تمرین اول

نام درس: داده کاوی پیشرفته

استاد درس: دکتر بهروز مینایی

نام: محمد حقیقت

شماره دانشجویی: 403722042

گرایش: هوش مصنوعی

دانشکده: مهندسی کامپیوتر

نیم سال دوم 1403-1404

سوال اول

1

۱. تحلیل احساسات (Sentiment Analysis)

چالش‌ها: احساسات ظریف یا متناقض: طعنه، کنایه یا احساسات ترکیبی (مانند «خیلی خیلی عالی بود!» به‌صورت طعنه‌آمیز) ممکن است به اشتباه تفسیر شوند.

تنوع فرهنگی و زمینه‌ای: یک عبارت ممکن است بسته به فرهنگ یا زمینه، معنای مثبت یا منفی داشته باشد.

تنوع زبانی: لهجه‌ها، اصطلاحات عامیانه یا زبان غیررسمی می‌توانند دقت مدل را کاهش دهند.

راه‌حل‌ها: تنظیم دقیق مدل‌ها روی مجموعه داده‌های خاص دامنه و غنی از زمینه که شامل نمونه‌هایی از طعنه و تفاوت‌های فرهنگی است.

استفاده از اطلاعات زمینه‌ای (مانند پیشینه نویسنده یا بستر گفتگو) برای بهبود تفسیر احساسات.

بهره‌گیری از مکانیزم‌های توجه پیشرفته برای تمرکز بر بخش‌های کلیدی متن و آموزش با داده‌های چندزبانه برای پوشش تنوع زبانی.

۲. طبقه‌بندی موضوعی (Topic Classification)

چالش‌ها: موضوعات مشابه یا هم‌پوشان: تمایز بین موضوعات نزدیک به هم یا متونی که چند موضوع را پوشش می‌دهند دشوار است.

موضوعات نوظهور یا نادر: داده‌های آموزشی محدود برای موضوعات جدید یا خاص، طبقه‌بندی را پیچیده می‌کند.

متون کوتاه: ورودی‌های کوتاه، مانند پست‌های شبکه‌های اجتماعی، ممکن است زمینه کافی برای تشخیص موضوع نداشته باشند.

راه‌حل‌ها: استفاده از مدل‌سازی سلسله‌مراتبی موضوعات یا طبقه‌بندی چندبرچسبی برای مدیریت موضوعات هم‌پوشان و تخصیص چندین برچسب در صورت نیاز.

بهره‌گیری از یادگیری با داده‌های کم یا بدون نظارت برای موضوعات نوظهور، همراه با برچسب‌گذاری نیمه‌خودکار.

استفاده از افزایش داده و جاسازی‌های زمینه‌ای برای بهبود درک معنایی، به‌ویژه در متون کوتاه.

۳. تشخیص هرزنامه (Spam Detection)

چالش‌ها:

تکامل روش‌های هرزنامه: هرزنامه‌نویسان از تکنیک‌هایی مانند غلط‌های املایی عمدی، کاراکترهای خاص یا تصاویر حاوی متن برای دور زدن فیلترها استفاده می‌کنند.

تمایز محتوای قانونی: ایمیل‌های تبلیغاتی ممکن است شبیه هرزنامه به نظر برسند و باعث خطای مثبت کاذب شوند.

نیاز به پردازش بلادرنگ: حجم بالای داده‌ها نیازمند تشخیص سریع و مقیاس‌پذیر است.

راه‌حل‌ها: بازآموزی مداوم مدل‌ها با داده‌های جدید هرزنامه برای سازگاری با الگوهای نوظهور.

ترکیب تحلیل متن با فراداده‌ها (مانند رفتار فرستنده، منبع ایمیل یا الگوهای لینک) با استفاده از مدل‌های ترکیبی یا چندگانه.

بهینه‌سازی معماری مدل‌ها برای مقیاس‌پذیری، مانند پردازش موازی یا مدل‌های سبک‌تر، برای مدیریت داده‌های بزرگ.

۴. طبقه‌بندی محتوای توهین‌آمیز (Toxic Content Classification)

چالش‌ها:

حساسیت فرهنگی: تعریف محتوای توهین‌آمیز بسته به فرهنگ و زمینه متفاوت است.

