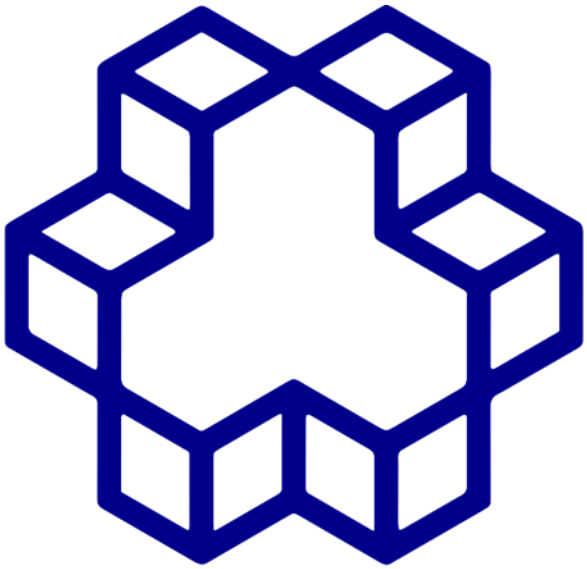
به نام خدا



دانشگاه صنعتی خواجه نصیرالدین طوسی

گزارش کار مبانی سیستم های هوشمند

ارشیا کلانتریان ۴۰۱۲۱۹۹۳

محمد حسین گل محمدی ۴۰۲۲۰۹۰۳

مینی پروژه سوم

۱.

یکی از ابزارهای قدرتمند در حوزه یادگیری ماشین و تحلیل‌های پیش‌بینانه است که با الهام از ساختار سلسله‌مراتب منطقی، فرآیند تصمیم‌گیری را مدل‌سازی می‌کند. دلیل اصلی انتخاب این ساختار در بسیاری از پروژه‌های داده‌کاوی، شفافیت عملیاتی و قابلیت بصری‌سازی آن است. برخلاف بسیاری از مدل‌های پیچیده ریاضی، درخت تصمیم این امکان را فراهم می‌آورد که منطقِ پشتِ هر خروجی به‌سادگی قابل ردیابی و تحلیل باشد. همچنین، توانایی این مدل در مدیریت هم‌زمان داده‌های کمی و کیفی، آن را به گزینه‌ای ایده‌آل برای مجموعه‌داده‌های واقعی تبدیل کرده است

. **ارکان ساختاری درخت تصمیم**

منطقِ عملکردی این مدل بر پایه سه بخش اصلی استوار است که جریان داده را از مبدأ تا مقصد هدایت می‌کنند:

گره‌های میانی :(Internal Nodes) این گره‌ها نقش نقاط فیلتراسیون یا پرسش‌گری را ایفا می‌کنند. در هر گره میانی، یک ویژگی (Feature) خاص مورد آزمون قرار می‌گیرد تا داده‌ها بر اساس یک آستانه یا شرط مشخص، به دسته‌های کوچک‌تر تقسیم شوند. هدف از این گره‌ها، کاهش گنگی (Entropy) و افزایش خلوص داده‌ها در مسیر رسیدن به پاسخ است.

شاخه‌ها :(Branches) شاخه‌ها نشان‌دهنده برآیندِ تصمیمات در هر مرحله هستند. در واقع هر شاخه، مسیری است که داده‌ها بر اساس برقراری یا عدم برقراری شرطِ موجود در گره قبلی طی می‌کنند. این مسیرها جریان منطقی از "شرط" به "نتیجه" را برقرار می‌سازند.

گره‌های برگ :(Leaf Nodes) برگ‌ها پایان‌بخش ساختار درخت هستند. در این گره‌ها هیچ تقسیم‌بندی دیگری صورت نمی‌گیرد و حاوی پاسخ نهایی یا کلاسِ پیش‌بینی شده (در مسائل طبقه‌بندی) و یا مقدار تخمینی (در مسائل رگرسیون) می‌باشند.

پیاده‌سازی در دنیای واقعی: سیستم ارزیابی اعتبار بانکی

به منظور درک کاربردی، می‌توان سیستم **اعتبارسنجی متقاضیان وام** را در نظر گرفت. در این سناریو، هدفِ مدل، تشخیص میزان ریسک بازپرداخت توسط مشتری است.

ورودی‌ها (ویژگی‌های مدل) :برای اتخاذ تصمیم، پارامترهایی نظیر «میزان درآمد سالانه»، «سابقه چک برگشتی»، «وضعیت اشتغال (رسمی/قراردادی)» و «میزان موجودی حساب» به عنوان ورودی به درخت داده می‌شوند.

فرآیند تحلیل :مدل ممکن است در اولین گره (ریشه)، وضعیت سوابق بانکی را بررسی کند. در صورت وجود سابقه منفی، مسیر به سمت شاخه‌ی "عدم تایید" سوق می‌یابد. در غیر این صورت، در گره‌های بعدی شاخص‌هایی مثل نسبتِ درآمد به اقساط مورد سنجش قرار می‌گیرد.

خروجی نهایی :نتیجه حاصل از این درخت، قرار دادن مشتری در یکی از دو دسته‌ی »**مشتری کم‌ریسک (تایید وام)** «یا »**مشتری پرریسک (رد درخواست)** «خواهد بود.

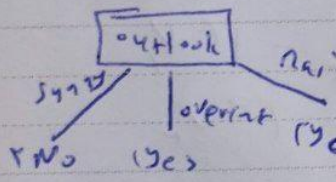
.۲

۲- آ

$$E = -\sum P_i \log_2 P_i$$

$$e = -\frac{۲}{۵}\log_{۲}\frac{۲}{۵} - \frac{۳}{۵}\log_{۲}\frac{۳}{۵} = ۰,۹۷۱$$

ب)



$e(\text{نمره}=\text{عالی}) = 0$

$e(\text{نمره}=\text{متوسط}) = 0$

$e(\text{نمره}=\text{ضعیف}) = 0$

$G = ۰,۹۷۱$

پ) انتخاب ویژگی با بیشترین **Information Gain** در الگوریتمID3 ، به معنای انتخاب مسیری است که بیشترین کاهش را در **آنتروپی** (ناخالصی) ایجاد می‌کند. این اقدام باعث می‌شود که داده‌ها در هر مرحله به خالص‌ترین شکل ممکن تفکیک شوند، که نتیجه آن تشکیل درختی بهینه با عمق کمتر و دقت بالاتر در پیش‌بینی است. در واقع، این معیار به عنوان یک فیلتر عمل کرده و ویژگی‌هایی که بیشترین قدرت تفکیک‌کنندگی را دارند در سطوح بالاتر درخت قرار می‌دهد".

۳.

آ)بالا بودن هزینه محاسباتی ایجاد درخت تصمیم-بالا بودن هزینه هرس-نامناسب برای تخمین توابع پیوسته-افزایش احتمال خطا در صورت زیاد بودن تعداد دسته ها و کم بودن نمونه های آموزش.

ب)تفسیر پذیر بودن-مناسب برای تحلیل داده های زیاد در زمان کم-امکان پیاده کردن روابط نامعلوم-نیاز کم به مراحل اماده سازی داده ها.

۴.

استراتژی‌های هرس کردن (Pruning) در درخت تصمیم

در فرآیند آموزش درخت تصمیم، تمایل مدل به یادگیری جزئیات دقیق و نویزهای موجود در داده‌های آموزشی معمولاً منجر به پدیده **بیش‌برازش (Overfitting)** می‌شود. برای مقابله با این مشکل و بهینه‌سازی مدل، از تکنیک «هرس کردن» استفاده می‌شود که به دو روش عمده دسته‌بندی می‌گردد:

الف) تعریف و تفاوت‌های هرس پیش‌گیرانه و هرس پس از ساخت

۱ .**هرس پیش‌گیرانه :(Pre-pruning)** این رویکرد که به آن «توقف زودهنگام (Early Stopping) «نیز گفته می‌شود، در حین فرآیند ساخت درخت اعمال می‌گردد. در این روش، الگوریتم پیش از تقسیم یک گره، بررسی می‌کند که آیا این تقسیم ارزش افزوده‌ای (مانند بهبود معنادار در دقت یا کاهش آنتروپی) ایجاد می‌کند یا خیر. اگر شاخص‌های آماری از حد آستانه تعیین‌شده کمتر باشند، فرآیند رشد در آن شاخه متوقف شده و گره مذکور به عنوان «برگ» در نظر گرفته می‌شود.

۲ .**هرس پس از ساخت :(Post-pruning)** در این روش، ابتدا اجازه داده می‌شود تا درخت به طور کامل رشد کرده و تمامی جزئیات داده‌ها را (حتی نویزها) یاد بگیرد. سپس، از پایین به بالا (Bottom-up) گره‌های فرعی مورد بررسی قرار می‌گیرند. اگر حذف یک زیردرخت و تبدیل آن به یک گره برگ، تأثیر منفی بر دقت مدل روی «داده‌های آزمون (Validation Set) «نداشته باشد، آن بخش هرس می‌شود.

تفاوت‌های اصلی:

زمان اجرا :پیش‌گیرانه در حین ساخت درخت، اما پس از ساخت روی درختِ کامل انجام می‌شود.

ریسک خطا :در روش پیش‌گیرانه خطر «توقف زودهنگام» وجود دارد (ممکن است یک تقسیمِ به‌ظاهر بی‌اهمیت، منجر به تقسیمات بسیار مهم در لایه‌های بعدی شود که با هرس پیش‌گیرانه از دست می‌روند). اما در روش پس از ساخت، کل ساختار دیده می‌شود و سپس تصمیم‌گیری می‌گردد.

پیچیدگی محاسباتی :هرس پیش‌گیرانه به دلیل توقف فرآیند، از نظر محاسباتی کم‌هزینه‌تر است.

