازی با محورهای مختصات تقسیم می‌کنند.

- تحلیل: اگر مرز تصمیم واقعی بین دو کلاس به صورت مورب باشد (مثلاً رابطه $y = x$)، درخت تصمیم برای تقریب زدن این خط صاف، مجبور است یک ساختار پله‌ای (Staircase) بسیار پیچیده با تعداد زیادی انشعاب ایجاد کند. این در حالی است که یک مدل ساده مانند رگرسیون لجستیک (Logistic Regression) می‌تواند این مرز را با یک خط ساده مدل کند.

۴. سوگیری به سمت ویژگی‌های دارای سطوح زیاد (**Bias towards attributes with many levels**):
در معیارهایی مانند بهره اطلاعات (Information Gain)، درخت تصمیم تمایل دارد ویژگی‌هایی را انتخاب کند که دارای مقادیر منحصربه‌فرد زیادی هستند.

- مثال: اگر ویژگی‌ای مانند «کد ملی» یا «شماره ردیف» در داده‌ها باشد، این ویژگی می‌تواند داده‌ها را کاملاً خالص تفکیک کند (چون هر کد ملی یکتاست)، اما این تفکیک هیچ ارزش تعمیم‌پذیری ندارد. اگرچه معیار Gain Ratio برای حل این مشکل معرفی شده است، اما در حالت پایه این یک ضعف ذاتی محسوب می‌شود.

۲.۳ بخش ب: مزایا و نقاط قوت

با وجود معایب ذکر شده، درخت تصمیم به دلیل ویژگی‌های منحصربه‌فردش همچنان بسیار پرکاربرد است:

۱. تفسیرپذیری و شفافیت بالا (**Interpretability**):
درخت تصمیم یک مدل «جعبه سفید» (White-box) محسوب می‌شود. برخلاف مدل‌هایی مانند شبکه‌های عصبی (Neural Networks) که فرآیند تصمیم‌گیری در آن‌ها پنهان است، منطق درخت تصمیم کاملاً قابل مشاهده است.

- تحلیل: مسیر تصمیم‌گیری از ریشه تا برگ را می‌توان به صورت مجموعه‌ای از قوانین «اگر-آنگاه» (If-Then Rules) بیان کرد. این ویژگی در حوزه‌هایی مانند پزشکی یا بانکداری که نیاز به توجیه دلیل تصمیم وجود دارد (مثلاً چرا وام رد شد؟) حیاتی است.

۲. عدم نیاز به پیش‌پردازش سنگین داده‌ها:
بسیاری از الگوریتم‌های یادگیری ماشین (مانند KNN یا SVM) نیاز مبرم به نرمال‌سازی (Normalization) یا استانداردسازی داده‌ها دارند، زیرا بر اساس فاصله اقلیدسی کار می‌کنند.

- استدلال: درخت تصمیم به مقیاس داده‌ها حساس نیست. چه بازه یک ویژگی بین ۰ تا ۱ باشد و چه بین ۰ تا ۱۰۰۰، نقطه برش (Split Point) صرفاً بر اساس مرتب‌سازی داده‌ها انتخاب می‌شود و تغییر مقیاس تأثیری در ساختار درخت ندارد. همچنین این مدل به خوبی با داده‌های گم‌شده (Missing Values) کنار می‌آید.

۳. پشتیبانی همزمان از داده‌های عددی و طبقه‌ای (**Numerical and Categorical Data**):
درخت تصمیم می‌تواند به‌طور طبیعی با انواع مختلف داده‌ها کار کند.

- تحلیل: برای داده‌های عددی، الگوریتم به دنبال بهترین آستانه عددی (مثلاً $Temperature > 25$) می‌گردد و برای داده‌های طبقه‌ای (مانند $Color = Red$) زیرمجموعه‌ها را ایجاد می‌کند. این در حالی است که بسیاری از روش‌های آماری تنها روی داده‌های عددی کار می‌کنند و نیاز به تبدیل داده‌های طبقه‌ای به بردار (One-Hot Encoding) دارند.

۴. انجام ضمنی انتخاب ویژگی (**Implicit Feature Selection**):
درخت تصمیم به صورت خودکار مهم‌ترین ویژگی‌ها را شناسایی می‌کند.

- استدلال: ویژگی‌هایی که در سطوح بالای درخت (نزدیک به ریشه) ظاهر می‌شوند، آنهایی هستند که بیشترین اطلاعات را دارند و قدرت تفکیک‌کنندگی بالایی دارند. ویژگی‌های بی‌اهمیت یا کم‌اهمیت ممکن است اصلاً در درخت ظاهر نشوند یا در عمق‌های بسیار پایین قرار گیرند. بنابراین، ساخت مدل خود به نوعی فرآیند انتخاب ویژگی‌های موثر را انجام می‌دهد.

۴ پرسش چهار: هرس کردن درخت تصمیم

هرس کردن (Pruning) تکنیکی حیاتی در درخت‌های تصمیم است که با کاهش اندازه درخت و حذف بخش‌های غیرضروری (که احتمالا نویز یا داده‌های پرت را مدل کرده‌اند)، از بیش‌برازش (Overfitting) جلوگیری کرده و قدرت تعمیم‌پذیری مدل را افزایش می‌دهد.

۱.۴ پاسخ الف: تعاریف و تفاوت‌های هرس پیش‌گیرانه و پس از ساخت

۱. هرس پیش‌گیرانه (**Pre-pruning** یا **Early Stopping**):
در این روش، فرآیند رشد درخت در حین ساخت متوقف می‌شود. قبل از هر انشعاب، الگوریتم بررسی می‌کند که آیا این انشعاب معیارهای خاصی را ارضا می‌کند یا خیر. اگر شرط برقرار نباشد، گره به برگ تبدیل می‌شود.
معیارهای رایج برای توقف (ریاضیات و شروط):

- آستانه بهره اطلاعات: اگر $IG(S, A) < \delta$ باشد (که δ یک حد آستانه کوچک است)، انشعاب انجام نمی‌شود.

- محدودیت عمق: اگر $Depth(Tree) = Max\_Depth$ باشد، رشد متوقف می‌شود.
- حداقل نمونه در گره: اگر تعداد نمونه‌ها در گره $N < Min\_Samples$ باشد، انشعاب ممنوع است.
- معیار آماری: استفاده از آزمون Chi-Square (χ^2) برای بررسی استقلال آماری توزیع کلاس‌ها قبل و بعد از انشعاب. اگر p-value از حد مشخصی بیشتر باشد، انشعاب رد می‌شود.

۲. هرس پس از ساخت (**Post-pruning**):

در این روش، ابتدا درخت به‌طور کامل رشد می‌کند (تا زمانی که تمام برگ‌ها خالص شوند یا به محدودیت‌های سخت‌افزاری برسیم). سپس، الگوریتم از پایین به بالا (از برگ‌ها به سمت ریشه) حرکت کرده و زیردرخت‌ها (Subtrees) را بررسی می‌کند. اگر حذف یک زیردرخت و جایگزینی آن با یک برگ ساده باعث افزایش خطای مدل روی داده‌های اعتبارسنجی (Validation Set) نشود (یا بهبود تابع هزینه را در پی داشته باشد)، آن زیردرخت هرس می‌شود.

ریاضیات Cost-Complexity Pruning (یکی از روش‌های محبوب Post-pruning):

در این روش، هدف مینیمم کردن تابع هزینه زیر است:

$$R_\alpha(T) = R(T) + \alpha|T| \tag{۶}$$

که در آن:

- $R(T)$: خطای آموزشی درخت T (مثلاً Misclassification Rate).
- $|T|$: تعداد برگ‌های درخت (معیاری از پیچیدگی مدل).
- $\alpha \geq 0$: پارامتر تنظیم‌کننده (Regularization Parameter) که جریمه پیچیدگی را کنترل می‌کند.

در این فرآیند، زیردرخت‌هایی که کاهش خطای ناچیزی دارند اما تعداد برگ‌ها را زیاد می‌کنند (پیچیدگی بالا)، حذف می‌شوند.

تفاوت‌های اصلی:

جدول ۲: مقایسه هرس پیش‌گیرانه و پس از ساخت

ویژگی	Pre-pruning	Post-pruning
سرعت	سریع (ساخت درخت زود متوقف می‌شود)	کند (ابتدا کل درخت ساخته می‌شود)
پیچیدگی محاسباتی	پایین	بالا
ریسک	خطر Underfitting	خطر کمتر، معمولاً دقیق‌تر
رویکرد	حریصانه و محلی	سراسری‌تر

۲.۴ پاسخ ب: مثال مفهومی و اثر افق (**Horizon Effect**)

برای نشان دادن برتری Post-pruning، مثالی را بررسی می‌کنیم که در آن ویژگی‌ها به تنهایی بی‌معنی هستند اما تعامل آن‌ها با هم (Interaction) ارزشمند است. این پدیده در دنیای واقعی مشابه مسئله XOR است.

سناریوی پزشکی: فرض کنید می‌خواهیم اثر دو داروی A و B را بر روی مسمومیت یک بیمار بررسی کنیم.

- اگر بیمار هیچ‌کدام را مصرف نکند: سالم است (Class = Healthy).

- اگر بیمار هر دو را با هم مصرف کند: داروها اثر هم را خنثی کرده و سالم می‌ماند (Class = Healthy).
- اگر فقط یکی از داروها را مصرف کند: دچار مسمومیت می‌شود (Class = Toxic).

جدول داده‌ها (۴ نمونه):

Status (Label)	Drug B	Drug A
Healthy	۰	۰
Toxic	۱	۰
Toxic	۰	۱
Healthy	۱	۱

تحلیل ۱: عملکرد هرس پیش‌گیرانه **(Pre-pruning)**
الگوریتم در ریشه درخت می‌خواهد بهترین ویژگی را انتخاب کند.

- آنتروپی کل سیستم: (۲ تا Healthy، ۲ تا Toxic) $\rightarrow H(S) = 1$.
- اگر بر اساس Drug A تقسیم کنیم:
  - شاخه $A = 0$: شامل یک Healthy (۰،۰) و یک Toxic (۰،۱). آنتروپی = ۱.
  - شاخه $A = 1$: شامل یک Toxic (۱،۰) و یک Healthy (۱،۱). آنتروپی = ۱.
- محاسبه بهره اطلاعات:

$$IG(S, A) = 1 - (0.5 \times 1 + 0.5 \times 1) = 0$$

برای ویژگی Drug B نیز همین اتفاق می‌افتد و $IG = 0$ خواهد بود.
نتیجه: از آنجا که بهره اطلاعات صفر است (یا کمتر از آستانه δ)، هرس پیش‌گیرانه تصمیم می‌گیرد که هیچ‌کدام از این ویژگی‌ها مفید نیستند. رشد درخت متوقف شده و یک برگ با رأی اکثریت (یا تصادفی) ایجاد می‌شود. مدل دچار Underfitting شدید می‌شود و هیچ الگویی یاد نمی‌گیرد. این پدیده اثر افق **(Horizon Effect)** نام دارد؛ یعنی الگوریتم نمی‌تواند ببیند که شاید یک انشعاب "بد" اکنون، منجر به انشعابات "عالی" در آینده شود.

تحلیل ۲: عملکرد هرس پس از ساخت **(Post-pruning)**
در این روش، درخت کامل ساخته می‌شود:

۱. در ریشه، با وجود $IG = 0$، الگوریتم (به دلیل عدم وجود شرط توقف سخت‌گیرانه) مثلاً Drug A را انتخاب می‌کند.
۲. در عمق بعدی، برای شاخه $A = 0$، ویژگی Drug B بررسی می‌شود. حالا اگر $B = 0$ باشد کلاس Healthy و اگر $B = 1$ باشد کلاس Toxic است. خلوص کامل حاصل می‌شود ($IG = 1$).
۳. همین اتفاق برای شاخه $A = 1$ می‌افتد.