زبان غیرمستقیم: سخنان نفرت‌پراکن ممکن است با استعاره یا عبارات ظریف بیان شوند و تشخیص آن‌ها پیچیده باشد.

سوگیری و خطاهای مثبت کاذب: داده‌های آموزشی مغرضانه ممکن است محتوای قانونی را به اشتباه توهین‌آمیز طبقه‌بندی کنند.

راه‌حل‌ها:

آموزش با مجموعه داده‌های متنوع و متعادل که توسط افراد با پیشینه‌های فرهنگی مختلف برچسب‌گذاری شده‌اند تا سوگیری کاهش یابد.

استفاده از تکنیک‌های تفسیرپذیری برای درک تصمیم‌گیری مدل و شناسایی سوگیری‌های احتمالی.

پیاده‌سازی آستانه‌های طبقه‌بندی قابل تنظیم و بازبینی انسانی برای موارد مرزی.

۵. طبقه‌بندی نیت کاربر (User Intent Classification)

چالش‌ها:

بیان‌های متنوع یا مبهم: کاربران ممکن است یک نیت را به روش‌های مختلفی بیان کنند (مانند درخواست رزرو با عبارات متفاوت).

نیت‌های پویا: نیت‌های جدید یا در حال تغییر کاربران نیازمند تطبیق سریع مدل هستند.

نیاز به پاسخ سریع: چت‌بات‌ها برای تعاملات روان به تشخیص سریع نیت نیاز دارند.

راه‌حل‌ها:

استفاده از طبقه‌بندی چندبرچسبی یا سلسله‌مراتبی برای مدیریت نیت‌های چندگانه یا مبهم.

بهره‌گیری از یادگیری فعال برای شناسایی و برچسب‌گذاری نیت‌های جدید و افزایش داده برای ثبت بیان‌های متنوع.

بهینه‌سازی سرعت با استفاده از مدل‌های فشرده یا پنجره‌های زمینه‌ای بزرگ‌تر برای بهبود درک نیت.

۶. طبقه‌بندی اسناد حقوقی یا پزشکی

چالش‌ها:

نیاز به دقت بالا: خطاها در حوزه‌های حساس مانند حقوقی یا پزشکی می‌توانند پیامدهای جدی داشته باشند.

اصطلاحات تخصصی: زبان پیچیده و اصطلاحات خاص این حوزه‌ها نیازمند دانش تخصصی است.

اسناد طولانی و پیچیده: درک روابط بین بخش‌های مختلف اسناد طولانی از نظر محاسباتی سنگین است.

راه‌حل‌ها:

تنظیم دقیق مدل‌های از پیش آموزش‌دیده خاص دامنه (مانند BioBERT برای متون پزشکی) روی داده‌های برچسب‌گذاری‌شده توسط متخصصان.

ادغام دانش‌نامه‌ها یا هستی‌شناسی‌های تخصصی برای بهبود درک اصطلاحات و روابط.

استفاده از معماری‌های پیشرفته ترانسفورمر با مکانیزم‌های توجه طولانی‌مدت برای پردازش اسناد گسترده.

2

برای اینکه مشخص کنیم آیا یکی از دو مدل M1 یا M2 به طور قابل توجهی بهتر از دیگری است، از یک آزمون t جفت شده (paired t-test) بر روی نرخ‌های خطای گزارش‌شده استفاده می‌کنیم. این آزمون به این دلیل مناسب است که در هر ۱۰ مرحله cross-validation، از تقسیم‌بندی یکسان داده‌ها برای هر دو مدل استفاده شده است، به این معنی که مشاهدات ما زوج هستند. هدف این آزمون بررسی این موضوع است که آیا میانگین تفاوت در نرخ‌های خطا به طور معناداری با صفر متفاوت است یا خیر.

فرضیه صفر (H_0): تفاوت معناداری بین میانگین نرخ خطای مدل M1 و مدل M2 وجود ندارد (یعنی $\mu_d=0$).