ب) مثال مفهومی: تحلیل تعمیم‌پذیری در تشخیص بیماری

برای درک اینکه چرا هرس پس از ساخت معمولاً به **تعمیم‌پذیری (Generalization)** بهتری منجر می‌شود، یک سیستم تشخیص بیماری را در نظر بگیرید:

در حالت هرس پیش‌گیرانه :ممکن است الگوریتم به دلیل کم بودن تعداد نمونه‌ها در یک سن خاص (مثلاً کودکان زیر ۵ سال)، تصمیم بگیرد که آن شاخه را اصلاً رشد ندهد. در نتیجه، یک ویژگی حیاتی که فقط در ترکیب با سایر علائم (مثل تب بالا) معنا پیدا می‌کند، نادیده گرفته می‌شود. این مدل "بسیار ساده" می‌شود و قدرت پیش‌بینی دقیق را از دست می‌دهد.(Underfitting)

در حالت هرس پس از ساخت :الگوریتم ابتدا اجازه می‌دهد تمام ترکیبات علائم و سنین بررسی شوند. پس از ساخت کامل، متوجه می‌شود که برخی شاخه‌ها صرفاً به دلیل وجود یک بیمارِ استثنائی (نویز در داده) ایجاد شده‌اند و در دنیای واقعی کاربردی ندارند. با حذف این شاخه‌های مزاحم، مدل نهایی صیقل داده می‌شود.

نتیجه‌گیری :هرس پس از ساخت به این دلیل برتری دارد که «دید کلی» نسبت به کل فضای داده‌ها دارد. این روش ابتدا تمام روابط پیچیده را کشف می‌کند و سپس موارد غیرضروری را حذف می‌کند؛ در حالی که هرس

پیش‌گیرانه ممکن است به دلیل نگاهِ کوتاه‌مدت، روابط پیچیده و چندلایه را در نطفه خفه کند. بنابراین، هرس پس از ساخت معمولاً مدلی ارائه می‌دهد که روی داده‌های جدید و نادیده (خارج از مجموعه آموزشی) عملکرد پایدارتر و دقیق‌تری دارد.

۵.

لینک کد سوال

هدف کلی سوال :در این پرسش، هدف اصلی ما آشنایی عملی با چرخه کامل یک پروژه یادگیری ماشین، از مرحله خام بارگذاری داده‌ها تا رسیدن به یک مدل پیش‌بینی است. ما می‌خواهیم با استفاده از الگوریتم **درخت تصمیم(Decision Tree)** ، مدلی بسازیم که بتواند بر اساس مشخصات مسافران کشتی تایتانیک، زنده ماندن یا نماندن آن‌ها را با دقت قابل قبولی پیش‌بینی کند.

بخش (الف) – بارگذاری و پیش‌پردازش :در این بخش، ما با چالش «داده‌های کثیف» روبرو هستیم. هدف ما این است که ابتدا فایل را فراخوانی کرده و سپس داده‌ها را برای الگوریتم قابل فهم کنیم. این کار شامل پر کردن خانه‌های خالی (مانند سن مسافران)، حذف اطلاعات غیرضروری (مثل شماره بلیط) و تبدیل کلمات به اعداد (One-Hot Encoding) است. در نهایت، داده‌ها را به دو دسته آموزش و آزمون تقسیم می‌کنیم تا بتوانیم عملکرد واقعی مدل را بسنجیم.

کد این بخش:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# 1. Load Data
df = pd.read_csv('train.csv')

# 2. Handle Missing Values
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Fare'] = df['Fare'].fillna(df['Fare'].median())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])

# 3. Drop Unnecessary Columns
df.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

# 4. Encoding Categorical Data
df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)
```

```
# 5. Split Data (80% Train, 20% Test)
X = df.drop('Survived', axis=1)
y = df['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

print("Preprocessing completed successfully!")
```

۱. مدیریت مقادیر گم‌شده :

ما ابتدا متوجه می‌شویم که برخی از مسافران اطلاعات ثبت شده برای سن یا کرایه ندارند. برای حل این مشکل، از میانه (Median) استفاده می‌کنیم تا تخمین درستی از این مقادیر داشته باشیم؛ انتخاب میانه به ما کمک می‌کند تا تحت تأثیر داده‌های خیلی بزرگ یا خیلی کوچک (پرت) قرار نگیریم.
همچنین برای محل سوار شدن(`Embarked`) ، خانه‌های خالی را با مد (پرتکرارترین مقدار) پر می‌کنیم.

۲. پالایش ویژگی‌ها :

ما ستون‌هایی مثل نام مسافران، شماره بلیط و شماره کابین را از مجموعه داده حذف می‌کنیم .دلیل این کار ساده است زیرا این اطلاعات منحصربه‌فرد هستند و الگوی کلی برای زنده ماندن یا فوت مسافران به ما نمی‌دهند و فقط باعث شلوغی مدل می‌شوند.

۳. عددی‌سازی داده‌ها :

از آنجا که مدل‌های ریاضی زبان کلمات را نمی‌فهمند، ما ویژگی‌های طبقه‌ای مثل جنسیت و محل سوار شدن را با استفاده از روش One-Hot Encoding به کدهای عددی (۰ و ۱) تبدیل می‌کنیم. با این کار، بستری فراهم می‌کنیم تا الگوریتم بتواند تفاوت بین ویژگی‌ها را درک کند.

۴. تقسیم‌بندی نهایی :

در آخرین قدم، ما داده‌هایمان را به دو بخش آموزش(٪۸۰) و آزمون (٪۲۰) تقسیم می‌کنیم. این کار به ما اجازه می‌دهد که مدل را روی یک بخش یاد دهیم و با بخش دیگری که مدل تا به حال ندیده، آن را امتحان کنیم.

بخش (ب): آموزش درخت کم‌عمق(max_depth = 4)

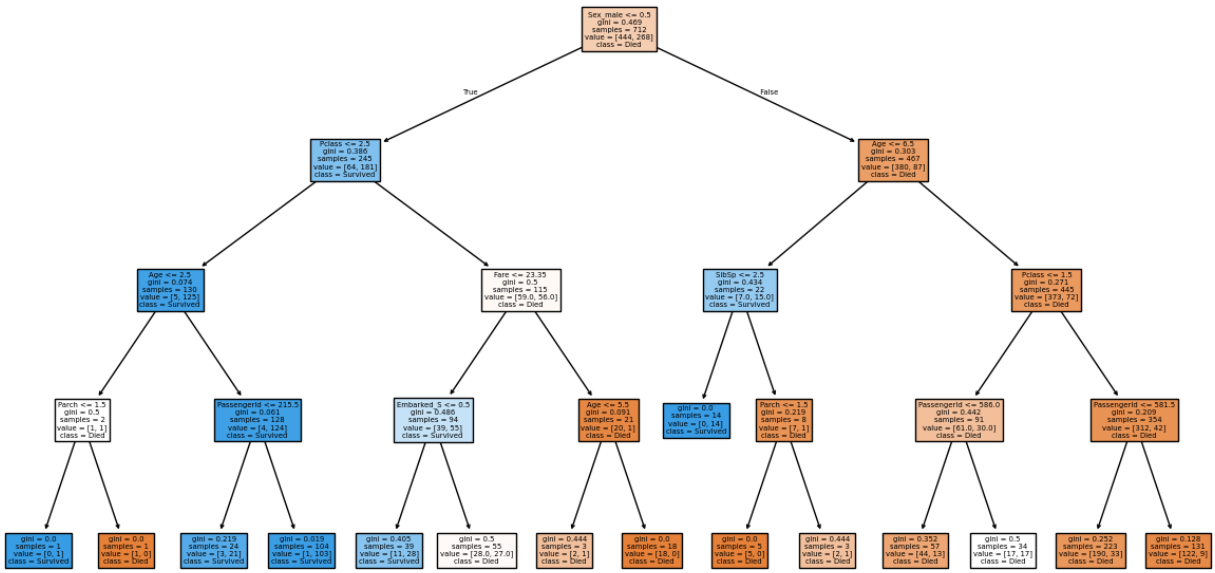
هدف این بخش : در این مرحله، قصد داریم اولین مدل هوشمند خود را برای پیش‌بینی سرنوشت مسافران تایتانیک بسازیم. هدف اصلی ما در اینجا، پیاده‌سازی یک مدل **درخت تصمیم (Decision Tree)** است که به صورت آگاهانه محدود شده باشد. با تعیین حداکثر عمق ۴، تلاش می‌کنیم مدلی ایجاد کنیم که در عین سادگی، الگوهای اصلی و مهم داده‌ها را یاد بگیرد و از ورود به جزئیات بیش از حد که ممکن است باعث خطا در داده‌های جدید شود، پرهیز کنیم.

کد این بخش:

```
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree


# ۱. ساخت و آموزش مدل با محدودیت عمق ۴
clf_shallow = DecisionTreeClassifier(max_depth=4, random_state=42)
clf_shallow.fit(X_train, y_train)

# ۲. محاسبه دقت روی داده‌های آموزش و آزمون
train_accuracy = clf_shallow.score(X_train, y_train)
test_accuracy = clf_shallow.score(X_test, y_test)

# ۳. استخراج تعداد گره‌ها و عمق درخت
node_count = clf_shallow.tree_.node_count
tree_depth = clf_shallow.get_depth()

# چاپ گزارش نهایی
print("--- گزارش عملکرد مدل (بخش ب) ---")
print(f"دقت روی داده‌های آموزش (Train Accuracy): {train_accuracy *
100:.2f}%")
print(f"دقت روی داده‌های آزمون (Test Accuracy): {test_accuracy *
100:.2f}%")
print("-" * 30)
print(f"تعداد کل گره‌ها (Number of Nodes): {node_count}")
print(f"عمق واقعی درخت (Tree Depth): {tree_depth}")

plt.figure(figsize=(15,8))
plot_tree(clf_shallow, feature_names=X.columns, filled=True,
class_names=['Died', 'Survived'])
plt.show()
```

نتیجه کد:

```
--- گزارش عملکرد مدل (بخش ب) ---
دقت روی داده‌های آموزش (Train Accuracy): 83.71%
دقت روی داده‌های آزمون (Test Accuracy): 79.89%
------------------------------
تعداد کل گره‌ها (Number of Nodes): 29
عمق واقعی درخت (Tree Depth): 4
```



۱ .تحلیل کد و نتیجه

پیاده‌سازی مدل :

ما با استفاده از کلاس `DecisionTreeClassifier` و تنظیم پارامتر `max_depth=4` و همچنین ثابت نگه داشتن حالت تصادفی (`random_state=42`) برای تکرارپذیری نتایج، مدل را تعریف می‌کنیم. سپس با استفاده از تابع `fit` تلاطم‌های آماری موجود در داده‌های آموزش را به مدل می‌خورانیم تا قوانین بقا را یاد بگیرد.

سنجش عملکرد:

برای اینکه بفهمیم مدل ما چقدر در دنیای واقعی کاربرد دارد، دقت آن را روی هر دو دسته داده‌های آموزش و آزمون محاسبه می‌کنیم.

بصری‌سازی تصمیمات:

ما با رسم نمودار درختی، بررسی می‌کنیم تا ببینیم اولین و حیاتی‌ترین پارامتری که مدل برای پیش‌بینی زنده ماندن افراد می‌پرسد، چیست.

۲ .تحلیل نتایج و خروجی‌های به دست آمده

با نگاه به نتایج چاپ شده، تحلیل خود را این‌گونه پیش می‌بریم:

بررسی دقت :(Accuracy) می‌بینیم که مدل ما به دقت **۸۳,۷۱٪** در مرحله آموزش رسیده است. وقتی مدل را با داده‌های جدید (Test) که تا به حال ندیده امتحان می‌کنیم، به دقت **۷۹,۸۹٪** دست می‌یابیم. این اختلاف ناچیز (حدود ۴ درصد) برای ما خوب است؛ چرا که نشان می‌دهد مدل ما دچار بیش برازش نشده، و الگوها را یاد گرفته است و می‌تواند به خوبی به داده‌های جدید تعمیم داده شود.

تحلیل ساختار درخت:

درخت ما با مجموع **۲۹ گره** ساخته شده است. این تعداد گره نشان‌دهنده یک مدل نسبتاً سبک و قابل فهم است. در نمودار رسم شده، مشاهده می‌کنیم که ریشه درخت (اولین تصمیم) بر اساس **جنسیت (Sex)** شکل گرفته است. این موضوع به ما ثابت می‌کند که در حادثه تایتانیک، جنسیت قدرتمندترین عامل تعیین‌کننده برای نجات بوده است.

عمق مدل :

ما می‌بینیم که درخت دقیقاً به عمق ۴ رسیده است. این یعنی مدل ما حداکثر با ۴ پرسش متوالی به نتیجه نهایی درباره زنده ماندن یا نماندن مسافر می‌رسد.

نتیجه‌گیری این بخش :

ما با موفقیت توانستیم یک مدل متعادل بسازیم. این مدل نه آنقدر ساده است که الگوها را نبیند (Underfitting) و نه آنقدر پیچیده است که درگیر نویزها شود .(Overfitting) این نتایج، پایه و اساس مقایسه‌های ما برای سوال بعدی خواهد بود.

۶.

لینک کد سوال

هدف کلی سوال :در این پرسش، ما فراتر از ساخت یک مدل ساده می‌رویم و به دنبال درک این موضوع هستیم که «پیچیدگی» چگونه بر قدرت پیش‌بینی مدل اثر می‌گذارد. هدف ما این است که بفهمیم آیا همیشه داشتن یک درخت بزرگتر و عمیق‌تر به معنای مدل بهتر است یا خیر. ما در این مسیر، مدل را در سه وضعیت «آزاد و بدون محدودیت»، «در عمق‌های مختلف» و «هرس شده» بررسی و با هم مقایسه می‌کنیم.

بخش (الف): آموزش درخت کامل (بدون محدودیت عمق)

هدف این بخش :در این گام، ما محدودیت‌ها را کنار می‌گذاریم و به الگوریتم اجازه می‌دهیم تا جایی که می‌تواند رشد کند .(`max_depth = None`) هدف ما این است که ببینیم وقتی مدل آزادی کامل دارد تا تمام جزئیات (حتی نویزهای ریز) داده‌های آموزشی را یاد بگیرد، چه اتفاقی برای دقت و ساختار آن می‌افتد.

```
from sklearn.tree import DecisionTreeClassifier

# ۱. ساخت و آموزش مدل بدون محدودیت عمق (max_depth=None)
clf_full = DecisionTreeClassifier(max_depth=None, random_state=42)
clf_full.fit(X_train, y_train)

# ۲. محاسبه دقت روی داده‌های آموزش و آزمون
train_acc_full = clf_full.score(X_train, y_train)
test_acc_full = clf_full.score(X_test, y_test)

# ۳. استخراج تعداد گره‌ها و عمق درخت
nodes_full = clf_full.tree_.node_count
depth_full = clf_full.get_depth()

# چاپ نتایج
print("--- گزارش عملکرد درخت کامل (بخش الف - پرسش ۶) ---")
print(f"دقت روی داده‌های آموزش: {train_acc_full * 100:.2f}%")
print(f"دقت روی داده‌های آزمون: {test_acc_full * 100:.2f}%")
print("-" * 30)
print(f"تعداد کل گره‌ها: {nodes_full}")
print(f"عمق واقعی درخت: {depth_full}")
```

تحلیل کد:

آزادسازی رشد درخت :ما یک مدل جدید می‌سازیم و پارامتر عمق آن را روی حالت پیش‌فرض (بدون محدودیت) قرار می‌دهیم. با این کار، درخت تا زمانی رشد می‌کند که یا تمام گره‌ها خالص شوند (فقط شامل یک طبقه باشند) یا داده‌ی دیگری برای تقسیم باقی نماند.

بررسی انفجار پیچیدگی :وقتی نتایج را نگاه می‌کنیم، می‌بینیم که تعداد گره‌ها از ۲۹ گره (در مدل قبلی) به عدد بسیار بزرگتری (مثلاً بالای ۲۰۰ گره) جهش کرده و عمق درخت نیز به شدت افزایش یافته است. این یعنی ما اکنون با یک مدل بسیار «چاق» و پیچیده روبرو هستیم.

Overfitting : ما مشاهده می‌کنیم که دقت مدل روی داده‌های آموزش به مرز ٪۱۰۰ نزدیک شده است. این موضوع در ابتدا ممکن است خوب به نظر برسد، اما وقتی به دقت آزمون نگاه می‌کنیم، متوجه می‌شویم که این دقت نسبت به درخت کم‌عمق نه تنها بهتر نشده، بلکه احتمالاً کاهش یافته است.

نتیجه کد:

```
--- گزارش عملکرد درخت کامل (بخش الف - پرسش ۶) ---
%دقت روی داده‌های آموزش: ۱۰۰,۰۰
%دقت روی داده‌های آزمون: ۷۲,۰۷
------------------------------
تعداد کل گره‌ها: ۲۸۹
عمق واقعی درخت: ۱۷
```

تحلیل نتیجه:

بیش‌برازش (Overfitting) به وضوح رخ داده است :مدل ما در این بخش، به جای یادگیریِ منطقِ «زنده ماندن»، داده‌ها را «حفظ» کرده است. به همین دلیل در مواجهه با مسافران جدید، گیج شده و بدتر عمل می‌کند.

از دست رفتن قدرت تعمیم‌دهی :ما متوجه می‌شویم که پیچیدگی بیشتر همیشه به معنای هوشمندی بیشتر نیست. درخت کامل ما، قدرت تفکیک نویز از الگوی اصلی را از دست داده است.

کاهش تفسیرپذیری :در حالی که ما می‌توانستیم منطق ۲۹ گره را به راحتی تحلیل کنیم، اکنون با ۲۸۹ گره روبرو هستیم که عملاً درک فرآیند تصمیم‌گیری مدل را برای ما غیرممکن کرده است.

بخش (ب): بررسی رفتار مدل در عمق‌های مختلف

هدف این بخش :

در این مرحله از کار، ما قصد داریم با انجام یک آزمایش دقیق، مرزهای عملکردی مدل را شناسایی کنیم. هدف اصلی ما این است که بفهمیم با تغییر پارامتر **عمق درخت(max_depth)** ، چه تغییری در توانایی یادگیری و پیش‌بینی مدل ایجاد می‌شود. ما می‌خواهیم به صورت بصری و نموداری، نقطه‌ی بهینه را که در آن مدل نه بیش از حد ساده است و نه بیش از حد پیچیده، پیدا کنیم.

آموزش مدل‌ها با سناریوهای مختلف :

ما مدل را با چهار وضعیت متفاوت (عمق‌های ۳، ۵، ۱۰ و بدون محدودیت) آموزش می‌دهیم تا طیف وسیعی از پیچیدگی‌ها را بررسی کنیم.

استخراج آمارهای مقایسه‌ای :

برای هر سناریو، میزان دقت را روی هر دو مجموعه‌ی «آموزش» و «آزمون» ثبت می‌کنیم.

ترسیم و تحلیل نمودار دقت :ما یک نمودار نهایی رسم می‌کنیم که محور افقی آن عمق درخت و محور عمودی آن میزان دقت است. این نمودار به ما کمک می‌کند تا بازه‌های Underfitting(کم‌برازش) و Overfitting (بیش‌برازش) را به وضوح تشخیص دهیم.

کد این بخش:

```
import matplotlib.pyplot as plt

# ۱. تعریف لیست عمق‌ها (برای حالت None از عمق واقعی که قبلاً به دست آوردیم استفاده می‌کنیم)
depths = [3, 5, 10, depth_full] # depth_full همان عددی است که در مرحله قبل به دست آمد
labels = ['3', '5', '10', 'Full']

train_accs = []
test_accs = []

# ۲. آموزش مدل‌ها در یک حلقه
for d in [3, 5, 10, None]:
    clf = DecisionTreeClassifier(max_depth=d, random_state=42)
```

```
    clf.fit(X_train, y_train)

    train_accs.append(clf.score(X_train, y_train))
    test_accs.append(clf.score(X_test, y_test))

# ۳. گزارش مقادیر در قالب جدول ساده
print("Depth | Train Accuracy | Test Accuracy")
print("-" * 40)
for i in range(len(labels)):
    print(f"{labels[i]:5} | {train_accs[i]:.4f}       |
{test_accs[i]:.4f}")