نتیجه: درخت کامل می‌تواند داده‌ها را با خطای صفر دسته‌بندی کند. در مرحله هرس، وقتی الگوریتم بررسی می‌کند که آیا حذف زیرشاخه‌ها خطا را کم می‌کند یا نه، متوجه می‌شود که حذف آن‌ها خطا را به شدت زیاد می‌کند (از ۰ به ۵۰ درصد). بنابراین ساختار حفظ می‌شود.

استدلال نهایی: هرس پس از ساخت به مدل اجازه می‌دهد تعاملات پیچیده بین ویژگی‌ها (مانند XOR) را کشف کند که در نگاه اول (حریصانه) بی‌ارزش به نظر می‌رسند، اما در ترکیب با هم قدرت پیش‌بینی بالایی دارند.

۵ پرسش پنج: پیاده‌سازی درخت تصمیم روی دادگان **Titanic**(بخش اول)

ابتدا دادهٔ Titanic – Machine Learning from Disaster را از سایت kaggle دانلود نمودیم و بر روی محیط کولب آوردیم سپس فایل train.csv آن را با کتابخانهٔ pandas باز نمودیم. کد زیر بدین منظور زده شده است:

```
import pandas as pd
df = pd.read_csv("train.csv")
df
```

Code ۱: بالا آوردن دیتاست روی محیط کولب و نمایش آن

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

شکل ۳: نمایش دیتاست

حال تعداد مقادیر گمشدهٔ مربوط به هرستون را محاسبه می‌نماییم.

```
df.isnull().sum()
```

Code ۲: محاسبهٔ تعداد مقادیر گمشده

PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

شکل ۴: مقادیر گمشده

ستون Embarked با استفاده از مد (Mode) پر شده است. دلیل این انتخاب آن است که کمتر از ۱% داده‌ها دارای مقدار گمشده هستند (تنها ۲ نمونه از ۸۹۱ داده). در چنین شرایطی، جایگزین کردن مقادیر گمشده با پرتکرارترین مقدار منطقی است، زیرا باعث حفظ توزیع کلی داده‌ها می‌شود و تأثیر معناداری بر نتایج مدل نخواهد داشت.

ستون Age با استفاده از میانه گروه‌بندی‌شده (Grouped Median) پر شده است. از آن‌جایی که توزیع سن معمولاً چوله (Skewed) است، استفاده از میانگین می‌تواند تحت تأثیر مقادیر پرت قرار گیرد، در حالی که میانه نسبت به این مقادیر مقاوم‌تر است.

همچنین، پر کردن مقادیر گمشده سن به صورت گروه‌بندی‌شده بر اساس ویژگی‌های Pclass و Sex انجام شده است، زیرا بین این ویژگی‌ها رابطه معناداری وجود دارد. برای مثال، مسافرانی که در کلاس اول (Pclass = 1) سفر می‌کردند، معمولاً از وضعیت اقتصادی بهتری برخوردار بوده و به طور متوسط سن بالاتری نسبت به مسافران کلاس سوم داشته‌اند.

این رویکرد باعث می‌شود که روابط بین ویژگی‌ها حفظ شود و ساختار واقعی داده‌ها بهتر بازنمایی گردد؛ به عنوان نمونه، ارتباط میان سن بالاتر، وضعیت اقتصادی بهتر و کلاس مسافرتی بالاتر در داده‌ها باقی می‌ماند.

```
mode_embarked  = df['Embarked'].mode()[0]
df['Embarked'] = df['Embarked'].fillna(mode_embarked)
group_ = df.groupby(['Pclass', 'Sex'])['Age'].transform('median')
df['Age'] = df['Age'].fillna(group_)
```

Code ۳: جایگذاری مقادیر گمشده

حال بار دیگر تعداد مقادیر گمشده را حساب نمودیم تا مطمئن شویم که این مقادیر گمشده پرشده‌اند. از آنجاییکه در ادامه ستون cabin را حذف می‌نماییم، مقادیر گمشدهٔ آن را پر نکردیم.

ستون‌های ticket، name و cabin را حذف نمودیم. نمایش دیتاست به صورت زیر می‌شود.

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...	...	...
886	887	0	2	male	27.0	0	0	13.0000	S
887	888	1	1	female	19.0	0	0	30.0000	S
888	889	0	3	female	21.5	1	2	23.4500	S
889	890	1	1	male	26.0	0	0	30.0000	C
890	891	0	3	male	32.0	0	0	7.7500	Q

891 rows × 9 columns

شکل ۵: نمایش دیتاست بعد از حذف سه ستون از ویژگی‌ها

در ادامه از تکنیک one hot Encoding استفاده می‌نماییم. در این روش از کد کردن، به ازای حالت‌های مختلف در هر ستون از ویژگی، n-1 ستون می‌سازیم و از آن استفاده می‌نماییم. مقادیر این ستون‌ها به صورت صفر و یک می‌باشند. مثلا در جنسیت یک ستون

ساخته می‌شود که شامل مقادیر صفر و یک می‌باشد.

نمایش دیتاست بعد از one hot Encoding:

```
df['Sex'] = df['Sex'].map({'female': 0, 'male': 1})
df['Sex']
```

Code ۴: one hot Encoding

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_Q	Embarked_S
0	1	0	3	1	22.0	1	0	7.2500	0	1
1	2	1	1	0	38.0	1	0	71.2833	0	0
2	3	1	3	0	26.0	0	0	7.9250	0	1
3	4	1	1	0	35.0	1	0	53.1000	0	1
4	5	0	3	1	35.0	0	0	8.0500	0	1
...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	1	27.0	0	0	13.0000	0	1
887	888	1	1	0	19.0	0	0	30.0000	0	1
888	889	0	3	0	21.5	1	2	23.4500	0	1
889	890	1	1	1	26.0	0	0	30.0000	0	0
890	891	0	3	1	32.0	0	0	7.7500	1	0

891 rows × 10 columns

شکل ۶: نمایش دیتاست بعد از one hot Encoding

حال ستون survived را به عنوان متغیر هدف در نظر می‌گیریم و از ستون‌های ویژگی این متغیر و passengerId را حذف می‌نماییم.

حال داده‌ها را با نسبت ۲۰ به ۸۰ به دو قسمت آموزش و آزمون تقسیم می‌کنیم.

```
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

Code ۵: تقسیم داده به دو بخش آموزش و آزمون

حال یک درخت تصمیم با عمق ۴ آموزش می‌دهیم:

```
from sklearn.tree import DecisionTreeClassifier

decisiontree_model = DecisionTreeClassifier(max_depth=4, random_state=42)
decisiontree_model
decisiontree_model.fit(X_train, y_train)
```

Code ۶: آموزش درخت تصمیم با عمق ۴

حال دقت را روی درخت آموزش دیده برای داده‌های آموزش و آزمون محاسبه می‌نماییم:

```
y_hat = decisiontree_model.predict(X_test)
y_hat_train = decisiontree_model.predict(X_train)
from sklearn.metrics import classification_report, accuracy_score
```

```
۴
۵  acc = accuracy_score(y_test, y_hat)
۶  print(f'Acc on test : {acc}\n')
۷  print(f'Acc on train : {accuracy_score(y_train, y_hat_train)}\n')
۸  print('Classification Report: \n')
۹  print(classification_report(y_test, y_hat))
```

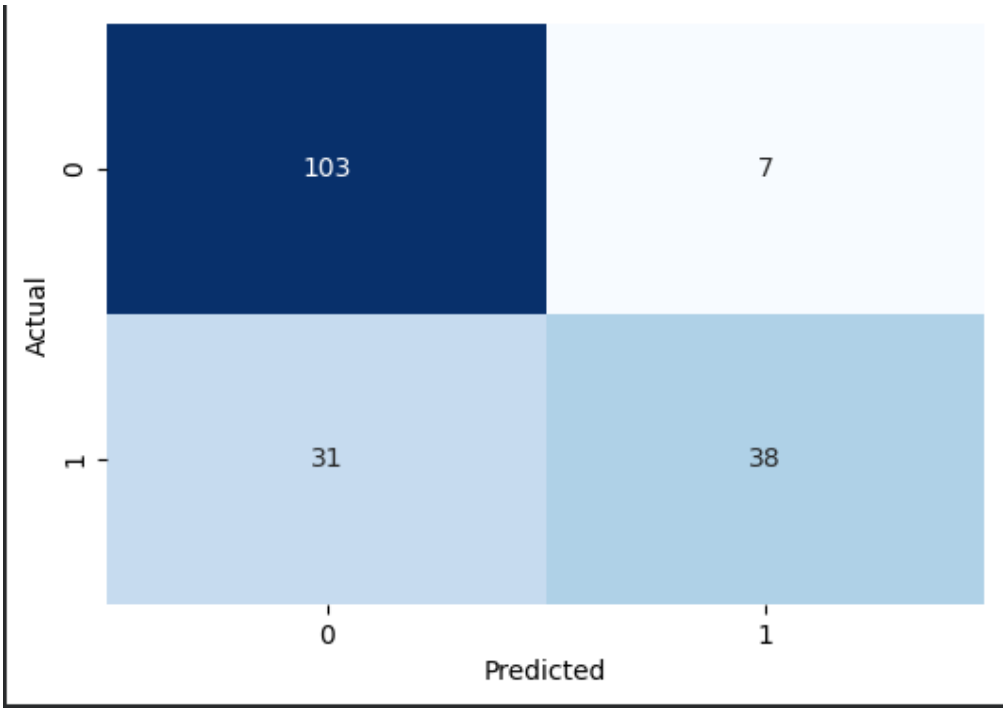
Code ۷: محاسبهٔ دقت برای داده‌های آموزش و آزمون

$$\text{Train Accuracy} = 0.8413 \quad , \quad \text{Test Accuracy} = 0.7877$$

یک کلسیفیکیشن ریپورت نیز گرفتیم که نتایج آن در فایل پایتون زده شده قابل مشاهده است.

```
۱  from sklearn.metrics import  confusion_matrix
۲  import seaborn as sns
۳  import matplotlib.pyplot as plt
۴  cm = confusion_matrix(y_test, y_hat)
۵  plt.figure(figsize=(6,4))
۶  sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
۷  plt.xlabel('Predicted')
۸  plt.ylabel('Actual')
۹  plt.show()
```

Code ۸: ترسیم کانفیوژن ماتریس

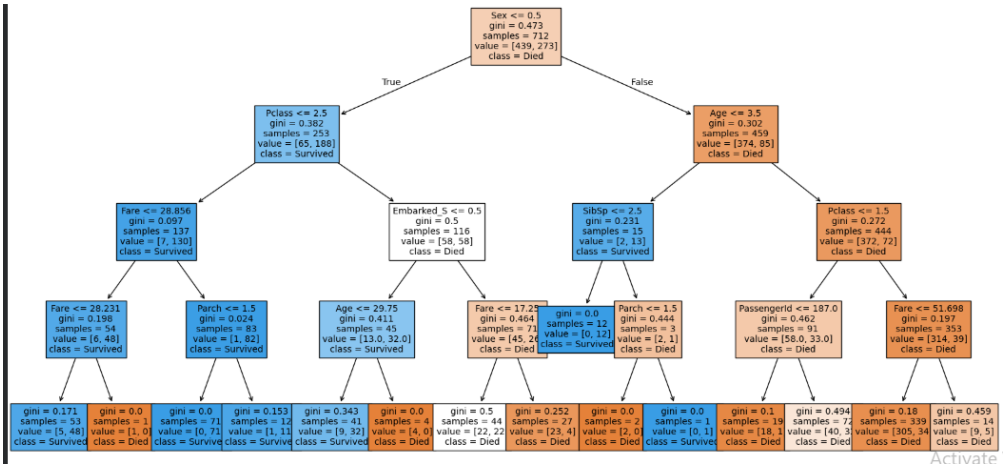


شکل ۷: کانفیوژن ماتریس

درخت تصمیم مطابق شکل زیر است:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(20,10))
plot_tree(decisiontree_model, feature_names=X.columns, class_names = ['Died', 'Survived'],
    filled = True, fontsize=10)
plt.show()
```