فرضیه جایگزین (H_a): تفاوت معناداری بین میانگین نرخ خطای مدل M1 و مدل M2 وجود دارد (یعنی $\mu\neq0$).

سطح معناداری (α):0.01 (یا ۱٪)

محاسبه تفاوت‌ها:

$D_i = Error_i\ (M1) - Error_i\ (M2)$

Round	M1	M2	d_i
1	30.5	22.4	8.1
2	32.2	14.5	17.7
3	20.7	22.4	-1.7
4	20.6	19.6	1.0
5	31.0	20.7	10.3
6	41.0	20.4	20.6
7	27.7	22.1	5.6
8	26.0	19.4	6.6
9	21.5	16.2	5.3
10	26.0	35.0	-9.0

محاسبه میانگین تفاوت‌ها:

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i = \frac{8.1 + 17.7 - 1.7 + 1.0 + 10.3 + 20.6 + 5.6 + 6.6 + 5.3 - 9.0}{10} = \frac{64.5}{10} = 6.45$$

محاسبه انحراف معیار تفاوت‌ها (s_d):

$$s_d = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(d_i - \bar{d}\right)^2}$$

ابتدا مربعات انحراف از میانگین $(d_i - \bar{d})^2$ را محاسبه می‌کنیم:

$(8.1 - 6.45)^2 = 2.7225$

$(17.7 - 6.45)^2 = 126.0225$

(-1.7 - 6.45)² = 67.4225

(1.0 - 6.45)² = 29.7025

(10.3 - 6.45)² = 14.8225

(20.6 - 6.45)² = 200.6025

(5.6 - 6.45)² = 0.7225

(6.6 - 6.45)² = 0.0225

(5.3 - 6.45)² = 1.3225

(-9.0 - 6.45)² = 237.9025

جمع مربعات: 681.27

$$s_d \approx \sqrt{\frac{681.27}{9}} = \sqrt{75.7} \approx 8.7$$

محاسبه t-statistic:

$$t = \frac{\bar{d}}{s_d/\sqrt{n}} = \frac{6.45}{8.7/\sqrt{10}} \approx \frac{6.45}{2.75} \approx 2.35$$

تعیین درجه آزادی (df)

$D_f = n - 1 = 10 - 1 = 9$

تعیین مقدار بحرانی (Critical Value)

در این مرحله هدف این است که ببینیم مقدار محاسبه‌شده‌ی t (یعنی ۲٫۳۵) به اندازه‌ای بزرگ هست که فرض صفر را رد کنیم یا نه. برای این کار به یک مقدار مرجع از جدول توزیع t نیاز داریم که به آن مقدار بحرانی (t-critical) گفته می‌شود.

چون ما از آزمون t جفت‌شده‌ی دوطرفه استفاده می‌کنیم، و تعداد نمونه‌ها n=10 است، پس تعداد درجه آزادی برابر می‌شود با:

df = n − 1 = 9

همچنین ما در این آزمون از سطح معناداری ۱٪ استفاده می‌کنیم، یعنی:

$\alpha = 0.01$

در آزمون دوطرفه، این مقدار به دو قسمت ۰/۰۰۵ (نیم درصد) در هر طرف تقسیم می‌شود. پس باید مقدار بحرانی را برای:

$t_{critical}\ (0.005\ ,\ 9)$

از جدول توزیع t پیدا کنیم. این مقدار از جدول تقریبا برابر است با:

$t_{critical} \approx \pm 3.250$

نتیجه‌گیری آماری

ما قبلاً مقدار t-statistic را محاسبه کرده بودیم:

$t = 2.35$

حالا این مقدار را با مقدار بحرانی مقایسه می‌کنیم:

$|2.35| < 3.25$

یعنی مقدار t-statistic از مقدار بحرانی کمتر است.

در نتیجه، فرض صفر را رد نمی‌کنیم. به زبان آماری:

ما نمی‌توانیم با اطمینان ۹۹٪ بگوییم که تفاوت میان عملکرد دو مدل (M1 و M2) از نظر آماری معنادار است.

میانگین نرخ خطا برای مدل M1 برابر با $27.72 = 10 / 277.2$ است.