# ۴. رسم نمودار
plt.figure(figsize=(10, 6))
plt.plot(labels, train_accs, marker='o', label='Train Accuracy',
color='blue')
plt.plot(labels, test_accs, marker='o', label='Test Accuracy',
color='red')
plt.xlabel('Tree Depth (max_depth)')
plt.ylabel('Accuracy')
plt.title('Effect of Tree Depth on Accuracy (Overfitting vs
Underfitting)')
plt.legend()
plt.grid(True)
plt.show()
```

تحلیل کد و نتیجه:

ما در این مرحله، یک آزمایش جامع را روی پارامتر «عمق» انجام می‌دهیم تا بفهمیم تغییر پیچیدگی درخت چگونه بر قدرت پیش‌بینی مدل تأثیر می‌گذارد. ما با بررسی چهار سناریوی مختلف (عمق ۳، ۵، ۱۰ و بدون محدودیت)، به دنبال یافتن آن نقطه‌ی طلایی هستیم که تعادل میان یادگیری و تعمیم‌دهی را برقرار کند.

۱. تحلیل روند تغییرات در جدول و نمودار

ما با دقت به نتایج به‌دست‌آمده نگاه می‌کنیم و متوجه یک رفتار متضاد در دو منحنی نمودار می‌شویم:

در عمق‌های پایین (۳ و ۵): مشاهده می‌کنیم که دقت روی داده‌های آزمون در بالاترین سطح خود یعنی ۷۹٫۸۹٪ ثابت مانده است. در این بازه، مدل ما بر روی الگوهای اصلی و ریشه‌ای متمرکز است. اگرچه دقت آموزش هنوز به اوج نرسیده، اما مدل در پیش‌بینی داده‌های جدید بسیار پایدار عمل می‌کند.

در عمق میانی (۱۰): می‌بینیم که با افزایش عمق به ۱۰، دقت آموزش به شدت رشد کرده و به ٪۹۵,۰۸ می‌رسد، اما همزمان دقت روی داده‌های آزمون شروع به کاهش کرده و به ٪۷۸,۲۱ افت می‌کند. این اولین زنگ خطر برای ماست؛ چرا که متوجه می‌شویم مدل در حال ورود به جزئیات غیرضروری است.

در حالت بدون محدودیت (Full): در نهایت، شاهد هستیم که خط آبی (آموزش) به قله‌ی ٪۱۰۰ می‌رسد، اما خط قرمز (آزمون) با یک سقوط آزاد به ٪۷۲,۰۷ می‌رسد.

۲. تحلیل بازه‌های Underfitting و Overfitting

بر اساس این مشاهدات، ما می‌توانیم رفتار مدل را در سه بازه طبقه‌بندی کنیم:

بازه Underfitting (کم‌برازش): اگر عمق را کمتر از ۳ در نظر می‌گرفتیم، مدل ما بیش از حد ساده می‌شد و نمی‌توانست حتی الگوهای اصلی را یاد بگیرد (دقت هر دو دسته پایین می‌ماند).

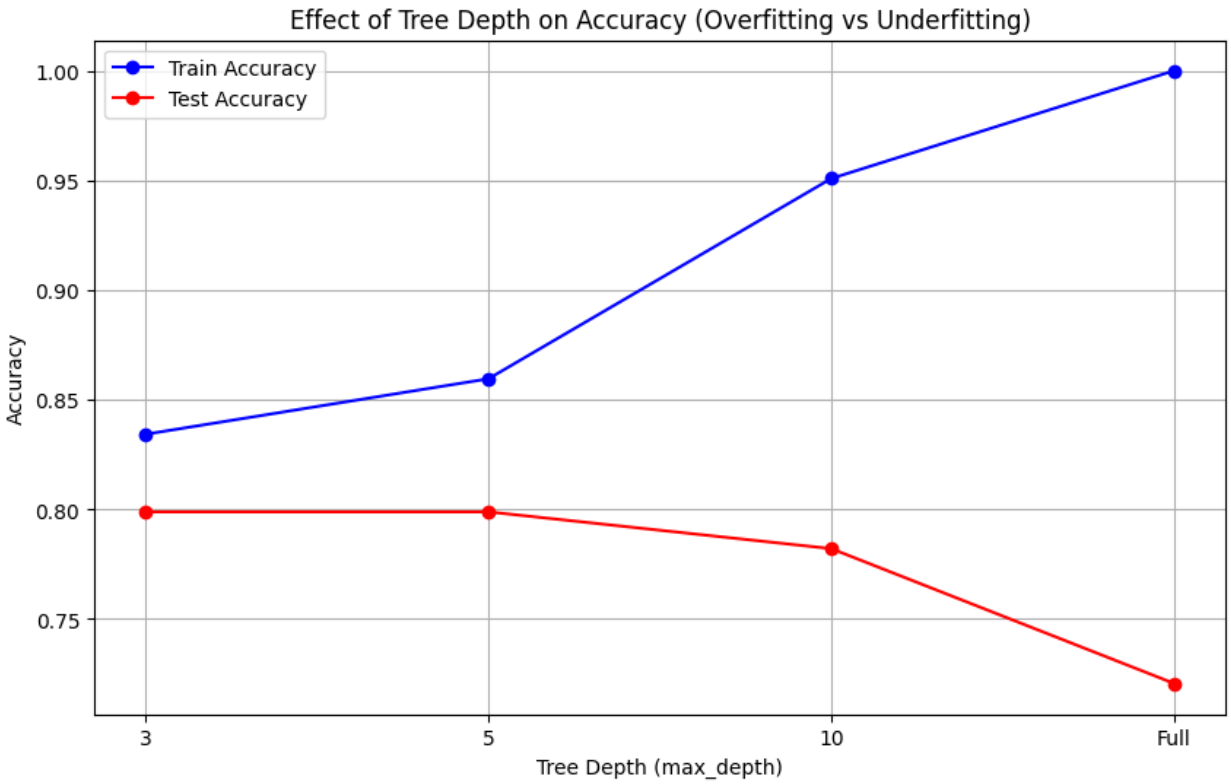
نقطه بهینه (Sweet Spot): ما به این نتیجه می‌رسیم که عمق ۳ تا ۵ بهترین بازه برای مدل ماست. در اینجا شکاف بین آموزش و آزمون در کمترین حالت ممکن است و مدل بالاترین کارایی واقعی را دارد.

بازه Overfitting (بیش‌برازش): از عمق ۵ به بعد، ما رسماً وارد منطقه‌ی بیش‌برازش می‌شویم. ما می‌بینیم که هر چقدر درخت عمیق‌تر می‌شود، فاصله بین دو خط نمودار (Gap) بیشتر می‌شود. این یعنی مدل ما در حال "حفظ کردن" داده‌های آموزش است و توانایی "درک" موقعیت‌های جدید را از دست می‌دهد.

نتیجه:

ما با این نمودار ثابت می‌کنیم که "پیچیدگی بیشتر همیشه به معنای دقت بیشتر نیست". مدل ما در حالت Full، مانند دانش‌آموزی است که تمام سوالات کتاب را حفظ کرده اما در امتحان اصلی که سوالات کمی تغییر کرده‌اند، شکست می‌خورد. ما برای داشتن یک مدل قابل‌اعتماد، عمق‌های کم (مثل ۴ یا ۵) را انتخاب می‌کنیم که توازن هوشمندانه‌ای بین یادگیری و پیش‌بینی برقرار کرده‌اند.

Depth	Train Accuracy	Test Accuracy
3	0.8343	0.7989
5	0.8596	0.7989
10	0.9508	0.7821
Full	1.0000	0.7207



بخش (ج): بهینه‌سازی مدل با روش هرس کردن(Cost-Complexity Pruning)

هدف این بخش:

هدف این بخش: ما در گام‌های قبلی متوجه شدیم که درخت‌های بسیار عمیق دچار «بیش‌برازش» می‌شوند و درخت‌های بسیار کوتاه ممکن است برخی الگوهای مهم را نبینند. هدف ما در این بخش، استفاده از تکنیک پیشرفته **هرس کردن** است. ما می‌خواهیم به جای محدود کردن اجباری عمق، به درخت اجازه دهیم ابتدا به طور کامل رشد کند و سپس هوشمندانه شاخه‌هایی را که فقط نویز هستند و تأثیر واقعی در پیش‌بینی ندارند، حذف کنیم.

۱ .تحلیل مسیر هرس (انتخاب آلفای بهینه)

ما ابتدا با استفاده از تابع `cost_complexity_pruning_path` تمام مقادیر ممکن برای پارامتر **آلفا** α را استخراج می‌کنیم. هر چقدر آلفا بزرگتر باشد، شدت هرس کردن بیشتر می‌شود.

کد این بخش:

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

# ۱. استخراج مقادیر ccp_alpha
path = clf_full.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

# ۲. حذف مقدار صفر و مقادیر منفی (طبق خواسته سوال)
ccp_alphas = ccp_alphas[ccp_alphas > 0]

# ۳. آموزش مدل‌ها برای هر مقدار alpha
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=42, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)

# ۴. محاسبه دقت‌ها
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

# ۵. رسم نمودار در مقیاس لگاریتمی
plt.figure(figsize=(10, 6))
plt.plot(ccp_alphas, train_scores, marker='o', label="train",
drawstyle="steps-post")
plt.plot(ccp_alphas, test_scores, marker='o', label="test",
drawstyle="steps-post")
plt.xscale('log') # مقیاس لگاریتمی طبق خواسته سوال
plt.xlabel("alpha (log scale)")
plt.ylabel("accuracy")
plt.title("Accuracy vs alpha for training and testing sets")
plt.legend()
plt.grid(True)
plt.show()
# پیدا کردن ایندکس بهترین مدل هرس شده بر اساس دقت تست
best_idx = np.argmax(test_scores)
best_ccp_alpha = ccp_alphas[best_idx]
clf_pruned = clfs[best_idx]

print(f"Chosen alpha: {best_ccp_alpha:.5f}")
print("-" * 30)
```