Code ۹: ترسیم کانفیوژن ماتریس



شکل ۸: درخت تصمیم

حال سوال از ما خواسته تا تعداد گره و عمق درخت را نیز محاسبه کنیم که به صورت زیر پیاده‌سازی شده است:

```
print(f'Depth : {decisiontree_model.get_depth()}')
print(f'Leaves : {decisiontree_model.get_n_leaves()}')
print(f'Nodes : {decisiontree_model.tree_.node_count}')
```

Code ۱۰: محاسبهٔ تعداد گره و عمق درخت

$$\text{Depth} = 4 \quad , \quad \text{Number of Leaves} = 15 \quad , \quad \text{Number of Nodes} = 29$$

حال به جای اینکه از کتابخونه استفاده نماییم، از کد from scratch بهره می‌بریم.

```
import numpy as np

class Node:
    # this class is defined for nodes
    def __init__(self, feature_index = None, threshold = None, left = None, right = None, value =
        None):
        self.feature_index = feature_index  # Which column to split on
        self.threshold = threshold          # Value to split at (e.g., Age < 10)
        self.left = left                    # Left child Node
```

```
        self.right = right                  # Right child Node


        self.value = value

class My_DecisionTreeClassifier:
    def __init__(self, max_depth = None):
        self.max_depth = max_depth
        self.root = None

    def fit(self, X, y):

        X = np.array(X)
        y = np.array(y)
        self.root = self.grow_tree(X, y)

    def grow_tree(self, X, y, depth = 0):
        n_sample, n_feature = X.shape
        n_label = len(np.unique(y))

        # alg stop mishe if:
        # 1 - max depth reside bashe
        # 2 - purity 100 darsad beshe
        # 3- hame nemone ha barresi beshe va hich sample baghi namone
        if(self.max_depth is not None and depth >= self.max_depth) or n_label == 1 or n_sample < 2:
            leaf_node = self.most_common(y)
            return Node(value=leaf_node)
        # best_feature: Which input column should we split on at this node?
        # best_thresh : At what value of that feature should we split?
        best_feature, best_thresh = self.best_split(X, y, n_feature)

        if best_feature is not None:
            left_idx = X [:, best_feature] < best_thresh
            right_idx = X [:, best_feature] >= best_thresh


            left_child = self.grow_tree(X[left_idx, :], y[left_idx], depth + 1)
            right_child = self.grow_tree(X[right_idx, :], y[right_idx], depth + 1)

            return Node(feature_index=best_feature, threshold=best_thresh, left=left_child, right=right_child)
        return Node(value=self.most_common(y))

    def best_split(self, X, y, n_features):
```

```
    # loop mikone hame features ha ro ta feature ba minimized gini peyda kone
    best_gain = -1
    split_idx, split_thresh = None, None
    parent_gini = self.gini(y)
    for f_index in range (n_features):
        thresholds = np.unique(X[:, f_index])
        for thresh in thresholds:
            gain = self.information_gain(y,X[:, f_index], thresh, parent_gini)

            if gain > best_gain:
                best_gain = gain
                split_idx = f_index
                split_thresh = thresh
    return split_idx, split_thresh

def information_gain(self, y, X_col, threshold, parent_gini):
    left_ = X_col < threshold
    right_ = ~left_

    if len(y[left_]) == 0 or len(y[right_]) == 0:
        return 0

    n = len(y)
    n_left , n_right = len(y[left_]), len(y[right_])
    gini_left, gini_right = self.gini(y[left_]), self.gini(y[right_])
    child_gini = (n_left/n) *gini_left + (n_right/n) *gini_right
    return parent_gini - child_gini

def gini(self, y):
    prob = [np.sum(y == c)/ len(y) for c in np.unique(y)]
    return 1 - np.sum(np.array(prob)**2)

def most_common(self, y):
    if len(y) == 0:
        return 0
    return np.bincount(y).argmax()

def predict (self, x):
    x = np.array(x)
    return np.array([self.traverse_tree(x_, self.root) for x_ in x])

def traverse_tree(self, x, node):
    if node.value is not None:
        return node.value

```

```
        if x[node.feature_index] < node.threshold:
            return self.traverse_tree(x, node.left)
        return self.traverse_tree(x, node.right)
```

Code ۱۱: نوشتن مدل درخت تصمیم به صورت from scratch

حال داده‌های آموزشی را به مدل می‌دهیم و مدل را آموزش می‌دهیم و سپس روی داده‌های تست معیارهای ارزیابی را محاسبه می‌نماییم.

```
my_model = My_DecisionTreeClassifier(max_depth=4)
my_model.fit(X_train, y_train)
my_model
y_pred_scratch = my_model.predict(X_test)
y_pred_scratch_train = my_model.predict(X_train)
from sklearn.metrics import classification_report, accuracy_score

acc = accuracy_score(y_test, y_pred_scratch)
print(f'Acc on test : {acc}\n')
print(f'Acc on train : {accuracy_score(y_train, y_pred_scratch_train)}\n')
print('Classification Report: \n')
print(classification_report(y_test, y_pred_scratch))
```

Code ۱۲: آموزش مدل و ارزیابی آن

$$\text{Train Accuracy} = 0.8413 \quad , \quad \text{Test Accuracy} = 0.7765$$

کلسیفیکیشن ریپورت را نیز به دست آوردیم که در فایل پایتون قابل مشاهده است.

نتایج مدل پیاده‌سازی‌شده تقریباً ۱% دقت کمتری نسبت به مدل پیاده‌سازی‌شده در کتابخانه `scikit-learn` نشان می‌دهد. این اختلاف جزئی بیانگر آن است که منطق مدل پیاده‌سازی‌شده صحیح است و عملکرد آن با مدل مرجع هم‌راستا می‌باشد.

اختلاف حدود ۱% می‌تواند ناشی از نویزهای مربوط به پیاده‌سازی و همچنین بهینه‌سازی‌های داخلی گسترده‌ای باشد که در کتابخانه `scikit-learn` برای آموزش مدل‌ها انجام شده است. به طور کلی، نزدیکی نتایج به مدل مرجع نشان می‌دهد که پیاده‌سازی انجام‌شده از نظر مفهومی و الگوریتمی معتبر است.

۶ پرسش شش: پیاده‌سازی درخت تصمیم روی دادگان **Titanic** (بخش دوم: عمق و هرس)

۱.۶ آموزش درخت کامل(بدون محدودیت عمق)

حال بدون محدودیت عمق مدل را آموزش می‌دهیم.

```
decisiontree_model = DecisionTreeClassifier(max_depth=None, random_state=42)
decisiontree_model.fit(X_train,y_train)
decisiontree_model
```

Code ۱۳: آموزش مدل

مدل آموزش دیده را با کد زیر ارزیابی می‌کنیم.

```
y_hat = decisiontree_model.predict(X_test)
y_hat_train = decisiontree_model.predict(X_train)

acc = accuracy_score(y_test, y_hat)
print(f'Acc on test : {acc}\n')
print(f'Acc on train : {accuracy_score(y_train, y_hat_train)}\n')

print(classification_report(y_test,y_hat))
```

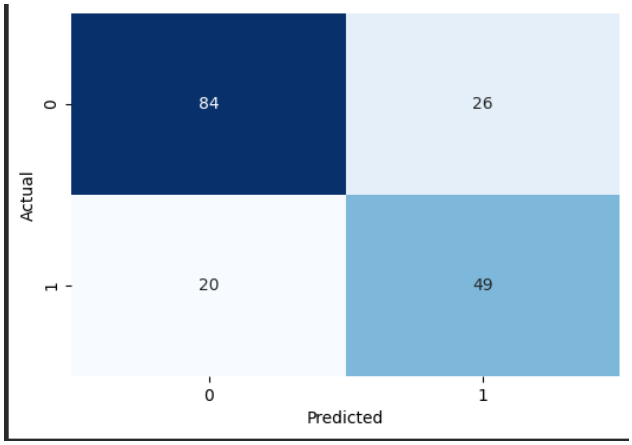
Code ۱۴: ارزیابی مدل آموزش دیده

$$\text{Train Accuracy} = 1.0000 \quad , \quad \text{Test Accuracy} = 0.7430$$

کلسیفیکیشن ریپورت نیز بر روی داده‌ها گرفته شده است که در فایل پایتون قابل مشاهده است.
کانفیوژن ماتریس بر روی داده‌های تست به صورت زیر است:

```
from sklearn.metrics import  confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, y_hat)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Code ۱۵: کد محاسبهٔ کانفیوژن ماتریس



شکل ۹: کانفیوژن ماتریس

```
from sklearn.tree import plot_tree
```

```
plt.figure(figsize=(30,30))
plot_tree(decisiontree_model, feature_names=X.columns, class_names = ['Died', 'Survived'],
filled = True, fontsize=10)
plt.show()
```

Code ۱۶: کد ترسیم درخت تصمیم

این درخت تصمیم به دلیل بزرگ بودن قابل نمایش در اینجا نبود. در فایل پایتون قابل مشاهده است. تعداد گره‌ها، عمق درخت و تعداد شاخه‌ها در کد زیر محاسبه شده است:

```
from sklearn.tree import plot_tree
plt.figure(figsize=(30,30))
plot_tree(decisiontree_model, feature_names=X.columns, class_names = ['Died', 'Survived'],
filled = True, fontsize=10)
plt.show()
```

Code ۱۷: محاسبهٔ عمق، تعداد گره و تعداد شاخه

$$\text{Depth} = 22 \quad , \quad \text{Number of Leaves} = 132 \quad , \quad \text{Number of Nodes} = 263$$

در حالت max_depth = None، درخت تصمیم دارای ۲۶۳ گره، عمق ۲۲ و ۱۳۲ برگ است که نشان‌دهنده‌ی یک مدل بسیار پیچیده می‌باشد. این مدل عملاً برای هر مسافر قوانین بسیار خاصی را یاد گرفته و می‌توان گفت که داده‌های آموزشی را حفظ (Memorize) کرده است.

در مقابل، مدل با max_depth = 4 تنها دارای ۲۹ گره، عمق ۴ و ۱۵ برگ است. این ساختار ساده‌تر باعث می‌شود که مدل صرفاً قوانین کلی و عمومی داده‌ها را بیاموزد و از یادگیری جزئیات غیرضروری پرهیز کند.

از نظر دقت، مدل با عمق نامحدود به دلیل بیش‌برازش دچار افت عملکرد شده و دقت آن حدود **74.3**% است، در حالی که مدل با عمق ۴ به دقت بالاتری در حدود **78.8**% دست یافته است. این نتیجه نشان می‌دهد که با افزایش بیش از حد پیچیدگی مدل، دقت روی داده‌های آزمون کاهش یافته و پدیده‌ی Overfitting رخ می‌دهد.

با وجود این که مدل با عمق ۴ از نظر تعمیم‌پذیری گزینه‌ی مناسب‌تری است اما مقدار Recall برای کلاس بازماندگان در این مدل کمتر می‌باشد. برای بهبود این معیار، می‌توان از روش‌های Ensemble مانند Random Forest یا Gradient Boosting استفاده کرد تا بدون افزایش بیش از حد بیش‌برازش، عملکرد مدل بهبود یابد.

۲.۶ بررسی عمق‌های مختلف درخت

مدل را به ازای عمق‌های مختلف آموزش می‌دهیم و سپس آن را ارزیابی می‌کنیم:

```
depths = [3, 5, 10, None]
results = {}
train_accs = []
test_accs = []
plot_labels = []

for depth in depths:
```

```
    model = DecisionTreeClassifier(max_depth=depth, random_state=42)
    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)
    report = classification_report(y_test, y_test_pred)

    key_name = f"Depth_{depth}"
    results[key_name] = {
        'model': model,
        'train_acc': train_acc,
        'test_acc': test_acc,
        'report': report
    }

    train_accs.append(train_acc)
    test_accs.append(test_acc)
    plot_labels.append(str(depth) if depth is not None else "None")
    for key, val in results.items():
print(f"\n Classification Report for {key}:")
print(f'Test acc : {val["test_acc"]}')
print(f'Train acc : {val["train_acc"]}')
print(f'\n')
print(val['report'])
```

Code ۱۸: آموزش مدل به ازای عمق‌های مختلف و ارزیابی آن

- عمق ۳:

دقت آموزش: 0.8329 دقت آزمون: 0.7821

- عمق ۵:

دقت آموزش: 0.8567

دقت آزمون: 0.7430

- عمق ۱۰:

دقت آموزش: 0.9354

دقت آزمون: 0.7933

- عمق نامحدود :(None)

دقت آموزش: 1.0000

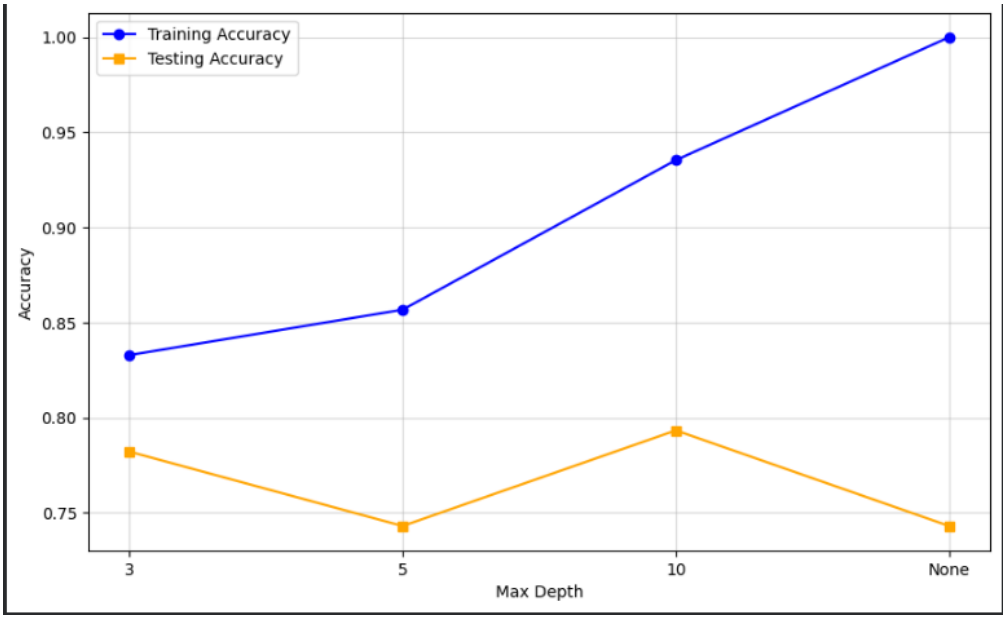
دقت آزمون: 0.7430

کلسیفیکیشن ریپورت نیز بر روی مجموعهٔ تست، برای هر عمق گرفته شده است.

در قسمت بعدی نمودار دقت برحسب عمق درخت برای دو مجموعهٔ آموزش و تست گرفته شده است:

```
plt.figure(figsize=(10, 6))
plt.plot(plot_labels, train_accs, marker='o', label='Training Accuracy', color='blue')
plt.plot(plot_labels, test_accs, marker='s', label='Testing Accuracy', color='orange')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True, alpha=0.5)

plt.show()
```

Code ۱۹: آموزش مدل به ازای عمق‌های مختلف و ارزیابی آن



شکل ۱۰: نمودار دقت برحسب عمق درخت برای دو مجموعهٔ آموزش و تست

:Overfitting

مدل با Depth = None تمام داده‌های مجموعه آموزش را حفظ کرده و بنابراین دقت آن روی داده‌های جدید و دیده‌نشده پایین است. دقت آموزش برابر ۱.۰ می‌باشد. در داده‌های واقعی مانند Titanic، رسیدن به ٪۱۰۰ معمولاً به این معناست که مدل قوانین را نمی‌آموزد و تنها شناسه‌ها (IDs) را حفظ کرده است.

:Underfitting

مدل با Depth = 3 شکاف بین دقت آموزش و دقت آزمون بسیار کم است. این نشان می‌دهد که مدل قابلیت generalization خوبی دارد، زیرا دقت آموزش و آزمون به هم نزدیک هستند. با این حال، مدل ممکن است کمی به سمت underfit شدن حرکت کند. در این حالت، مدل قوانین کلی را یاد گرفته است؛ به عنوان مثال: زنان بیشتر زنده می‌مانند و افراد ثروتمند بیشتر زنده می‌مانند.

:Depth = 5

این حالت حتی از Depth = 3 هم بدتر است! با افزایش پیچیدگی (عمق) دقت آزمون از ٪۷۸ به ٪۷۴ کاهش یافته است. این نشان می‌دهد که تقسیمات اضافه‌ای که بین عمق ۳ و ۵ ایجاد شد، بیشتر نویز را یاد گرفته‌اند و کمکی به عملکرد واقعی نکرده‌اند.

Depth = 10:

در این حالت بیشترین دقت آزمون (٪۷۹) به دست آمده است، اما دقت آموزش بسیار بالا (٪۹۴) است. با اینکه مدل در این حالت عملکرد خوبی دارد، شکاف بزرگ بین دقت آموزش و آزمون (٪۱۵) نشان می‌دهد که مدل در حال ناپایدار شدن و احتمالاً **overfit** شدن است.

۳.۶ هرس کردن درخت با **Cost-Complexity Pruning**:

ابتدا مدل را بدون محدودیت عمق روی داده‌های آموزش، آموزش می‌دهیم. سپس مقادیر ccpalpha را محاسبه می‌نماییم. همانطور که سوال از ما خواسته مقادیر مثبت آن را برمی‌داریم و آن را sort می‌کنیم. طول آن به اندازهٔ ۴۴ است.

در بخش بعد به ازای مقادیر مختلف از ccpalpha مدل را آموزش می‌دهیم و روی مجموعهٔ تست آن را ارزیابی می‌نماییم. نتایج آن در فایل پایتون قابل مشاهده است.

حال به ازای دقت‌های به دست آمده برای دو مجموعهٔ آموزش و تست، نمودار دقت برحسب ccpalpha که در مقیاس لگاریتمی است را ترسیم می‌نماییم.

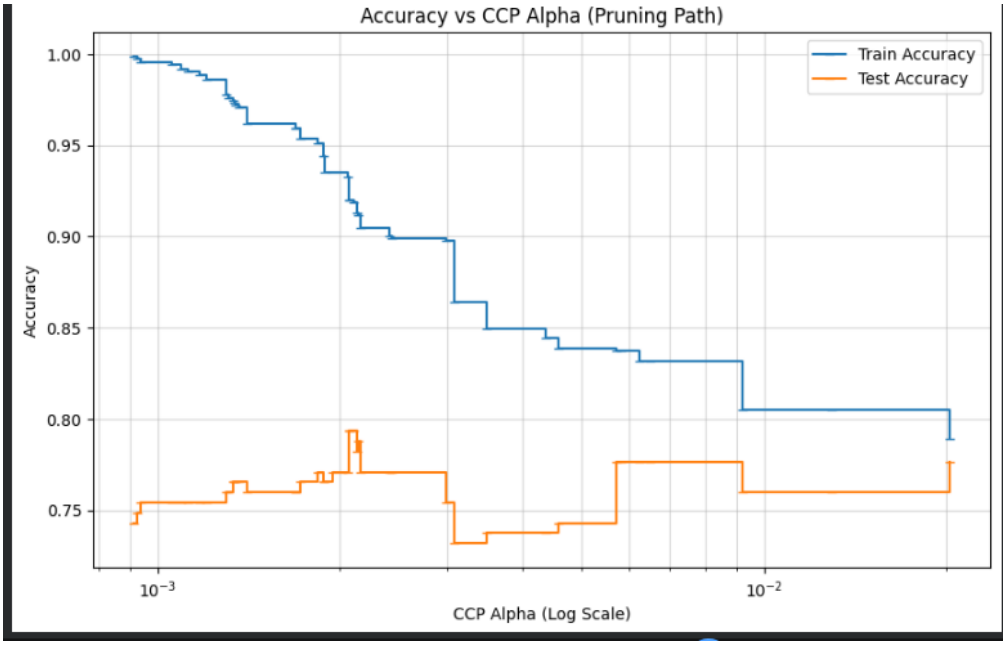
کدهای زده شده تا به اینجا همه به صورت زیر می‌باشند:

```
tree_ = DecisionTreeClassifier(random_state=42)
tree_.fit(X_train, y_train)
path = tree_.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas
impurities = path.impurities
ccp_alphas = [alpha for alpha in ccp_alphas if alpha > 0]
pruning_results = {}
train_accs = []
test_accs = []
plot_alphas = []
results = []
for alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=42, ccp_alpha=alpha)
    clf.fit(X_train, y_train)

    y_train_pred = clf.predict(X_train)
    y_test_pred = clf.predict(X_test)

    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)
    report = classification_report(y_test, y_test_pred)

    key = f"alpha_{alpha:.5f}" # Formatting key for readability
    pruning_results[key] = {
        'ccp_alpha': alpha,
        'model': clf,
        'train_acc': train_acc,
        'test_acc': test_acc,
```

```
        'classification report': report,
        'leaves': clf.get_n_leaves(),
        'depth' : clf.get_depth(),
        'nodes' : clf.tree_.node_count
    }
    results.append({
        'ccp_alpha': alpha,
        'model': clf,
        'train_acc': train_acc,
        'test_acc': test_acc,
        'classification report': report,
        'leaves': clf.get_n_leaves(),
        'depth' : clf.get_depth(),
        'nodes' : clf.tree_.node_count
    })