میانگین نرخ خطا برای مدل M2 برابر با $21.27 = 10 / 212.7$ است.

به طور متوسط، مدل M2 نرخ خطای پایین‌تری (21.27%) نسبت به مدل M1 (27.72%) داشته است و میانگین این تفاوت برابر با 6.45% است.

با اینکه میانگین خطای مدل M2 کمتر از M1 است (یعنی ظاهرا بهتر عمل کرده)، اما وقتی تحلیل آماری انجام می‌دهیم، می‌بینیم که این تفاوت در سطح اطمینان ۹۹٪ معنادار نیست.

بنابراین، نمی‌توانیم نتیجه بگیریم که M2 به‌طور قطعی بهتر از M1 است.

ممکن است این تفاوت صرفا به خاطر تصادف در داده‌ها باشد.

3

۱. زمانی که دانش حوزه‌ای (Domain Knowledge) تعداد خوشه‌ها را مشخص کرده است

در برخی مسائل، تعداد خوشه‌ها از قبل و بر اساس تجربه یا دانش کارشناسی مشخص است و نیازی به تعیین خودکار آن نیست.

مثال: در یک مسئله تشخیص پزشکی، ممکن است دقیقا بدانیم که سه نوع بیماری وجود دارد. اگر الگوریتم به صورت خودکار تعداد خوشه‌ها را تعیین کند، ممکن است:

خوشه‌ها را بیش از حد تقسیم یا ترکیب کند.

باعث شود که دسته‌بندی واقعی و مفهومی داده‌ها نادیده گرفته شود.

در این حالت، تعیین خودکار خوشه‌ها می‌تواند با واقعیت و نیاز مسئله تعارض داشته باشد.

۲. زمانی که داده‌ها ساختار خوشه‌ای مشخصی ندارند

برخی داده‌ها ساختار خوشه‌ای واضحی ندارند و به جای خوشه‌های مجزا، ممکن است بر روی یک طیف پیوسته (continuum) یا یک ساختار پیچیده‌تر مانند منیفولد (manifold) قرار گرفته باشند.

مثال: در داده‌های اجتماعی یا داده‌های تصویری، ممکن است هیچ مرز مشخصی بین گروه‌ها وجود نداشته باشد.

در این شرایط الگوریتم ممکن است به اشتباه خوشه‌هایی ایجاد کند که صرفا نتیجه‌ی نوسانات یا نویز در داده‌ها هستند.

این کار منجر به خوشه‌بندی غیرواقعی و غیرقابل‌تفسیر می‌شود.

در این حالت، خروجی الگوریتم بیشتر به تحمیل ساختار به داده شباهت دارد تا کشف یک ساختار واقعی.

سوال دوم

ماتریس شباهت اولیه:

	p1	p2	p3	p4	p5
p1	1	0.1	0.41	0.55	0.35
p2	0.1	1	0.64	0.47	0.98
p3	0.41	0.64	1	0.44	0.85
p4	0.55	0.47	0.44	1	0.76
p5	0.35	0.98	0.85	0.76	1

خوشه‌بندی سلسله‌مراتبی پیوند یگانه (Single Linkage)

در Single Linkage ، شباهت بین دو خوشه، بیشترین شباهت بین هر دو نقطه در آن خوشه‌ها است.

گام اول: پیدا کردن مشابه ترین جفت

با نگاه کردن به ماتریس، بیشترین شباهت 0.98 است، بین p2 و p5.

ادغام: {p2, p5}

سطح شباهت: 0.98

گام بعدی: به‌روزرسانی ماتریس شباهت

اکنون خوشه‌های {p1}، {p3}، {p4} و {p2, p5} را داریم. باید شباهت خوشه جدید {p2, p5} را با بقیه با استفاده از پیوند یگانه محاسبه کنیم:

Sim({p2, p5}, p1) = max(Sim(p2, p1), Sim(p5, p1)) = max(0.10, 0.35) = 0.35

Sim({p2, p5}, p3) = max(Sim(p2, p3), Sim(p5, p3)) = max(0.64, 0.85) = 0.85

Sim({p2, p5}, p4) = max(Sim(p2, p4), Sim(p5, p4)) = max(0.47, 0.76) = 0.76

ماتریس جدید:

	p1	p3	p4	{p2,p5}
p1	1	0.41	0.55	0.35
p3	0.41	1	0.44	0.85
p4	0.55	0.44	1	0.76
{p2,p5}	0.35	0.85	0.76	1

گام بعدی: پیدا کردن مشابه ترین جفت بعدی

بیشترین شباهت در ماتریس جدید 0.85 است، بین p3 و {p2, p5}.