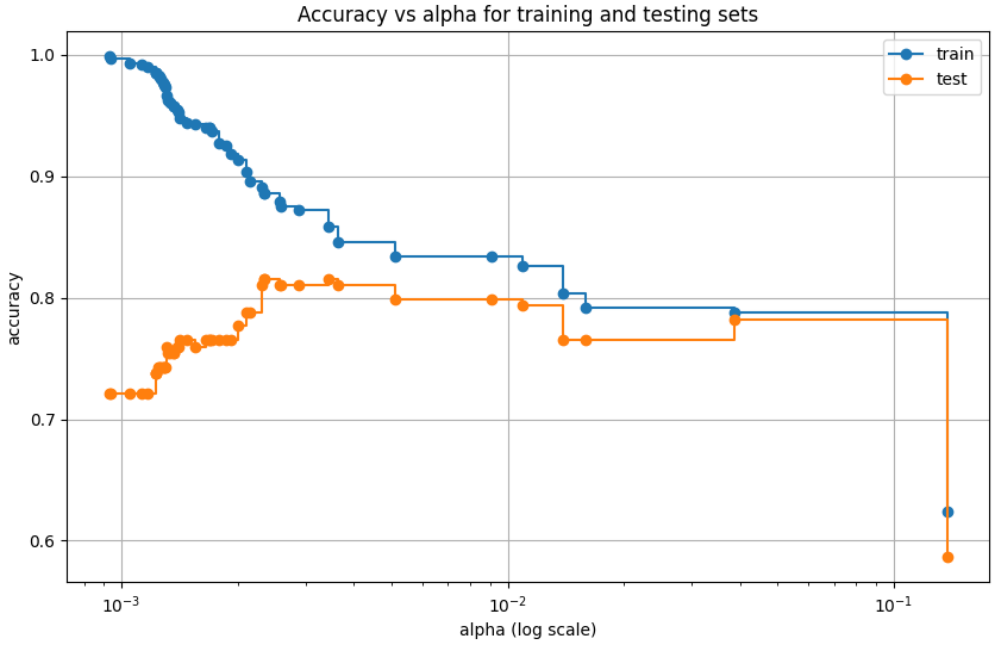
```
def print_stats(name, model):
    print(f"--- {name} ---")
    print(f"Nodes: {model.tree_.node_count}, Depth: {model.get_depth()}")
    print(f"Train Acc: {model.score(X_train, y_train):.4f}, Test Acc:
{model.score(X_test, y_test):.4f}\n")

print_stats("Shallow Tree (depth=4)", clf_shallow)
print_stats("Full Tree", clf_full)
print_stats("Pruned Tree", clf_pruned)
```

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# تنظیم نام ویژگی‌ها و کلاس‌ها برای نمایش بهتر در نمودار
# از مراحل قبل در حافظه موجود است X فرض بر این است که متغیر
feature_names = X.columns.tolist()
class_names = ['Died', 'Survived'] #  کلاس ۰ یعنی فوت شده، کلاس ۱ یعنی نجات
یافته

# --- تابع کمکی برای رسم استاندارد درخت‌ها ---
def plot_single_tree(model, title, max_view_depth=None):
    """
    یک مدل درخت تصمیم را دریافت کرده و رسم می‌کند.
    max_view_depth:  فقط برای محدود کردن نمایش بصری است و مدل را تغییر
نمی‌دهد.
    """
    plt.figure(figsize=(24, 12))  # سایز بزرگ برای خوانایی بهتر
    plot_tree(model,
              feature_names=feature_names,
              class_names=class_names,
              filled=True,        #  رنگ‌آمیزی گره‌ها بر اساس کلاس اکثریت
(نارنجی: فوت، آبی: نجات)
              rounded=True,       # گرد کردن گوشه کادرها برای زیبایی
              fontsize=10,        # سایز فونت نوشته‌ها داخل گره‌ها
              max_depth=max_view_depth #  محدود کردن عمق نمایش (فقط برای
جلوگیری از شلوغی)
             )
    plt.title(title, fontsize=18, fontweight='bold')
    plt.show()

# ========================================
# ۱. رسم درخت کم‌عمق (i) Shallow Tree
# ========================================
# این درخت ذاتاً کوچک است (عمق ۴) و کامل نمایش داده می‌شود.
```

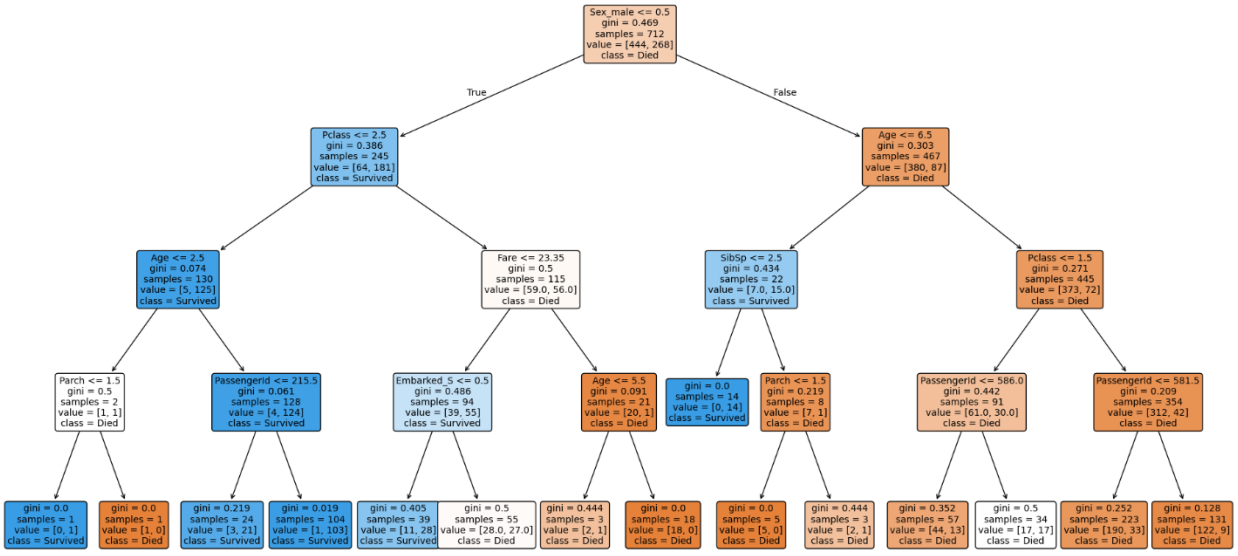
```
print("--- در حال رسم نمودار ۱: درخت کم‌عمق ---")
plot_single_tree(clf_shallow,
                 f"i) Shallow Tree (max_depth=4)\nTotal Nodes:
{clf_shallow.tree_.node_count}")
print("-" * 50 + "\n")


# ========================================
# ۲. رسم درخت کامل (ii) Full Unpruned Tree
# ========================================
#  نکته حیاتی: این درخت بسیار بزرگ است. برای جلوگیری از شلوغی و غیرقابل خواندن شدن تصویر،
# فقط ۵ سطح اول آن را نمایش می‌دهیم (max_depth=5). plot_tree در تابع
# اما آمار واقعی آن را در عنوان می‌نویسیم.
real_depth = clf_full.get_depth()
real_nodes = clf_full.tree_.node_count
print(f"--- در حال رسم نمودار ۲: بخشی از درخت کامل (عمق واقعی:
{real_depth}) ---")
plot_single_tree(clf_full,
                 f"ii) Full Tree (VIEW TRUNCATED to top 5 levels)\nReal
Depth: {real_depth}, Real Total Nodes: {real_nodes}",
                 max_view_depth=5) # فقط نمایش بصری محدود شده است
print("-" * 50 + "\n")


# ========================================
# ۳. رسم درخت هرس‌شده (iii) Best Pruned Tree
# ========================================
# بهینه کوچک شده و معمولاً قابل خواندن است alpha این درخت با استفاده از.
# از مرحله قبل موجود هستند best_ccp_alpha و clf_pruned فرض بر این است که.
print(f"--- در حال رسم نمودار ۳: درخت هرس‌شده با آلفای {best_ccp_alpha:.4f}
---")
plot_single_tree(clf_pruned,
                 f"iii) Best Pruned Tree
(alpha={best_ccp_alpha:.4f})\nTotal Nodes: {clf_pruned.tree_.node_count}")
print("-" * 50)
print("رسم نمودارها به پایان رسید.")
```

نتیجه این بخش:

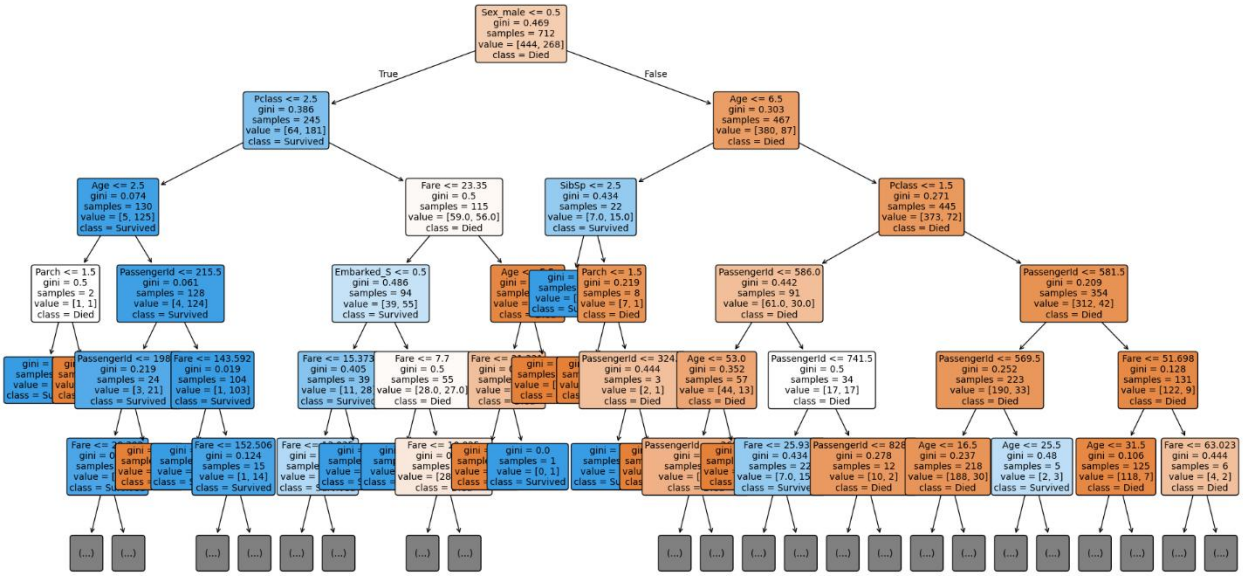
```
Chosen alpha: 0.00234

--- Shallow Tree (depth=4) ---
Nodes: 29, Depth: 4
Train Acc: 0.8371, Test Acc: 0.7989