    train_accs.append(train_acc)
    test_accs.append(test_acc)
    plot_alphas.append(alpha)
    for key, val in pruning_results.items():
    print(f"\n Classification Report for {key}:")
    print(f'Test acc : {val["test_acc"]}')
    print(f'Train acc : {val["train_acc"]}')
    print(f'Depth : {val["depth"]}')
    print(f'leaves : {val["leaves"]}')
    print(f'\n')
    print(val['classification report'])
```

Code ۲۰: کدهای مربوط به هرس درخت

نتایج مربوط به آموزش هرمدل به دلیل زیاد بودن در فایل پایتون قابل مشاهده است. نمودار دقت برحسب ccpalpha به صورت زیر است:



شکل ۱۱: نمودار دقت برحسب ccpalpha برای دو مجموعهٔ آموزش و تست

مقادیر ccp_alpha معمولاً بسیار کوچک هستند (برای مثال 0.0001، 0.002 یا 0.005)، اما در انتهای بازه می‌توانند به‌صورت ناگهانی به مقادیر بزرگ‌تری مانند 0.1 یا 0.5 برسند. اگر این مقادیر روی یک محور خطی رسم شوند، بیش از ٪۹۰ از مقادیر کوچک در سمت چپ نمودار به‌صورت فشرده و غیرقابل تفکیک نمایش داده می‌شوند.

استفاده از مقیاس لگاریتمی این مشکل را برطرف می‌کند و باعث می‌شود تمام مقادیر ccp_alpha به‌صورت خوانا و قابل مقایسه نمایش داده شوند.

سمت راست نمودار (مقادیر بزرگ **ccp_alpha**): در این ناحیه، درخت تصمیم بیش از حد ساده شده است و مدل دچار کم‌برازش (Underfitting) می‌باشد. در نتیجه، دقت مدل کاهش می‌یابد.

سمت چپ نمودار (مقادیر کوچک **ccp_alpha**): در این ناحیه، درخت تصمیم بسیار پیچیده است و مدل دچار بیش‌برازش (Over-fitting) می‌شود. در این حالت، دقت روی داده‌های آموزش بالا است، اما دقت روی داده‌های آزمون کاهش می‌یابد.

حال مقادیر نتایج را بر روی دقت دادهٔ تست مرتب می‌نماییم و بهترین آن را بر می‌داریم. مشخصات مربوط به بهترین مدل در فایل پایتون آورده شده است.

مقایسه سه مدل درخت تصمیم

مقایسه از نظر قابلیت توضیح‌پذیری (**Explainability**)

بهترین مدل: مدل A (عمق **4**)

مدل A (**Depth 4**): این مدل تنها دارای 4 سطح و 15 برگ (تصمیم) است. ساختار درخت آن به‌قدری ساده است که می‌توان کل درخت را روی یک صفحه چاپ کرد. یک انسان می‌تواند مسیر تصمیم‌گیری را به‌راحتی دنبال کند و کاملاً متوجه شود که چرا یک مسافر به‌عنوان «زنده‌مانده» یا «فوت‌شده» طبقه‌بندی شده است. این مدل بر قوانین کلی و بدیهی تکیه دارد؛ مانند جنسیت، کلاس مسافرت و

سن.

مدل B (هرس‌شده): این مدل دارای 14 سطح و 79 گره است و دیگر به‌راحتی قابل تفسیر توسط انسان نیست. با وجود اینکه بهترین عملکرد را دارد، توضیح یک تصمیم خاص نیازمند دنبال‌کردن یک مسیر طولانی و پیچیده در درخت است.

مدل C (**Depth None**): این مدل به دلیل اندازه‌ی بسیار بزرگ خود عملاً یک جعبه سیاه محسوب می‌شود. با 263 گره، این درخت داده‌های آموزشی را تقریباً به‌صورت شناسه‌به‌شناسه حفظ کرده است. منطق تصمیم‌گیری آن قابل توضیح نیست، زیرا بیشتر شامل نویز است تا الگوی واقعی.

مقایسه از نظر تعمیم‌پذیری (**Overfitting** در مقابل **Underfitting**)

بهترین مدل: مدل B (هرس‌شده با بهترین **alpha**)

مدل A (**Depth 4**): کم‌برازش جزئی

شکاف بین دقت آموزش (84%) و دقت آزمون (79%) بسیار کم است که نشان‌دهنده‌ی تعمیم‌پذیری خوب مدل می‌باشد. اما مقدار **Recall** پایین (**0.55**) نشان می‌دهد که مدل بیش از حد ساده است و توانایی شناسایی بازماندگان «سخت‌تشخیص» (مانند مردان کلاس اول یا زنان کلاس سوم) را ندارد. این مدل تنها الگوهای واضح را یاد می‌گیرد.

مدل B (هرس‌شده): برازش متعادل

این مدل بالاترین دقت آزمون (79.3%) را به دست آورده است. عمق بیشتر (14 سطح) به مدل اجازه می‌دهد الگوهای ظریف‌تر را یاد بگیرد و مقدار **Recall** را به **0.71** افزایش دهد (یعنی شناسایی 71% از بازماندگان در مقایسه با 55% در مدل A).

هرچند شکاف بین دقت آموزش (91%) و آزمون (79%) بیشتر است، اما این میزان بیش‌برازش قابل قبول بوده و در ازای بهبود عملکرد، یک معامله‌ی منطقی محسوب می‌شود.

مدل C (**Depth None**): بیش‌برازش شدید

این مدل احتمالاً به دقت آموزشی نزدیک به 100% رسیده است، اما کمترین دقت آزمون را دارد. مدل به‌جای یادگیری سیگنال، نویز داده‌ها را آموخته است.

مقایسه از نظر عملکرد (**Precision** در مقابل **Recall**)

این بخش مهم‌ترین تفاوت بین مدل با عمق 4 و مدل هرس‌شده را نشان می‌دهد.

مدل A (**Depth 4**): محافظه‌کار (**Conservative**)

- **Precision** = **0.84** (بسیار بالا): زمانی که مدل پیش‌بینی می‌کند «زنده می‌مانی»، تقریباً همیشه درست است.
- **Recall** = **0.55** (بسیار پایین): تقریباً نیمی از بازماندگان واقعی را شناسایی نمی‌کند.

مثال دنیای واقعی: «فقط زمانی تو را نجات می‌دهم که کاملاً مطمئن باشم زنده می‌مانی؛ اگر کوچک‌ترین شکی وجود داشته باشد، تو را فوت‌شده در نظر می‌گیرم.»

مدل B (هرس‌شده): تهاجمی (**Aggressive**)

- **Precision** = **0.74** (کمتر): مدل خطاهای مثبت کاذب بیشتری دارد (پیش‌بینی بقا برای افرادی که فوت کرده‌اند).
- **Recall** = **0.71** (بسیار بالاتر): بخش قابل توجهی از بازماندگان واقعی را شناسایی می‌کند.

مثال دنیای واقعی: «برخی الگوهای ظریف را تشخیص داده‌ام؛ حتی اگر کمی ریسک داشته باشد، تو را زنده پیش‌بینی می‌کنم تا بازماندگان بیشتری را نجات دهم.»

در شمل زیر نیز به صورت یک جدول مقایسه بین سه مدل انجام شده است:

Feature	Model A: Depth 4	Model B: Pruned (Best Alpha)	Model C: Depth None
Test Accuracy	78.8%	**79.3% (Winner)**	~74.3%
Train Accuracy	84.1%	91.9%	**100% (Overfit)**
Generalization Gap	**5.3% (Best)**	12.6%	~25.7% (Worst)
Depth	**4 (Simplest)**	14	22
Total Nodes	**29**	79	263
Leaves	**15**	40	132
Recall (Survivors)	0.55 (Low)	**0.71 (High)**	0.71
Precision (Survivors)	**0.84 (High)**	0.74	0.65 (Low)
Explainability	**Excellent**	Moderate	None (Black Box)

شکل ۱۲: مقایسهٔ سه مدل

۷ پرسش یک: مفاهیم پایه **Ensemble Learning**

۱.۷ پاسخ الف: معرفی **Ensemble Learning** و چرایی عملکرد بهتر

یادگیری گروهی یا Ensemble Learning یک پارادایم در یادگیری ماشین است که در آن به جای استفاده از یک مدل یادگیری واحد (Single Learner)، مجموعه‌ای از مدل‌ها (که معمولاً «یادگیرنده‌های ضعیف» یا Weak Learners نامیده می‌شوند) آموزش داده شده و خروجی آن‌ها برای رسیدن به یک پیش‌بینی نهایی با هم ترکیب می‌شود.

چرا ترکیب مدل‌ها بهتر کار می‌کند؟ (تحلیل ریاضی و منطقی) دلیل اصلی موفقیت روش‌های Ensemble را می‌توان در سه جنبه اصلی بررسی کرد:

۱. جنبه آماری (کاهش واریانس): یک مدل تکی ممکن است به دلیل محدودیت داده‌های آموزشی، روی یک فرضیه خاص همگرا شود که لزوماً بهترین نیست. با ترکیب چندین مدل، ما در واقع میانگین فرضیات را می‌گیریم که خطر انتخاب یک فرضیه غلط را کاهش می‌دهد.

۲. جنبه محاسباتی (فرار از مینیمم‌های محلی): در الگوریتم‌هایی مانند شبکه‌های عصبی، مدل ممکن است در یک مینیمم محلی (Local Minimum) گیر کند. با شروع آموزش چندین مدل از نقاط مختلف، Ensemble فضای جستجو را بهتر پوشش می‌دهد.

۳. قضیه هیئت منصفه کندورسه (**Condorcet's Jury Theorem**): از نظر ریاضی، اگر ما N طبقه‌بند (Classifier) داشته باشیم که مستقل از هم باشند و احتمال خطای هر کدام کمتر از 0.5 باشد (یعنی کمی بهتر از تصادفی عمل کنند)، با افزایش تعداد مدل‌ها، احتمال اینکه رأی اکثریت صحیح باشد به سمت ۱ میل می‌کند.

$$\lim_{N \to \infty} P(\text{Correct Majority}) = 1$$

مثال دنیای واقعی: مسابقه مشهور Netflix Prize. برندگان این مسابقه برای بهبود سیستم توصیه فیلم نتفلیکس، از ترکیب ده‌ها مدل مختلف (شامل SVD، RBM، kNN) استفاده کردند تا خطای پیش‌بینی را به زیر ۱۰ درصد برسانند.

۲.۷ پاسخ ب: تفاوت Diversification و Aggregation

موفقیت یک سیستم Ensemble وابسته به دو رکن اصلی است: اینکه مدل‌ها با هم متفاوت باشند (تنوع) و اینکه چگونه نظرات آن‌ها ترکیب شود (تجمیع).

۱. مفهوم **Diversification** (تنوع‌بخشی):

تنوع به این معنی است که مدل‌های پایه نباید خطاهای یکسانی داشته باشند. اگر تمام مدل‌ها در یک نقطه خاص اشتباه کنند، ترکیب آن‌ها نیز اشتباه خواهد کرد. هدف Diversification ایجاد مدل‌هایی است که خطاهایشان «ناهمبسته» (Uncorrelated) باشد. روش‌های ایجاد تنوع:

- تنوع در داده‌های آموزشی: استفاده از تکنیک‌هایی مثل Bootstrap (در روش Bagging) که هر مدل با زیرمجموعه‌ای متفاوت از داده‌ها آموزش می‌بیند.
- تنوع در ویژگی‌ها: هر مدل تنها به زیرمجموعه‌ای از ویژگی‌ها (Features) دسترسی دارد (مانند Random Forest).
- تنوع در الگوریتم: استفاده همزمان از درخت تصمیم، شبکه عصبی و ماشین بردار پشتیبان (SVM).

۲. مفهوم **Aggregation** (تجمیع):

تجمیع مکانیزمی است که خروجی‌های مدل‌های متنوع را به یک خروجی واحد تبدیل می‌کند. روش‌های تجمیع:

- رأی‌گیری اکثریت (**Majority Voting**): برای مسائل طبقه‌بندی. کلاسی انتخاب می‌شود که بیشترین رأی را دارد.

$$\hat{y} = \text{mode}\{h_1(x), h_2(x), ..., h_n(x)\}$$

- میانگین‌گیری (**Averaging**): برای مسائل رگرسیون.

$$\hat{y} = \frac{1}{N}\sum_{i=1}^{N} h_i(x)$$

- میانگین‌گیری وزنی (**Weighted Averaging**): به مدل‌های دقیق‌تر وزن بیشتری داده می‌شود (مانند روش Boosting).

تحلیل تفاوت: Diversification در مرحله «آموزش» مدل‌ها رخ می‌دهد تا استقلال آن‌ها تضمین شود، در حالی که Aggregation در مرحله «استنتاج» (Inference) رخ می‌دهد تا خروجی‌ها ترکیب شوند.

۳.۷ پاسخ ج: مثال واقعی (سیستم تشخیص پزشکی هوشمند)

در اینجا یک سیستم Ensemble برای تشخیص سرطان سینه را تحلیل می‌کنیم که ترکیبی از داده‌های تصویربرداری و داده‌های بالینی است.

هدف: تشخیص خوش‌خیم (Benign) یا بدخیم (Malignant) بودن تومور.

ساختار سیستم:

۱. ورودی‌ها (**Inputs**):

- ورودی ۱: تصاویر ماموگرافی (داده‌های تصویری).
- ورودی ۲: سوابق ژنتیکی بیمار و نتایج آزمایش خون (داده‌های ساختاریافته عددی).
- ورودی ۳: گزارش متنی پزشک رادیولوژیست (داده‌های متنی).

۲. مدل‌های پایه (**Base Learners**) - ایجاد تنوع:

- مدل A (**CNN**): یک شبکه عصبی کانولوشنی (ResNet) که فقط تصاویر ماموگرافی را تحلیل می‌کند.
- مدل B (**Random Forest**): مدلی که روی داده‌های آزمایش خون و سن بیمار آموزش دیده است.
- مدل C (**BERT**): یک مدل زبانی که گزارش‌های متنی پزشکان قبلی را تحلیل می‌کند تا ریسک را بسنجد.

۳. روش تجمیع (**Aggregation**): از روش Stacking استفاده می‌شود. خروجی (احتمالات) هر سه مدل، A، B و C به عنوان ورودی به یک مدل نهایی (مثلاً یک Logistic Regression ساده) داده می‌شود.