ادغام: {p2, p3, p5} و سطح شباهت: 0.85

گام بعدی: به‌روزرسانی ماتریس شباهت

اکنون خوشه‌های {p1} ، {p4} و {p2, p3, p5} را داریم.

Sim({p2, p3, p5}, p1) = max(Sim(p2,p1), Sim(p3,p1), Sim(p5,p1)) = max(0.10, 0.41, 0.35) = 0.41

Sim({p2, p3, p5}, p4) = max(Sim(p2,p4), Sim(p3,p4), Sim(p5,p4)) = max(0.47, 0.44, 0.76) = 0.76

ماتریس جدید:

	p1	p4	{p2,p3,p5}
p1	1	0.55	0.41
p4	0.55	1	0.76
{p2,p3,p5}	0.41	0.76	1

گام بعدی: پیدا کردن مشابه ترین جفت بعدی

بیشترین شباهت 0.76 است، بین p4 و {p2, p3, p5}.

ادغام: {p2, p3, p4, p5} و سطح شباهت: 0.76

گام بعدی: به‌روزرسانی ماتریس شباهت

اکنون خوشه‌های {p1} و {p2, p3, p4, p5} را داریم.

Sim({p2, p3, p4, p5}, p1) = max(Sim(p2,p1), Sim(p3,p1), Sim(p4,p1), Sim(p5,p1)) = max(0.10, 0.41, 0.55, 0.35) = 0.55

ماتریس جدید:

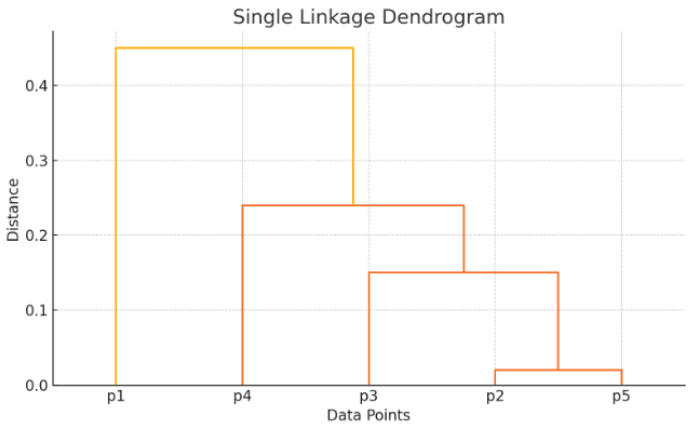
	p1	{p2,p3,p4,p5}
p1	1	0.55
{p2,p3,p4,p5}	0.55	1

گام بعدی: ادغام نهایی

بیشترین و تنها شباهت 0.55 است، بین p1 و {p2, p3, p4, p5}.

ادغام: {p1, p2, p3, p4, p5} و سطح شباهت: 0.55

دندروگرام (نمودار درختی) پیوند یگانه:



دندروگرام فرآیند ادغام را به صورت بصری نشان می‌دهد. ارتفاع خطوط عمودی نشان دهنده فاصله است که در آن خوشه‌ها ادغام شده‌اند.

فاصله از فرمول Similarty – 1 بدست می آید.

خوشه‌بندی سلسله‌مراتبی پیوند کامل (Complete Linkage)

در پیوند کامل، شباهت بین دو خوشه، کمترین شباهت بین هر دو نقطه در آن خوشه‌ها است.