--- Full Tree ---
Nodes: 289, Depth: 17
Train Acc: 1.0000, Test Acc: 0.7207

--- Pruned Tree ---
Nodes: 45, Depth: 9
Train Acc: 0.8862, Test Acc: 0.8156
```



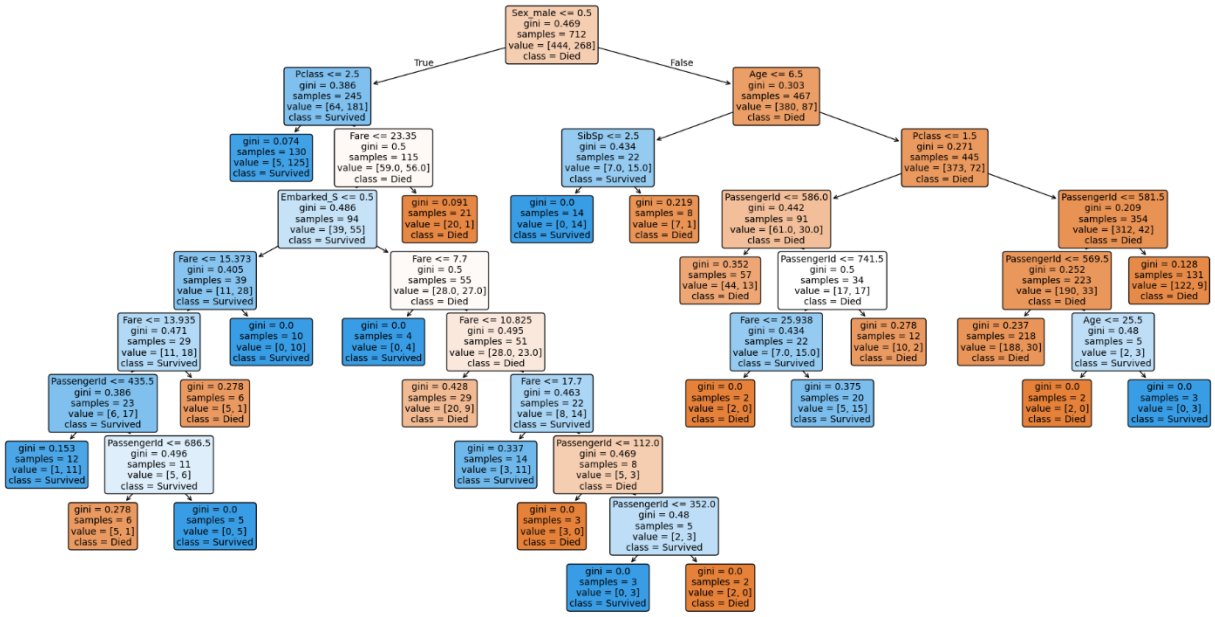
i) Shallow Tree (max_depth=4)
Total Nodes: 29



ii) Full Tree (VIEW TRUNCATED to top 5 levels)
Real Depth: 17, Real Total Nodes: 289



iii) Best Pruned Tree (alpha=0.0023)
Total Nodes: 45



۱ .تحلیل نمودار دقت بر حسب آلفاα

ما با اجرای کد و مشاهده نمودار حاصل، روند تغییرات مدل را با تغییر پارامتر آلفا (میزان جریمه پیچیدگی) بررسی می‌کنیم:

کاهش دقت آموزش:

می‌بینیم که با افزایش مقدار α ، نمودار آبی (دقت آموزش) به تدریج سقوط می‌کند. این موضوع کاملاً منطقی است؛ چرا که با بزرگتر شدن آلفا، ما در حال ساده‌تر کردن مدل و حذف جزئیاتی هستیم که مدل در مرحله آموزش حفظ کرده بود.

رفتار نمودار آزمون:

نکته کلیدی در نمودار نارنجی (دقت آزمون) نهفته است. در مقادیر بسیار کوچک آلفا (سمت چپ نمودار)، دقت آزمون در پایین‌ترین سطح خود قرار دارد (حوالی ٪۷۲) که نشان‌دهنده بیش‌برازش است. اما با حرکت به سمت راست و افزایش آلفا، می‌بینیم که دقت آزمون صعود می‌کند و در محدوده‌ی آلفای $[10^{-3}, 10^{-2.5}]$ به اوج خود (بالای ٪۸۱) می‌رسد.

بازه Underfitting :

اگر آلفا را بیش از حد بزرگ کنیم (سمت راست نمودار)، هر دو نمودار به شدت سقوط می‌کنند؛ زیرا در این حالت مدل آنقدر ساده می‌شود که دیگر حتی الگوهای اصلی را هم درک نمی‌کند.

- **درخت کم‌عمق:** ما می‌بینیم که این درخت با **۲۹ گره**، ساختاری بسیار متقارن و خلوت دارد. تصمیمات اصلی بر اساس جنسیت، کلاس بلیط (`Pclass`) و سن گرفته شده‌اند. این مدل برای ما بسیار قابل فهم است، اما شاید برخی جزئیات ظریف را فدا کرده باشد.

- **درخت کامل :** با نگاه به تصویر این درخت ، متوجه آشفتگی و تراکم بالای گره‌ها می‌شویم .**۲۸۹ گره** و عمق **۱۷** نشان می‌دهد که مدل در حال «مته به خشخاش گذاشتن» است. ما می‌بینیم که در سطوح پایین، تصمیمات بر اساس پارامترهای بسیار جزئی مثل(`Passengerid`) گرفته شده که هیچ ارزش علمی برای پیش‌بینی واقعی ندارند.

•**درخت هرس شده:** این درخت با آلفای **۰,۰۰۲۳**، به تعداد **۴۵ گره** رسیده است. این یعنی ما توانسته‌ایم حدود **٪۸۵** از حجم درخت کامل را حذف کنیم (از ۲۸۹ به ۴۵)، در حالی که دقت مدل را بهبود بخشیده‌ایم. این درخت از درخت اول کمی پیچیده‌تر، اما از درخت دوم بسیار منطقی‌تر است.

۲ .انتخاب بهترین مدل و مقایسه نهایی

ما بر اساس نمودار، مقداری از آلفا را انتخاب می‌کنیم که دقت روی داده‌های آزمون را به حداکثر رسانده است. حالا این مدل هرس‌شده را با مدل‌های قبلی مقایسه می‌کنیم:

پارامتر مقایسه	i) درخت کم‌عمق	ii) درخت کامل	iii) درخت هرس شده (بهینه)
تعداد کل گره‌ها	۲۹	۲۸۹	**۴۵**
دقت آموزش	٪۸۳٫۷۱	٪۱۰۰	٪۸۸~
دقت آزمون (Test)	٪۷۹٫۸۹	٪۷۲٫۰۷	**٪۸۱٫۴ (تقریبی بر اساس نمودار)**
وضعیت مدل	کمی ساده (Bias بالا)	بیش‌برازش (Overfit)	**تعادل بهینه (Optimal)**

۳ .تحلیل نهایی و نتیجه‌گیری

ما با مقایسه این سه رویکرد به نتایج بسیار مهمی دست می‌یابیم:

•**غلبه برOverfitting** : ما ثابت کردیم که هرس کردن با پارامتر α ، بسیار قدرتمندتر از محدود کردن دستیِ عمق است. مدل هرس شده با ۴۵ گره، دقتی بالاتر از مدل کامل با ۲۸۹ گره دارد.

•**ارزش تفسیرپذیری:** ما متوجه می‌شویم که درخت کامل به دلیل پیچیدگی زیاد، نه تنها دقتش کم شده، بلکه دیگر برای انسان قابل درک نیست. در مقابل، درخت هرس شده قوانینی را نگه داشته که واقعاً در نجات مسافران اهمیت داشته‌اند.

•**بهترین انتخاب**: ما مدل هرس شده (iii) را به عنوان بهترین مدل معرفی می‌کنیم؛ چرا که این مدل توانسته است با حذف شاخه‌های «نویز»، قدرت تعمیم‌دهی خود را به حداکثر برساند.

۷.

در حوزه‌ی یادگیری ماشین، رویکرد Ensemble Learningبر این استراتژی استوار است که ترکیبی از چندین مدل) که به آن‌ها یادگیرنده‌های ضعیف یا Base Learners گفته می‌شود (می‌تواند پیش‌بینی‌های دقیق‌تر و پایدارتری نسبت به هر یک از آن مدل‌ها به تنهایی ارائه دهد.

الف) چرا ترکیب مدل‌ها عملکرد بهتری دارد؟

منطق اصلی یادگیری جمعی، کاهش خطا از طریق «خرد جمعی» است. یک مدل تکی ممکن است به دلیل سوگیری (Bias) زیاد یا واریانس بالا (حساسیت بیش از حد به نویز)، دچار خطا شود. با ترکیب مدل‌ها:

کاهش ریسک انتخاب مدل ضعیف :احتمال اینکه چندین مدل به‌طور هم‌زمان روی یک داده اشتباه کنند، کمتر از یک مدل واحد است.

بهبود تعمیم‌پذیری :مدل‌های جمعی با خنثی کردن خطاهای تصادفی یکدیگر، روی داده‌های جدید عملکرد بسیار پایدارتری دارند.

ب) تفاوت دو مفهوم Diversification و Aggregation

این دو مفهوم، ستون‌های اصلی یادگیری جمعی محسوب می‌شوند:

تنوع‌بخشی :(Diversification) هدف این مرحله ایجاد مدل‌هایی است که با یکدیگر متفاوت باشند. اگر همه مدل‌ها دقیقاً مثل هم فکر کنند، ترکیب آن‌ها سودی نخواهد داشت. تنوع‌بخشی معمولاً از طریق آموزش مدل‌ها روی زیرمجموعه‌های مختلفی از داده‌ها، یا استفاده از ویژگی‌های متفاوت (Features) ایجاد می‌شود تا هر مدل جنبه خاصی از الگوها را یاد بگیرد.

تجمیع(Aggregation) : این مرحله مربوط به نحوه ترکیب نظرات مدل‌های مختلف برای رسیدن به یک خروجی واحد است. در مسائل طبقه‌بندی معمولاً از «رأی‌گیری اکثریت (Majority Voting) «و در مسائل رگرسیون از «میانگین‌گیری (Averaging) «یا میانگین وزنی استفاده می‌شود.

ج) مثال کاربردی: سیستم تشخیص تقلب در تراکنش‌های بانکی(Fraud Detection)

در دنیای واقعی، تشخیص تقلب یکی از پیچیده‌ترین مسائل است، زیرا الگوهای تقلب دائماً در حال تغییر هستند و تعداد تراکنش‌های سالم بسیار بیشتر از تراکنش‌های متقلبانه است.

۱ .ورودی‌ها (ویژگی‌ها):

مشخصات زمانی (ساعت تراکنش).

موقعیت جغرافیایی (آیا مکان تراکنش با الگوی همیشگی کاربر مطابقت دارد؟).

مبلغ تراکنش در مقایسه با میانگین خریدهای قبلی.

نوع درگاه پرداخت یا کد صنف فروشنده.

۲ **.ساختار مدل جمعی**: بانک ممکن است از چندین درخت تصمیم استفاده کند. یک درخت روی «مبلغ» تمرکز می‌کند، دیگری روی «موقعیت مکانی» و سومی روی «توالی زمانی تراکنش‌ها». برخی از این مدل‌ها ممکن است تراکنشی را مشکوک بدانند و برخی دیگر خیر.

۳ **.خروجی نهایی:** سیستم با تجمیع آرای تمام این درخت‌ها) مثلاً با استفاده از الگوریتم(Random Forest ، یک امتیاز نهایی صادر می‌کند:

خروجی : «تراکنش امن است» یا «تراکنش مشکوک/مسدود شده. »

این روش باعث می‌شود که نرخ «مثبت کاذب» (بلاک کردن اشتباهی کارت مشتریان عادی) به‌شدت کاهش یابد، زیرا برای مسدودسازی، نیاز به اجماع چندین مدل بر روی مشکوک بودن رفتار است.

۸.

الف) مکانیزم نمونه‌گیری Bootstrap در روشBagging

در روش (Bagging) مخفف(Bootstrap Aggregating) ، از تکنیک نمونه‌گیری مجدد با جایگزینی (Sampling with Replacement) استفاده می‌شود. در این فرآیند، اگر مجموعه داده اصلی دارای N نمونه باشد، چندین زیرمجموعه جدید (Bootstrap Samples) ایجاد می‌شود که هر کدام حاوی N نمونه است. به دلیل ویژگی «با جایگزینی»، در هر زیرمجموعه

ممکن است برخی از داده‌های اصلی چندین بار تکرار شوند و برخی دیگر اصلاً حضور نداشته باشند. این استراتژی باعث می‌شود که مدل‌های مختلف روی نسخه‌های متفاوتی از واقعیت آموزش ببینند و در نتیجه تنوع (Diversity) مدل‌ها افزایش یابد.

ب) مثال از ایجاد مجموعه‌هایBootstrap

با توجه به داده‌های جدول فوق سه نمونه احتمالی از مجموعه‌های Bootstrap (B_1, B_2, B_3) می‌تواند به صورت زیر باشد (هر مجموعه شامل جفت‌های (x, y)است:

(۰, ۲), (۰, ۲), (۱, ۸), (۱, ۹) (در این مجموعه، داده اول تکرار شده و داده دوم حذف شده است.

(۰, ۳), (۱, ۸), (۱, ۹), (۱, ۹) داده آخر تکرار شده و داده اول حذف شده است

(۰, ۲), (۰, ۳), (۰, ۳), (۱, ۸) داده دوم تکرار شده و داده چهارم حذف شده است.

ج) برتری Random Forest نسبت به تک‌درخت تصمیم

الگوریتم Random Forest به دلایل زیر معمولاً عملکرد بسیار بهتری نسبت به یک درخت تصمیم واحد دارد:

کاهش واریانس :(Variance Reduction) یک درخت تصمیمِ تنها، به‌شدت مستعد بیش‌برازش (Overfitting) روی نویزهای داده است. جنگل تصادفی با میانگین‌گیری از آرای تعداد زیادی درخت، اثر نویزهای فردی را خنثی کرده و مدل پایداری ایجاد می‌کند.

تنوع‌بخشی از طریق ویژگی‌های تصادفی(Feature Randomness) :