$$P(\text{Final}) = w_1 P(A) + w_2 P(B) + w_3 P(C)$$

۴. خروجی (**Output**): احتمال نهایی بدخیم بودن تومور.

تحلیل چرایی موفقیت: ممکن است تصویر ماموگرافی در یک بافت متراکم مبهم باشد و مدل A اشتباه کند. اما مدل B با توجه به جهش ژنتیکی (BRCA1) ریسک را بالا تشخیص می‌دهد. سیستم Ensemble با تجمیع این اطلاعات، خطای مدل تصویری را با داده‌های ژنتیکی جبران کرده و تشخیص صحیح می‌دهد.

۸ پرسش دو: Random Forest و Bagging

داده‌های آموزشی ارائه شده به شرح زیر هستند:

جدول ۳: داده‌های اولیه

شناسه نمونه	x	y
۱	۲	۰
۲	۳	۰
۳	۸	۱
۴	۹	۱

۱.۸ پاسخ الف: فرآیند Bootstrap Sampling در Bagging

روش Bagging (مخفف Bootstrap Aggregating) بر پایه یک تکنیک آماری به نام Bootstrap Sampling بنا شده است. هدف این روش ایجاد چندین مجموعه داده جدید از روی داده‌های اصلی است تا واریانس مدل نهایی کاهش یابد.

مکانیزم دقیق نمونه‌برداری: فرض کنید مجموعه داده اصلی D دارای N نمونه است. برای ساخت یک مجموعه Bootstrap جدید (مثلاً B_k)، مراحل زیر انجام می‌شود:

۱. ما یک نمونه از داده‌های اصلی D را به صورت کاملاً تصادفی انتخاب می‌کنیم.

۲. آن نمونه را به مجموعه جدید B_k اضافه می‌کنیم.

۳. نکته کلیدی (جایگذاری مجدد یا **With Replacement**): نمونه انتخاب شده را به ظرف اصلی D برمی‌گردانیم. این یعنی شانس انتخاب مجدد همان نمونه در انتخاب‌های بعدی وجود دارد.

۴. این عمل را N بار تکرار می‌کنیم تا سایز مجموعه B_k دقیقاً برابر با سایز مجموعه اصلی D شود.

تحلیل آماری (قانون 63.2%):

از آنجا که نمونه‌برداری با جایگذاری انجام می‌شود، برخی نمونه‌ها ممکن است چندین بار در یک مجموعه Bootstrap ظاهر شوند و برخی دیگر اصلاً انتخاب نشوند. احتمال اینکه یک نمونه خاص در یک بار انتخاب شدن، انتخاب نشود برابر است با: $1 - \frac{1}{N}$. احتمال اینکه آن نمونه در کل N بار انتخاب، هرگز انتخاب نشود برابر است با:

$$P(\text{Selected Not}) = \left(1 - \frac{1}{N}\right)^N \tag{۷}$$

اگر N به اندازه کافی بزرگ باشد، حد این عبارت برابر است با:

$$\lim_{N\to\infty} \left(1 - \frac{1}{N}\right)^N = e^{-1} \approx 0.368$$

این بدین معناست که به‌طور میانگین، هر مجموعه Bootstrap شامل حدود 63.2% $(1 - 0.368)$ از داده‌های یکتای اصلی است و حدود 36.8% از داده‌ها در آن حضور ندارند (که به آن‌ها داده‌های Out-of-Bag یا OOB می‌گویند).

۲.۸ پاسخ ب: ایجاد سه نمونه **Bootstrap** احتمالی

داده‌های اصلی ما $D = \{(2,0), (3,0), (8,1), (9,1)\}$ هستند. تعداد نمونه‌ها $N = 4$ است. برای هر مجموعه Bootstrap باید **۴** بار نمونه‌برداری با جایگذاری انجام دهیم. یک سناریوی احتمالی به شرح زیر است:

مجموعه **Bootstrap 1** (B_1):

- انتخاب ۱: $(2,0)$
- انتخاب ۲: $(8,1)$
- انتخاب ۳: $(2,0)$ (تکراری)
- انتخاب ۴: $(9,1)$

$$B_1 = \{(2,0), (8,1), (2,0), (9,1)\}$$

در اینجا نمونه $(3,0)$ حذف شده است.

مجموعه **Bootstrap 2** (B_2):

- انتخاب ۱: $(3, 0)$
- انتخاب ۲: $(3, 0)$ (تکراری)
- انتخاب ۳: $(8, 1)$
- انتخاب ۴: $(8, 1)$ (تکراری)

$$B_2 = \{(3, 0), (3, 0), (8, 1), (8, 1)\}$$

در اینجا نمونه‌های $(2, 0)$ و $(9, 1)$ حذف شده‌اند.

مجموعه **Bootstrap 3** (B_3):

- انتخاب ۱: $(9, 1)$
- انتخاب ۲: $(2, 0)$
- انتخاب ۳: $(3, 0)$
- انتخاب ۴: $(8, 1)$

$$B_3 = \{(9, 1), (2, 0), (3, 0), (8, 1)\}$$

در این حالت خاص (که تصادفی است)، مجموعه دقیقاً مشابه داده‌های اصلی شد، اما ترتیب انتخاب متفاوت بوده است.

۳.۸ پاسخ ج: برتری **Random Forest** نسبت به درخت تصمیم معمولی

الگوریتم Random Forest معمولاً عملکرد بسیار بهتری نسبت به یک درخت تصمیم تکی (Single Decision Tree) دارد. دلایل این برتری عبارتند از:

۱. کاهش واریانس (**Variance Reduction**): درخت‌های تصمیم مدل‌هایی با «واریانس بالا» هستند؛ یعنی با تغییر اندک در داده‌های آموزشی، ساختار درخت به کلی تغییر می‌کند. Random Forest با میانگین‌گیری از صدها درخت (که هر کدام روی زیرمجموعه‌ای متفاوت از داده‌ها آموزش دیده‌اند)، واریانس کل را کاهش می‌دهد.

$$Var(\text{Forest}) \approx \frac{Var(\text{Tree})}{M}$$

(البته با در نظر گرفتن همبستگی بین درخت‌ها). این باعث می‌شود مدل نهایی پایدارتر باشد و کمتر دچار بیش‌برازش (Overfitting) شود.

۲. تنوع‌بخشی مضاعف (تفاوت با **Bagging** معمولی): یک درخت تصمیم معمولی در هر گره، تمام ویژگی‌ها را بررسی می‌کند تا بهترین شکاف را بیابد. اگر یک ویژگی بسیار قوی در داده‌ها وجود داشته باشد، تمام درخت‌های موجود در Bagging از همان ویژگی در ریشه استفاده می‌کنند و درخت‌ها شبیه به هم (Correlated) می‌شوند.

اما Random Forest یک گام فراتر می‌رود: در هر گره، تنها زیرمجموعه‌ای تصادفی از ویژگی‌ها (مثلاً $\sqrt{p}$) را در نظر می‌گیرد. این کار باعث می‌شود:

- همبستگی بین درخت‌ها کاهش یابد.
- درخت‌ها مجبور شوند از ویژگی‌های دیگر نیز استفاده کنند و الگوهای پنهان‌تری را کشف کنند.

۳. مثال مفهومی: فرض کنید می‌خواهیم مرز بین کلاس‌ها را یاد بگیریم.

- درخت تکی: ممکن است یک مرز پلکانی تیز و غیرعادی ایجاد کند که نویزها را هم پوشش دهد.
- جنگل تصادفی: با روی هم انداختن صدها مرز پلکانی مختلف، نتیجه نهایی شبیه به یک منحنی صاف و هموار می‌شود که تعمیم‌پذیری بهتری روی داده‌های جدید دارد.

۹ پرسش سه: Boosting

۱.۹ پاسخ قسمت ۱: مفهوم Boosting و تفاوت با Bagging

مفهوم **Boosting**:

روش Boosting (تقویت) یک خانواده از الگوریتم‌های یادگیری گروهی (Ensemble Learning) است که هدف اصلی آن تبدیل مجموعه‌ای از «یادگیرنده‌های ضعیف» (Weak Learners) به یک «یادگیرنده قوی» (Strong Learner) است. برخلاف Bagging که مدل‌ها را به صورت موازی می‌سازد، Boosting مدل‌ها را به صورت توالی **(Sequential)** ایجاد می‌کند. ایده اصلی این است: هر مدل جدید تلاش می‌کند خطاهای مدل قبلی را جبران کند. به عبارت دیگر، تمرکز الگوریتم روی نمونه‌هایی است که مدل‌های قبلی در پیش‌بینی آن‌ها ناتوان بوده‌اند.

تفاوت‌های کلیدی با **Bagging**:

جدول ۴: مقایسه Bagging و Boosting

ویژگی	(Bootstrap Aggregating) Bagging	Boosting
نحوه ساخت مدل‌ها	موازی و مستقل از هم	سریالی و وابسته به هم
هدف اصلی	کاهش واریانس (Variance Reduction)	کاهش بایاس (Bias Reduction)
وزن‌دهی نمونه‌ها	یکسان (انتخاب تصادفی)	وزن‌دهی به نمونه‌های سخت (خطادار)
نقاط قوت	جلوگیری از بیش‌برازش (Overfitting)	افزایش دقت روی داده‌های آموزشی
حساسیت به نویز	مقاوم در برابر نویز	حساس به نویز و داده‌های پرت
مثال معروف	Random Forest	AdaBoost، XGBoost، Gradient Boosting

۲.۹ پاسخ قسمت ۲: الگوریتم AdaBoost و نقش وزن‌ها

الگوریتم AdaBoost (Adaptive Boosting) یکی از اولین و محبوب‌ترین الگوریتم‌های Boosting است.

نقش حیاتی وزن نمونه‌ها:

در AdaBoost، به هر نمونه آموزشی (x_i, y_i) یک وزن w_i اختصاص داده می‌شود. این وزن نشان‌دهنده «اهمیت» آن نمونه است.

- در ابتدا، تمام نمونه‌ها وزن برابر دارند.

- پس از آموزش هر مدل ضعیف، وزن نمونه‌هایی که اشتباه طبقه‌بندی شده‌اند افزایش می‌یابد.
- وزن نمونه‌هایی که درست طبقه‌بندی شده‌اند کاهش می‌یابد.

این مکانیزم باعث می‌شود مدل بعدی مجبور شود تمرکز خود را روی نمونه‌های سخت (آن‌هایی که وزنشان زیاد شده) بگذارد.
مراحل دقیق الگوریتم **AdaBoost**: فرض کنید داده‌های آموزشی $\{(x_1, y_1), ..., (x_N, y_N)\}$ هستند که $y_i \in \{-1, +1\}$.

۱. مقداردهی اولیه: وزن تمام نمونه‌ها برابر تنظیم می‌شود:

$$w_i^{(1)} = \frac{1}{N} \quad \text{for} i = 1, ..., N$$

۲. تکرار برای $t = 1$ تا T:

- الف) یک مدل ضعیف $h_t(x)$ را با استفاده از توزیع وزنی $w^{(t)}$ آموزش دهید تا خطای وزنی زیر مینیمم شود:

$$\epsilon_t = \sum_{i=1}^{N} w_i^{(t)} \mathbb{I}(y_i \neq h_t(x_i))$$

(که $\mathbb{I}$ تابع نشانگر است که اگر شرط برقرار باشد ۱ وگرنه ۰ است).

- ب) اهمیت (وزن) مدل h_t را محاسبه کنید (α_t):

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{۸}$$

(اگر خطا کم باشد، α_t بزرگ می‌شود؛ یعنی مدل دقیق‌تر، حرفش در رأی‌گیری نهایی خریدار بیشتری دارد).

- ج) به‌روزرسانی وزن نمونه‌ها برای مرحله بعد:

$$w_i^{(t+1)} = w_i^{(t)} \exp\left(-\alpha_t y_i h_t(x_i)\right) \tag{۹}$$

تحلیل فرمول: اگر پیش‌بینی درست باشد ($y_i h_t(x_i) = 1$)، توان منفی شده و وزن کم می‌شود. اگر اشتباه باشد، توان مثبت شده و وزن زیاد می‌شود.

- د) نرمال‌سازی وزن‌ها: $w_i^{(t+1)} = \frac{w_i^{(t+1)}}{\sum w_j^{(t+1)}}$ تا مجموعشان ۱ شود.

۳. خروجی نهایی: ترکیب خطی وزن‌دار مدل‌های ضعیف:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

۳.۹ پاسخ قسمت ۳: قدرت مدل و حساسیت به نویز

چرا **Boosting** مدل‌های بسیار قوی تولید می‌کند؟
قدرت Boosting از توانایی آن در کاهش شدید بایاس (**Bias**) ناشی می‌شود. حتی اگر یادگیرنده‌های اولیه بسیار ساده باشند (مثلاً درخت‌های تصمیم با عمق ۱ که به Decision Stumps معروفند)، Boosting با تمرکز مکرر بر روی نقاط ضعف، مرزهای تصمیم بسیار پیچیده و غیرخطی ایجاد می‌کند. این الگوریتم اثبات کرده است که می‌تواند خطای آموزشی را به صفر میل دهد.
چرا نسبت به نویز حساس است؟ (پاشنه آشیل **Boosting**)
این الگوریتم ذاتا سعی می‌کند «سخت‌ترین» نمونه‌ها را یاد بگیرد.

- تحلیل: اگر داده‌های ما دارای نویز باشند (مثلاً برچسب اشتباه یا داده‌های پرت)، الگوریتم Boosting نمی‌تواند تشخیص دهد که این نمونه نویز است.
- در عوض، فکر می‌کند که این یک نمونه «سخت» است که هنوز یاد گرفته نشده.
- در نتیجه، در هر مرحله وزن این نمونه نویزی را به شدت افزایش می‌دهد و مدل‌های بعدی تمام تلاش خود را می‌کنند تا خود را با این داده غلط تطبیق دهند.
- این رفتار منجر به ایجاد مرزهای تصمیم عجیب و غریب و پیچیده در اطراف داده‌های نویزی می‌شود که مصداق بارز بیش‌برازش **(Overfitting)** است.

مثال واقعی: فرض کنید در حال طراحی یک سیستم تشخیص هرزنامه (Spam Detection) هستید.

- اکثر ایمیل‌های اسپم کلمات مشخصی دارند. اما یک ایمیل سالم هم وجود دارد که به دلیل اشتباه کاربر یا سیستم، برچسب Spam خورده است (نویز).
- الگوریتم AdaBoost در دور اول آن را اشتباه تشخیص می‌دهد. در دورهای بعدی، وزن این ایمیل خاص را آنقدر بالا می‌برد که مدل مجبور می‌شود قانونی بسازد که این ایمیل را اسپم بداند.
- نتیجه: مدل نهایی ممکن است بسیاری از ایمیل‌های سالم و مشابه آن ایمیل خاص را نیز به اشتباه اسپم تشخیص دهد، فقط برای اینکه آن یک داده نویزی را پوشش دهد.