ماتریس اولیه:

	p1	p2	p3	p4	p5
p1	1	0.1	0.41	0.55	0.35
p2	0.1	1	0.64	0.47	0.98
p3	0.41	0.64	1	0.44	0.85
p4	0.55	0.47	0.44	1	0.76
p5	0.35	0.98	0.85	0.76	1

گام اول: پیدا کردن مشابه ترین جفت

این مرحله در ابتدا مشابه پیوند یگانه است، زیرا نقاط منفرد را مقایسه می‌کنیم.

بیشترین شباهت 0.98 است، بین p2 و p5.

ادغام: {p2, p5}

سطح شباهت: 0.98

گام بعدی: به‌روزرسانی ماتریس شباهت

اکنون خوشه‌های {p1}، {p3}، {p4} و {p2, p5} را داریم. باید شباهت خوشه جدید {p2, p5} را با بقیه با استفاده از پیوند کامل محاسبه کنیم:

Sim({p2, p5}, p1) = min(Sim(p2, p1), Sim(p5, p1)) = min(0.10, 0.35) = **0.10**

Sim({p2, p5}, p3) = min(Sim(p2, p3), Sim(p5, p3)) = min(0.64, 0.85) = **0.64**

Sim({p2, p5}, p4) = min(Sim(p2, p4), Sim(p5, p4)) = min(0.47, 0.76) = **0.47**

ماتریس جدید:

	p1	p3	p4	{p2,p5}
p1	1	0.41	0.55	0.1
p3	0.41	1	0.44	0.64
p4	0.55	0.44	1	0.47
{p2,p5}	0.1	0.64	0.47	1

گام ۳: پیدا کردن مشابه ترین جفت بعدی

بیشترین شباهت در ماتریس جدید 0.64 است، بین p3 و {p2, p5}.

ادغام: {p2, p3, p5}

سطح شباهت: 0.64

گام بعدی: به‌روزرسانی ماتریس شباهت

اکنون خوشه‌های {p1}، {p4} و {p2, p3, p5} را داریم.

Sim({p2, p3, p5}, p1) = min(Sim(p2,p1), Sim(p3,p1), Sim(p5,p1)) = min(0.10, 0.41, 0.35) = **0.10**

Sim({p2, p3, p5}, p4) = min(Sim(p2,p4), Sim(p3,p4), Sim(p5,p4)) = min(0.47, 0.44, 0.76) = **0.44**

ماتریس جدید:

	p1	p4	{p2,p3,p5}
p1	1	0.55	0.1
p4	0.55	1	0.44
{p2,p3,p5}	0.1	0.44	1

گام بعدی: پیدا کردن مشابه ترین جفت بعدی

بیشترین شباهت 0.55 است، بین p1 و p4.

ادغام: {p1, p4}

سطح شباهت: 0.55

گام بعدی: به‌روزرسانی ماتریس شباهت

اکنون خوشه‌های {p1, p4} و {p2, p3, p5} را داریم.

Sim({p1, p4}, {p2, p3, p5}) = min(Sim(p1,p2), Sim(p1,p3), Sim(p1,p5), Sim(p4,p2), Sim(p4,p3), Sim(p4,p5)) = min(0.10, 0.41, 0.35, 0.47, 0.44, 0.76) = 0.10

ماتریس جدید:

	{p1,p4}	{p2,p3,p5}
{p1,p4}	1	0.1
{p2,p3,p5}	0.1	1

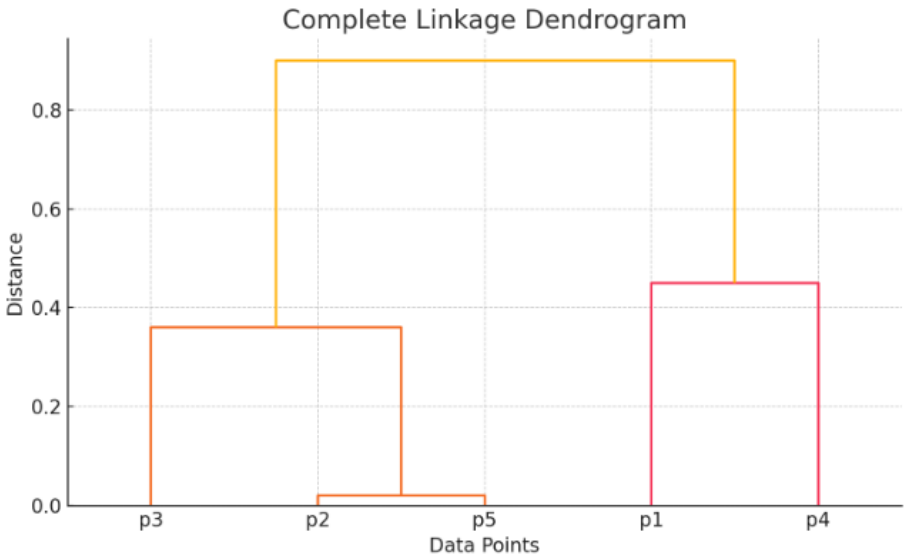
گام بعدی: ادغام نهایی

بیشترین و تنها شباهت 0.10 است، بین {p1, p4} و {p2, p3, p5}.

ادغام: {p1, p2, p3, p4, p5}

سطح شباهت: 0.10

دندروگرام (نمودار درختی) پیوند کامل:



سوال سوم

برای محاسبه ضریب سیلوئت (Silhouette Coefficient) برای هر نقطه، از فرمول زیر استفاده می‌کنیم:

$$s(i) = \frac{b(i) - a(i)}{\max\big(a(i), b(i)\big)}$$

a(i): میانگین فاصله نقطه i تا سایر نقاط در همان خوشه

b(i) :کمترین میانگین فاصله بین نقطه i و نقاط در خوشه‌های دیگر

ضریب سیلوئت بین 1- و 1 است:

اگر 1 = s(i): نقطه i به خوبی در خوشه خودش قرار گرفته است.

اگر 1- = s(i): نقطه i به اشتباه در این خوشه قرار گرفته و به خوشه دیگری تعلق دارد.

اگر 0 = s(i): نقطه i در مرز بین خوشه‌ها قرار دارد.

خوشه‌ها:

خوشه 1: {P1, P2}

خوشه 2: {P3, P4}

محاسبات برای P1:

چون فقط یک نقطه دیگر در خوشه خودش دارد:

$$a(P1) = d(P1, P2) = 0.10$$

$$b(P1) = \frac{d(P1, P3) + d(P1, P4)}{2} = \frac{0.65 + 0.55}{2} = 0.60$$

$$s(P1) = \frac{0.60 - 0.10}{\max(0.60,0.10)} = \frac{0.50}{0.60} \approx 0.833$$

محاسبات برای P2:

$$a(P2) = d(P2, P1) = 0.10$$

$$b(P2) = \frac{d(P2, P3) + d(P2, P4)}{2} = \frac{0.70 + 0.60}{2} = 0.65$$

$$s(P2) = \frac{0.65 - 0.10}{\max(0.65,0.10)} = \frac{0.55}{0.65} \approx 0.846$$

محاسبات مربوط به P3:

$$a(P3) = d(P3, P4) = 0.30$$

$$b(P3) = \frac{d(P3,P1) + d(P3,P2)}{2} = \frac{0.65 + 0.70}{2} = 0.675$$

$$s(P3) = \frac{0.675 - 0.30}{\max(0.675,0.30)} = \frac{0.375}{0.675} \approx 0.556$$

محاسبات مربوط به P4:

$$a(P4) = d(P4,P3) = 0.30$$

$$b(P4) = \frac{d(P4,P1) + d(P4,P2)}{2} = \frac{0.55 + 0.60}{2} = 0.575$$

$$s(P4) = \frac{0.575 - 0.30}{\max(0.575,0.30)} = \frac{0.275}{0.575} \approx 0.478$$

محاسبه ضریب سیلوئت برای هر خوشه:

ضریب سیلوئت هر خوشه، میانگین ضریب سیلوئت نقاط آن خوشه است.