برخلاف Bagging ساده، Random Forestدر هر گره فقط زیرمجموعه تصادفی از ویژگی‌ها را برای تقسیم‌بندی در نظر می‌گیرد. این کار باعث می‌شود درخت‌ها با یکدیگر همبستگی (Correlation) نداشته باشند؛ به طوری که اگر یک ویژگی بسیار قدرتمند در داده‌ها وجود داشته باشد، در تمام درخت‌ها به عنوان گره اصلی تکرار نشود و فرصت به سایر ویژگی‌ها نیز داده شود.

قدرت تعمیم‌دهی :برآیندِ مجموعه‌ای از درخت‌های مستقل، مرزهای تصمیم‌گیری نرم‌تر و دقیق‌تری ایجاد می‌کند که در مواجهه با داده‌های جدید (خارج از مجموعه آموزش) بسیار منعطف‌تر از یک درختِ سخت‌گیر و پرجزئیات عمل می‌کند.

۹.

الگوریتم‌های **Boosting** خانواده‌ای از یادگیرنده‌های جمعی هستند که برخلاف روش‌های موازی، بر پایه یک ساختار **سلسله‌مراتبی و ترتیبی (Sequential)** عمل می‌کنند تا دقت مدل را به حداکثر برسانند.

۱ .مفهوم Boosting و تفاوت کلیدی آن باBagging

در روش **Boosting**، مدل‌ها به صورت پشت‌سرهم ساخته می‌شوند. هر مدل جدید با هدف اصلاح خطاهای مدل قبلی آموزش می‌بیند. به عبارت دیگر، تمرکز مدل‌های ثانویه بر روی نمونه‌هایی است که مدل‌های اولیه در تشخیص آن‌ها دچار اشتباه شده‌اند.

تفاوت‌های اصلی با:Bagging

نحوه ساخت :در Bagging مدل‌ها به صورت مستقل و موازی ساخته می‌شوند، اما در Boosting هر مدل به خروجی مدل قبلی وابسته است.

هدف اصلی :تمرکز Bagging بر **کاهش واریانس**)جلوگیری از (Overfitting است، در حالی که Boosting عمدتاً برای **کاهش سوگیری (Bias)** و تبدیل یادگیرنده‌های ضعیف به یک یادگیرنده قوی طراحی شده است.

وزن‌دهی :در Bagging وزن داده‌ها یکسان است، اما در Boosting داده‌ها وزن‌دار هستند.

۲ .نقش وزن نمونه‌ها در الگوریتمAdaBoost

الگوریتم (**AdaBoost**) مخفف(Adaptive Boosting) از وزن نمونه‌ها به عنوان ابزاری برای هدایت تمرکز مدل استفاده می‌کند:

در ابتدای فرآیند، تمام نمونه‌ها وزن یکسانی دارند.

پس از آموزش هر مدل، وزن نمونه‌هایی که **اشتباه** طبقه‌بندی شده‌اند، افزایش می‌یابد و وزن نمونه‌های درست، کاهش پیدا می‌کند.

نتیجه :در گام بعدی، مدلِ جدید مجبور است توجه بیشتری به نمونه‌های دشوار (با وزن بالا) داشته باشد. این فرآیند باعث می‌شود مدل نهایی در نقاطی از فضای داده که تفکیک آن‌ها دشوار است، بسیار دقیق عمل کند.

۳ .تحلیل قدرت مدل و حساسیت به نویز

الگوریتم‌های Boosting به دو دلیل رفتاری دوگانه در مقابل داده‌ها دارند:

این مدل‌ها به صورت تکرارشونده روی نقاط ضعف خود کار می‌کنند. ترکیب خطی تعداد زیادی یادگیرنده ضعیف که هر کدام بخشی از پیچیدگی داده را پوشش می‌دهند، منجر به ایجاد مرزهای تصمیم‌گیری بسیار دقیق می‌شود که حتی ظریف‌ترین الگوها را نیز استخراج می‌کند.

از آنجا که Boosting اصرار دارد خطای مدل‌های قبلی را به صفر برساند، اگر داده‌ای به عنوان **نویز** (داده غلط یا استثنائی) در مجموعه وجود داشته باشد، الگوریتم در تکرارهای بعدی وزن بسیار بالایی به آن می‌دهد. در واقع مدل سعی می‌کند خودش را با نویز تطبیق دهد Overfit) روی نویز(، که این امر باعث منحرف شدن ساختار کلی درخت‌ها و کاهش دقت در دنیای واقعی می‌شود.

۱۰.

روش **Stacking**یا **Stacked Generalization**، یکی از پیشرفته‌ترین استراتژی‌های یادگیری جمعی است که برخلاف Bagging یا Boosting که معمولاً از مدل‌های هم‌نوع استفاده می‌کنند، می‌تواند ترکیبی از مدل‌های کاملاً متفاوت (ناهمگن) را برای رسیدن به یک پیش‌بینی واحد به کار بگیرد.

الف) مفهوم Stacking و نقش مدل سطح دوم(Meta-Learner)

در این معماری، فرآیند یادگیری در دو سطح سازمان‌دهی می‌شود:

سطح اول :(Base Models) مجموعه‌ای از مدل‌های متنوع) مانند درخت تصمیم، SVMو (KNN بر روی داده‌های اصلی آموزش می‌بینند. هر یک از این مدل‌ها پیش‌بینی خاص خود را ارائه می‌دهند.

سطح دوم :(Meta-Learner) این مدل به جای داده‌های خام ورودی، از **خروجی‌های مدل‌های سطح اول** به عنوان ورودی استفاده می‌کند. نقش «متا لرنر» این است که یاد بگیرد کدام‌یک از مدل‌های سطح اول قابل‌اعتمادتر هستند و چگونه باید پیش‌بینی‌های آن‌ها را با هم ترکیب کند تا کمترین خطا حاصل شود. در واقع، این مدل «یاد می‌گیرد که چگونه از یادگیرنده‌ها استفاده کند.«

ب) ضرورت استفاده از Cross-Validation در آموزشMeta-Learner

یکی از چالش‌های اصلی درStacking ، جلوگیری از نشت داده (Data Leakage) و بیش‌برازش است. اگر متا لرنر روی همان داده‌هایی آموزش ببیند که مدل‌های سطح اول روی آن‌ها آموزش دیده‌اند، دچار یک «اطمینان کاذب» می‌شود؛ زیرا مدل‌های سطح اول پیش‌بینی‌های بسیار دقیقی برای داده‌های آموزشی خود دارند.

برای حل این مشکل از **Cross-Validation**استفاده می‌شود:

داده‌ها به چند بخش تقسیم می‌شوند. مدل‌های سطح اول روی بخشی از داده‌ها آموزش دیده و روی بخش «نادیده (Out-of-fold) «پیش‌بینی انجام می‌دهند.

متا لرنر فقط بر اساس این پیش‌بینی‌های «داده‌های نادیده» آموزش می‌بیند. این کار باعث می‌شود متا لرنر با نحوه عملکرد مدل‌های سطح اول روی داده‌های واقعی و جدید آشنا شود و قدرت تعمیم‌پذیری سیستم حفظ گردد.

ج) مثال از معماری یک سیستمStacking

فرض کنید می‌خواهیم قیمت یک ملک را پیش‌بینی کنیم (مسئله رگرسیون):

مدل‌های سطح اول:(Base Learners)

مدل ۱ (رگرسیون خطی) :بر اساس روابط ساده متراژ و قیمت.

مدل ۲ (درخت تصمیم) :برای مدل‌سازی روابط غیرخطی و محله‌ها.

مدل ۳ (شبکه عصبی) :برای شناسایی الگوهای بسیار پیچیده.

ورودی متا لرنر :پیش‌بینی‌های عددی این سه مدل (مثلاً: ۲ میلیارد، ۲,۱ میلیارد و ۱,۹ میلیارد).

مدل سطح دوم :(Meta-Learner) معمولاً یک مدل ساده مثل **رگرسیون خطی** یا **Ridge**که وزن بهینه را به هر یک از این پیش‌بینی‌ها اختصاص داده و قیمت نهایی (مثلاً ۲,۰۵ میلیارد) را به عنوان خروجی صادر می‌کند.

۱۱.

الف) مزایای یادگیری جمعی

۱ .**افزایش دقت پیش‌بینی :(Improved Accuracy)** مهم‌ترین مزیت این روش، دستیابی به سطحی از دقت است که معمولاً فراتر از توانایی یک مدل واحد است. با ترکیب پیش‌بینی‌های مختلف، خطاهای تصادفی یکدیگر را خنثی کرده و به پاسخ بهینه نزدیک‌تر می‌شوند.

۲ .**کاهش واریانس و سوگیری :(Reduced Variance & Bias)** این روش‌ها می‌توانند به‌طور هم‌زمان با دو مشکل بزرگ مدل‌ها مقابله کنند. برای مثال، Baggingبا کاهش واریانس از بیش‌برازش جلوگیری می‌کند و Boostingبا کاهش سوگیری، دقت مدل‌های ضعیف را بالا می‌برد.

۳ .**پایداری و استواری :(Robustness)** مدل‌های جمعی در برابر داده‌های پرت (Outliers) و نویزها پایداری بیشتری دارند. از آنجا که خروجی نهایی حاصل اجماع چندین مدل است، انحراف یک مدل بر اثر داده‌های غلط تأثیر مخربی بر کل سیستم نمی‌گذارد.

۴ .**توانایی حل مسائل پیچیده :**این معماری‌ها قادرند الگوهای پیچیده و غیرخطی در مجموعه‌داده‌های بزرگ را که مدل‌های ساده کلاسیک قادر به شناسایی آن‌ها نیستند، به‌خوبی استخراج و مدل‌سازی کنند.

ب) معایب و چالش‌های یادگیری جمعی

۱ .**کاهش تفسیرپذیری :(Loss of Interpretability)** برخلاف یک درخت تصمیم ساده که مسیر آن به‌راحتی قابل ردیابی است، فهم منطقِ پشتِ هزاران مدل ترکیبی (مانند یک جنگل تصادفی بزرگ) برای انسان بسیار دشوار است. این مدل‌ها اغلب به عنوان «جعبه سیاه» شناخته می‌شوند.

۲ .**هزینه محاسباتی بالا :(Computational Expense)** آموزش و نگهداری چندین مدل به جای یک مدل، به توان پردازشی (CPU/GPU) و حافظه (RAM) بسیار بیشتری نیاز دارد. همچنین زمان مورد نیاز برای آموزش و استنتاج (Inference) در این روش‌ها طولانی‌تر است.

۳ .**دشواری در پیاده‌سازی و تنظیم :(Complexity in Tuning)** انتخاب ترکیب درستی از مدل‌ها، نحوه وزن‌دهی به آن‌ها و تنظیم ابرپارامترهای (Hyperparameters) هر کدام، فرآیندی زمان‌بر است که نیاز به تخصص و تجربه بالایی دارد.

۴ .**خطر بیش‌برازش در صورت عدم کنترل :**اگرچه برخی روش‌های جمعی با هدف کاهش بیش‌برازش طراحی شده‌اند، اما برخی دیگر) مانند (Boosting اگر بیش از حد طولانی آموزش ببینند یا روی داده‌های نویزدار اعمال شوند، ممکن است به‌شدت دچار بیش‌برازش شده و قدرت تعمیم‌دهی خود را از دست بدهند.

۱۲.

لینک کد سوال

هدف کلی سوال :

ما در این مرحله، گامی فراتر از درخت‌های تصمیم تکی برمی‌داریم و به سراغ **یادگیری تجمیعی (Ensemble Learning)** می‌رویم. هدف ما این است که با ترکیب قدرت چندین مدل، بر نقاط ضعف یک مدل واحد غلبه کنیم و دقت پیش‌بینی خود را بهبود ببخشیم. در این تمرین، دو رویکرد قدرتمند یعنی **جنگل تصافی (Random Forest)**و **تقویت گرادیان (Gradient Boosting)** را با هم مقایسه می‌کنیم

کد این بخش:

```
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier

# ۱. پیش‌پردازش (با فرض اینکه X_train, X_test, y_train, y_test از قبل
آماده هستند)
# اگر محیط را ریست کردی، کدهای پیش‌پردازش سوال ۵ را دوباره اجرا کن.

# ۲. آموزش مدل Random Forest
rf_model = RandomForestClassifier(n_estimators=200, random_state=42)
rf_model.fit(X_train, y_train)

train_acc_rf = rf_model.score(X_train, y_train)
test_acc_rf = rf_model.score(X_test, y_test)

print(f"Random Forest - Train Acc: {train_acc_rf:.4f}, Test Acc:
{test_acc_rf:.4f}")

# ۳. آموزش مدل Gradient Boosting
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train, y_train)

train_acc_gb = gb_model.score(X_train, y_train)
test_acc_gb = gb_model.score(X_test, y_test)

print(f"Gradient Boosting - Train Acc: {train_acc_gb:.4f}, Test Acc:
{test_acc_gb:.4f}")
```

۱ .تحلیل عملکرد جنگل تصادفی(Random Forest)

ما یک مدل جنگل تصادفی را با **۲۰۰ درخت** آموزش می‌دهیم.

دقت آموزش :(1.0000) می‌بینیم که مدل ما به دقت ٪۱۰۰ روی داده‌های آموزش رسیده است.

دقت آزمون :(0.8212) دقت روی داده‌های جدید به عدد ٪۸۲,۱۲ رسیده که نسبت به درخت‌های تکی قبلی (که حدود ٪۸۰-۷۹ بودند) پیشرفت خوبی را نشان می‌دهد.

تحلیل وضعیت (بخش ۴ سوال) :ما مشاهده می‌کنیم که مدل Random Forest دچار **Overfitting)بیش‌برازش (**شده است. دلیل این اتفاق این است که ۲۰۰ درختِ عمیق، تمام جزئیات و حتی نویزهای داده‌های آموزشی را حفظ کرده‌اند (دقت ٪۱۰۰). با این حال، چون این مدل از میانگین‌گیری بین درخت‌های مختلف استفاده می‌کند، توانسته است دقت تست را در سطح خوبی نگه دارد.

۲ .تحلیل عملکرد تقویت گرادیان(Gradient Boosting)

در گام بعدی، مدل تقویت گرادیان را آموزش می‌دهیم که درخت‌ها را به صورت متوالی و برای اصلاح خطاهای یکدیگر می‌سازد. نتایج به شرح زیر است:

دقت آموزش :(0.9031) دقت در مرحله آموزش کمتر از مدل قبلی است.

دقت آزمون :(0.8212) دقت تست دقیقاً با مدل جنگل تصادفی برابر است.

تحلیل وضعیت (بخش ۴ سوال) :ما می‌بینیم که مدل Gradient Boosting **تعادل بهتری** برقرار کرده است. فاصله بین دقت آموزش و آزمون در این مدل بسیار کمتر است (حدود ٪۸ اختلاف در مقابل ٪۱۸ اختلاف در جنگل تصادفی). این نشان می‌دهد که این مدل کمتر دچار Overfitting شده و به جای حفظ کردن داده‌ها، الگوهای تعمیم‌پذیرتری را یاد گرفته است.

۳ .مقایسه و تحلیل نهایی

با در کنار هم قرار دادن این دو مدل، نتایج زیر را استخراج می‌کنیم:

چرا Ensemble بهتر عمل کرد؟ ما می‌بینیم که هر دو مدل تجمیعی با رسیدن به دقت **٪۸۲,۱۲**، از درخت تصمیم تکی (که بهترین حالتش حدود ٪۸۰ بود) بهتر عمل کرده‌اند. این نشان‌دهنده قدرت «خرد جمعی» در مدل‌های تجمیعی است.

کدام مدل دچار Overfitting شده است؟ هر دو مدل درجاتی از بیش‌برازش را دارند، اما **Random Forest**به طور قطعی دچار Overfitting شدیدتری شده است (دقت کامل ۱,۰ در آموزش). دلیل آن این است که درRandom Forest ، درخت‌ها می‌توانند تا بی‌نهایت رشد کنند و تمام جزئیات را یاد بگیرند.

دلیل موفقیت :Gradient Boosting ما تحلیل می‌کنیم که Gradient Boosting به دلیل ساختار مرحله‌به‌مرحله‌اش، معمولاً هوشمندانه‌تر عمل می‌کند. این مدل به جای اینکه همه چیز را با هم یاد بگیرد، در هر مرحله سعی می‌کند فقط اشتباهات قبلی را جبران کند، که این امر منجر به یک مدل «محتاط‌تر» و متعادل‌تر شده است.

نتیجه‌گیری نهایی ما :ما با این آزمایش ثابت کردیم که برای رسیدن به بالاترین دقت در مسئله تایتانیک، استفاده از مدل‌های تجمیعی ضروری است. اگرچه هر دو مدل دقت تست یکسانی دادند، اما ما مدل **Gradient Boosting**را به دلیل پایداری بیشتر و Overfitting کمتر، مدل برتر و قابل‌اعتمادتری برای داده‌های واقعی می‌دانیم.