۱۰ پرسش چهار: Stacking

۱.۱۰ پاسخ الف: مفهوم Stacking و Meta-Learner

روش Stacking (یا Stacked Generalization) یک تکنیک یادگیری گروهی پیشرفته است که در آن چندین مدل رگرسیون یا طبقه‌بندی (معمولاً از انواع مختلف) با یکدیگر ترکیب می‌شوند. برخلاف Bagging (که بر پایه رأی‌گیری یا میانگین‌گیری ساده است) و Boosting (که مدل‌ها را به صورت متوالی اصلاح می‌کند)، Stacking از یک مدل یادگیرنده دیگر استفاده می‌کند تا یاد بگیرد چگونه پیش‌بینی‌های مدل‌های پایه را با هم ترکیب کند.

ساختار دو سطحی:

۱. مدل‌های سطح صفر (**Base Learners** یا **Level-0 Models**):

این‌ها مدل‌های یادگیری ماشینی متنوعی هستند (مثلاً یک SVM، یک Random Forest و یک KNN) که روی داده‌های اصلی آموزش می‌بینند. هدف این است که این مدل‌ها تا حد امکان متفاوت باشند (Diverse) تا نقاط قوت یکدیگر را پوشش دهند.

۲. مدل سطح یک (**Meta-Learner** یا **Level-1 Model**):

این مدل نهایی است که وظیفه «تجمیع هوشمندانه» را بر عهده دارد.

- ورودی **Meta-Learner**: ورودی این مدل، داده‌های اصلی نیستند؛ بلکه خروجی‌ها (پیش‌بینی‌های) مدل‌های سطح صفر هستند.

- هدف **Meta-Learner**: این مدل یاد می‌گیرد که کدام مدل پایه در چه شرایطی عملکرد بهتری دارد. برای مثال، ممکن است یاد بگیرد که "اگر مدل SVM احتمال کلاس ۱ را بالا داد اما KNN احتمال کمی داد، باید به حرف SVM اعتماد کرد".

به زبان ساده، Meta-Learner نقش یک «مدیر» را بازی می‌کند که نظرات کارشناسان مختلف (مدل‌های پایه) را می‌شنود و تصمیم نهایی را می‌گیرد.

۲.۱۰ پاسخ ب: ضرورت استفاده از Cross-Validation

سوالی کلیدی در Stacking این است: چگونه داده‌های آموزشی برای *Meta-Learner* را تهیه کنیم؟ اگر ما مدل‌های پایه را روی کل داده‌های آموزشی ($Train$) آموزش دهیم و سپس از همان مدل‌ها بخواهیم روی همان داده‌های $Train$ پیش‌بینی انجام دهند تا ورودی Meta-Learner ساخته شود، دچار مشکل جدی نشت داده (**Data Leakage**) می‌شویم.

استدلال دقیق و چرایی نیاز به **CV**:

۱. مشکل بیش‌برازش در مدل‌های پایه: مدل‌های پایه (مثل درخت تصمیم) ممکن است داده‌های آموزشی را حفظ کرده باشند. اگر از آن‌ها بخواهیم روی داده‌های آموزشی پیش‌بینی کنند، ممکن است دقتی نزدیک به ٪۱۰۰ داشته باشند.

۲. فریب خوردن **Meta-Learner**: اگر Meta-Learner با این پیش‌بینی‌های مصنوعیِ عالی آموزش ببیند، یاد می‌گیرد که همواره به مدل‌های پایه اعتماد مطلق کند و هیچ اصلاحی انجام ندهد. اما وقتی داده‌های تست واقعی (دیده نشده) وارد شوند، مدل‌های پایه خطا خواهند داشت و Meta-Learner (که برای شرایط ایده‌آل آموزش دیده) شکست می‌خورد.

راه حل: پیش‌بینی‌های خارج از قطعه (**Out-of-Fold Predictions**):

برای حل این مشکل از روش K-Fold Cross-Validation استفاده می‌کنیم تا مطمئن شویم پیش‌بینی‌هایی که به Meta-Learner داده می‌شود، توسط مدل‌هایی تولید شده‌اند که آن داده خاص را در حین آموزش ندیده‌اند.

فرآیند گام‌به‌گام:

- داده‌ها به K بخش تقسیم می‌شوند (مثلاً ۵ بخش).
- برای هر بخش k (به عنوان داده اعتبارسنجی):
  - مدل‌های پایه روی $K-1$ بخش دیگر آموزش می‌بینند.
  - سپس روی بخش k (که ندیده‌اند) پیش‌بینی انجام می‌دهند.
- این پیش‌بینی‌ها جمع‌آوری شده و تشکیل یک ستون ویژگی جدید برای آموزش Meta-Learner می‌دهند.

۳.۱۰ پاسخ ج: مثال معماری یک سیستم Stacking

فرض کنید می‌خواهیم قیمت خانه را پیش‌بینی کنیم (مسئله رگرسیون). معماری پیشنهادی به صورت زیر است:

۱. مجموعه داده: شامل N سطر داده با ویژگی‌هایی مثل متراژ، تعداد اتاق و سال ساخت (X) و قیمت واقعی (Y).

۲. لایه اول (**Base Learners**): ما از سه مدل متنوع استفاده می‌کنیم تا الگوهای مختلف را کشف کنند:

- مدل A (**K-Nearest Neighbors**): برای کشف شباهت‌های محلی بین خانه‌ها.

- مدل B (**Support Vector Machine**): برای پیدا کردن مرزهای پیچیده و غیرخطی.
- مدل C (**Random Forest**): برای مدل‌سازی قوانین تصمیم‌گیری و مدیریت داده‌های ناهمگن.

خروجی این لایه: سه عدد پیش‌بینی شده برای هر خانه: $\hat{y}_A, \hat{y}_B, \hat{y}_C$.

۳. مجموعه داده جدید (**Meta-Features**): با استفاده از روش Cross-Validation که در بخش قبل توضیح داده شد، ما یک جدول داده جدید می‌سازیم که N سطر و ۳ ستون دارد. ستون‌ها همان پیش‌بینی‌های مدل‌های A و B و C هستند.

۴. لایه دوم (**Meta-Learner**): یک مدل ساده و خطی (معمولاً) انتخاب می‌شود تا از بیش‌برازش جلوگیری کند.

- مدل: رگرسیون خطی (Linear Regression).
- آموزش: این مدل یاد می‌گیرد که چگونه $\hat{y}_A, \hat{y}_B, \hat{y}_C$ را ترکیب کند تا به Y واقعی نزدیک شود.
- رابطه ریاضی:

$$Final\_Prediction = w_0 + w_1(\hat{y}_A) + w_2(\hat{y}_B) + w_3(\hat{y}_C)$$

در این معماری، اگر مثلاً KNN در مناطق پرتراکم شهر خوب عمل کند ولی در حومه شهر بد باشد، رگرسیون خطی نهایی ضرایب w را طوری تنظیم می‌کند که در آن شرایط خاص، وزن کمتری به KNN داده شود.

۱۱ پرسش پنج: مزایا و معایب Ensemble Learning

استفاده از روش‌های یادگیری گروهی در پروژه‌های عملی و مسابقات داده‌کاوی (مانند Kaggle) بسیار رایج است. با این حال، انتخاب این رویکرد نیازمند درک دقیق مبادله (Trade-off) بین دقت و منابع است.

۱.۱۱ پاسخ قسمت الف: مزایا و نقاط قوت

۱. بهبود قابل توجه دقت پیش‌بینی (**Improved Accuracy**):
اصلی‌ترین دلیل استفاده از Ensemble، دستیابی به دقتی بالاتر از تک‌تک مدل‌های تشکیل‌دهنده آن است.

- استدلال: همانطور که پیش‌تر با قضیه کندورسه اشاره شد، تجمیع نظرات چندین مدل (حتی اگر مدل‌های ضعیفی باشند)، احتمال خطای کلی را کاهش می‌دهد. این روش‌ها می‌توانند خطای ناشی از بایاس (در Boosting) و واریانس (در Bagging) را همزمان مدیریت کنند و به خطای تعمیم‌یافتگی (Generalization Error) کمتری دست یابند.

۲. پایداری و استحکام مدل (**Robustness and Stability**):
مدل‌های تکی، به خصوص درخت‌های تصمیم، نسبت به نویز و تغییرات کوچک در داده‌های آموزشی بسیار حساس هستند (واریانس بالا).

- تحلیل: با میانگین‌گیری از نتایج چندین مدل متنوع، اثر نویزها خنثی می‌شود. اگر یک مدل به دلیل نویز خاصی در داده‌ها منحرف شود، سایر مدل‌ها (که آن نویز را ندیده‌اند یا جور دیگری یاد گرفته‌اند) این خطا را جبران می‌کنند. نتیجه نهایی مدلی است که در برابر تغییرات داده‌های ورودی بسیار مقاوم‌تر است.

۳. مدل‌سازی مرزهای تصمیم پیچیده (**Handling Complex Boundaries**):
برخی مدل‌های پایه (مثل طبقه‌بندهای خطی) به تنهایی نمی‌توانند داده‌هایی با ساختار غیرخطی پیچیده را تفکیک کنند.

- استدلال: روش‌های Ensemble می‌توانند با ترکیب چندین مرز تصمیم ساده، یک مرز تصمیم بسیار پیچیده و انعطاف‌پذیر بسازند. برای مثال، ترکیب تعداد زیادی خط صاف (در Boosting) می‌تواند یک منحنی بسیار دقیق را تقریب بزند که هیچ‌کدام از آن خطوط به تنهایی قادر به ساخت آن نبودند.

۴. اجتناب از بهینه‌های محلی (**Avoiding Local Minima**):
در الگوریتم‌هایی که از روش‌های بهینه‌سازی (مانند گرادیان کاهشی) استفاده می‌کنند، خطر گیر افتادن در مینیمم‌های محلی وجود دارد.

- تحلیل: یک مدل تکی ممکن است در یک نقطه شروع نامناسب، در یک بهینه محلی گیر کند. اما در Ensemble، ما چندین مدل را از نقاط شروع مختلف (Random Seeds) آغاز می‌کنیم. تجمیع این مدل‌ها باعث می‌شود که فضای جستجو بهتر پوشش داده شود و شانس رسیدن به بهینه سراسری (Global Optimum) افزایش یابد.

۲.۱۱ پاسخ قسمت ب: معایب و چالش‌ها

۱. کاهش تفسیرپذیری (**Loss of Interpretability**):
یکی از بزرگترین معایب تبدیل شدن یک مدل ساده به Ensemble، تبدیل شدن آن به یک «جعبه سیاه» (Black Box) است.

- استدلال: یک درخت تصمیم تکی به راحتی قابل رسم و فهم است (مثلاً: اگر سن > آنگاه بیماری دارد). اما وقتی شما یک Random Forest با ۱۰۰۰ درخت دارید، دیگر نمی‌توانید به راحتی توضیح دهید که چرا سیستم تصمیم خاصی گرفته است. این موضوع در حوزه‌های حساس (پزشکی، حقوقی) که نیاز به توضیح‌پذیری (Explainability) دارند، چالش‌برانگیز است.

۲. هزینه محاسباتی بالا در زمان آموزش (**High Computational Cost**):
آموزش یک سیستم Ensemble به مراتب سنگین‌تر از یک مدل تکی است.

- تحلیل: اگر شما از ۱۰۰ مدل پایه استفاده می‌کنید، زمان آموزش و منابع پردازشی (CPU/GPU) مورد نیاز شما تقریباً ۱۰۰ برابر می‌شود. در روش‌هایی مثل Boosting که مدل‌ها باید به صورت سریال (پشت سر هم) آموزش ببینند، حتی قابلیت موازی‌سازی هم وجود ندارد و زمان آموزش بسیار طولانی می‌شود.

۳. کندی در زمان استنتاج (**Slow Inference/Prediction Latency**):
علاوه بر زمان آموزش، زمان پاسخ‌دهی به کاربر نهایی نیز افزایش می‌یابد.

- استدلال: برای اینکه سیستم یک پیش‌بینی انجام دهد، باید داده ورودی را از تمام مدل‌های زیرمجموعه عبور دهد و سپس نتایج را ترکیب کند. این تاخیر (Latency) ممکن است برای سیستم‌های بلادرنگ (Real-time) مانند ترمز خودکار خودرو یا معاملات بورسی فرکانس‌بالا مناسب نباشد.

۴. حجم بالای ذخیره‌سازی و پیچیدگی پیاده‌سازی (**Storage and Complexity**):
مدل‌های Ensemble حافظه زیادی اشغال می‌کنند.

- تحلیل: ذخیره کردن پارامترهای صدها یا هزاران مدل نیازمند فضای دیسک و رم زیادی است. این مسئله پیاده‌سازی این مدل‌ها را روی دستگاه‌های لبه (Edge Devices) مانند موبایل‌ها یا سنسورهای اینترنت اشیا (IoT) که محدودیت حافظه دارند، دشوار یا غیرممکن می‌سازد.

۱۲ پرسش شش: پیاده‌سازی Ensemble

۱.۱۲ کتابخانه sklearn.ensemble

کتابخانه scikit-learn (یا به اختصار sklearn) قدرتمندترین و محبوب‌ترین کتابخانه یادگیری ماشین در زبان پایتون است. ماژول ensemble در این کتابخانه، مجموعه‌ای جامع از روش‌های یادگیری گروهی را فراهم می‌کند.
این ماژول شامل کلاس‌های بهینه‌سازی شده‌ای برای پیاده‌سازی الگوریتم‌های مبتنی بر درخت است:

- برای **Random Forest**: از کلاس RandomForestClassifier استفاده می‌شود که پیاده‌سازی روش Bagging با درخت‌های تصمیم است.
- برای **Gradient Boosting**: از کلاس GradientBoostingClassifier استفاده می‌شود که پیاده‌سازی روش Boosting است و از تابع هزینه مشتق‌پذیر برای بهبود مرحله‌به‌مرحله استفاده می‌کند.

این کلاس‌ها امکان تنظیم دقیق هایپرپارامترها، مدیریت وزن‌دهی به کلاس‌ها و موازی‌سازی (برای Random Forest) را فراهم می‌کنند.

۲.۱۲ مدل‌های Random Forest و Gradient Boosting

در این آزمایش، دو مدل با تنظیمات زیر تعریف شده‌اند:
۱. مدل **Random Forest**:

```
rf_model = RandomForestClassifier(n_estimators=200, random_state=42)
```

۲. مدل **Gradient Boosting**:

```
gb_model = GradientBoostingClassifier(n_estimators=200, random_state=42)
```

توضیح پارامتر **n_estimators**:
این پارامتر نشان‌دهنده «تعداد تخمین‌گرها» یا همان تعداد درخت‌های تصمیم پایه‌ای است که در مدل گروهی ساخته می‌شوند.

- در Random Forest: تعداد ۲۰۰ درخت مستقل ساخته می‌شود که در نهایت بین آن‌ها رأی‌گیری انجام می‌شود. افزایش این عدد معمولاً باعث پایداری بیشتر مدل می‌شود و ریسک بیش‌برازش را (برخلاف Boosting) لزوماً افزایش نمی‌دهد.
- در Gradient Boosting: تعداد ۲۰۰ مرحله اصلاح انجام می‌شود. هر درخت جدید سعی می‌کند خطای درختان قبلی را اصلاح کند. در اینجا، اگر این عدد بیش از حد بزرگ انتخاب شود، مدل شروع به یادگیری نویزها کرده و دچار بیش‌برازش می‌شود.

۳.۱۲ تحلیل نتایج

نتایج بصری:

```
Training Accuracy: 1.0000
Testing Accuracy:  0.7933
classification report for Test:

              precision    recall  f1-score   support

           0       0.81      0.86      0.84       110
           1       0.76      0.68      0.72        69

    accuracy                           0.79       179
   macro avg       0.79      0.77      0.78       179
weighted avg       0.79      0.79      0.79       179
```

شکل ۱۳: نتیجه Random Forest

```
Training Accuracy: 0.9480
Testing Accuracy:  0.7877
classification report for Test:

              precision    recall  f1-score   support

           0       0.80      0.88      0.84       110
           1       0.77      0.64      0.70        69

    accuracy                           0.79       179
   macro avg       0.78      0.76      0.77       179
weighted avg       0.79      0.79      0.78       179
```

شکل ۱۴: نتیجه Gradient Boosting

تحلیل و بررسی کمی نتایج:

در جدول زیر خلاصه عملکرد دو مدل مقایسه شده است:

جدول ۵: مقایسه عملکرد Random Forest و Gradient Boosting

ویژگی (Feature)	Random Forest	Gradient Boosting	برنده نهایی
Training Score	(Pure Memory) 100%	(Hard Learning) 94.8%	GB (منطقی‌تر)
Test Score	79.33%	78.77%	Random Forest
Overfitting Gap	(بسیار زیاد) ~ 20.7%	(زیاد) ~ 16.0%	GB (شکاف کمتر)
Finding Survivors (Recall)	0.68	0.64	Random Forest

۱. تحلیل عمیق مدل **Random Forest** (با ۲۰۰ درخت):

آمار نشان‌دهنده وضعیت جالبی است:

- دقت آموزش ٪۱۰۰: این عدد نشان می‌دهد که مدل دقیقاً داده‌ها را «حفظ» (Memorize) کرده است. دلیل این امر تنظیمات پیش‌فرض کتابخانه scikit-learn است که در آن درخت‌ها تا عمق بی‌نهایت رشد می‌کنند (بدون هرس کردن).
- شکاف عملکرد (**The Gap**): اختلاف فاحش ۲۰.۷ درصدی بین آموزش و تست، نشانگر یادگیری نویزها و جزئیات تصادفی در داده‌های آموزشی است که در داده‌های تست وجود ندارند.
- نقطه قوت: با وجود بیش‌برازش شدید، دقت تست (۷۹.۳٪) و نرخ بازیابی (Recall) برابر با ۰.۶۸ است که بهتر از مدل رقیب است. این موفقیت مدیون مکانیزم «رأی‌گیری» (Voting) است که خطاهای فردی درخت‌های بیش‌برازش شده را تا حد زیادی خنثی کرده است.

۲. تحلیل عمیق مدل **Gradient Boosting** (با ۲۰۰ درخت):

- دقت آموزش **۹۴.۸**٪: این مدل به جای حفظ کردن، سعی کرده با اصلاح خطاهای متوالی یاد بگیرد، اما همچنان بیش‌برازش بالایی دارد.
- نقطه ضعف حیاتی (**Low Recall**): نرخ بازیابی کلاس ۱ (بازماندگان) تنها ۰.۶۴ است. این نشان می‌دهد مدل «محافظه‌کار» عمل کرده و تمرکز خود را روی کلاس اکثریت (فوت‌شدگان) گذاشته است تا خطای کلی را کاهش دهد، اما در تشخیص کلاس مهم‌تر (بازماندگان) ضعیف‌تر عمل کرده است.

۵. ریشه اصلی مشکل (**Root Cause of Overfitting**):

هر دو مدل از یک مشکل مشترک رنج می‌برند: پیچیدگی نامحدود (**Unconstrained Complexity**).

۱. در Random Forest، اجازه رشد نامحدود درخت‌ها باعث ایجاد برگ‌هایی شده که هر کدام فقط یک نمونه خاص را پوشش می‌دهند.

۲. در Gradient Boosting، تعداد ۲۰۰ درخت (n_estimators=200) بدون محدودیت عمق کافی، به مدل فرصت داده تا تمام نویزهای موجود در داده‌های آموزشی را مدل‌سازی کند.

برای بهبود، باید از تکنیک‌هایی مثل محدود کردن عمق درخت (max_depth) یا تنظیم حداقل نمونه در هر برگ (min_samples_leaf) استفاده کرد.