میانگین خوشه 1:

$$\text{Mean}_{\text{Cluster 1}} = \frac{s(P1) + s(P2)}{2} = \frac{0.833 + 0.846}{2} = 0.839$$

میانگین خوشه 2:

$$\text{Mean}_{\text{Cluster 2}} = \frac{s(P3) + s(P4)}{2} = \frac{0.556 + 0.478}{2} = 0.517$$

محاسبه ضریب سیلوئت کلی خوشه‌بندی:

ضریب سیلوئت کلی، میانگین ضریب سیلوئت همه نقاط است:

$$\text{Overall Silhouette} = \frac{s(P1) + s(P2) + s(P3) + s(P4)}{4} = \frac{2.713}{4} = 0.678$$

میانگین خوشه 1: 0.839

میانگین خوشه 2: 0.517

میانگین کل: 0.678

ضریب سیلوئت کلی 0.678 نشان می‌دهد که خوشه‌بندی نسبتاً خوب است (چون به 1 نزدیک‌تر است تا 1-).

خوشه ۱ با ضریب 0.840 کیفیت بهتری نسبت به خوشه ۲ با ضریب 0.517 دارد.

سوال چهارم

```
#################################################################
#          TODO: read dataset as pandas dataframe               #
#################################################################

df = pd.read_excel('titanic.xls')

df.head()
```

در این بخش فایل اکسل titanic.xls را می‌خواند و محتوای آن را در یک DataFrame به نام df ذخیره می‌کند.

با دستور df.head() پنج سطر اول DataFrame را نمایش می‌دهد تا یک دید کلی و سریع از شکل داده‌ها به دست آوریم.

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1	0	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON

```
df.shape
```

```
(1309, 14)
```

ابعاد جدول (تعداد سطرها و ستون‌ها) را برمی‌گرداند. برای مثال، (1309, 14) یعنی ۱۳۰۹ سطر و ۱۴ ستون.

```
df.describe()
```

	pclass	survived	age	sibsp	parch	fare	body
count	1309.000000	1309.000000	1046.000000	1309.000000	1309.000000	1308.000000	121.000000
mean	2.294882	0.381971	29.881135	0.498854	0.385027	33.295479	160.809917
std	0.837836	0.486055	14.413500	1.041658	0.865560	51.758668	97.696922
min	1.000000	0.000000	0.166700	0.000000	0.000000	0.000000	1.000000
25%	2.000000	0.000000	21.000000	0.000000	0.000000	7.895800	72.000000
50%	3.000000	0.000000	28.000000	0.000000	0.000000	14.454200	155.000000
75%	3.000000	1.000000	39.000000	1.000000	0.000000	31.275000	256.000000
max	3.000000	1.000000	80.000000	8.000000	9.000000	512.329200	328.000000

یک خلاصه آماری (مثل میانگین، انحراف معیار، مینیمم و ماکزیمم) برای ستون‌های عددی فراهم می‌کند. این دستور به ما کمک می‌کند تا توزیع داده‌های عددی را بهتر درک کنیم.

اطلاعات کلی در مورد DataFrame، شامل نوع داده هر ستون و تعداد مقادیر غیرخالی را نمایش می‌دهد. این دستور برای پیدا کردن ستون‌هایی که مقادیر خالی (missing values) دارند بسیار مفید است.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   pclass     1309 non-null   int64  
 1   survived   1309 non-null   int64  
 2   name       1309 non-null   object 
 3   sex        1309 non-null   object 
 4   age        1046 non-null   float64
 5   sibsp      1309 non-null   int64  
 6   parch      1309 non-null   int64  
 7   ticket     1309 non-null   object 
 8   fare       1308 non-null   float64
 9   cabin      295 non-null    object 
 10  embarked   1307 non-null   object 
 11  boat       486 non-null    object 
 12  body       121 non-null    float64
 13  home.dest  745 non-null    object 
dtypes: float64(3), int64(4), object(7)
memory usage: 143.3+ KB
```

```
X = df.drop('survived', axis=1)
y = df['survived']
```

در اینجا داده‌ها را برای مدل‌سازی یادگیری ماشین آماده می‌کنیم:

در خط اول یک DataFrame جدید به نام X ساخته می‌شود که شامل تمام ستون‌های df به جز ستون survived است. در یادگیری ماشین، به این داده‌ها ویژگی‌ها (Features) می‌گویند؛ یعنی اطلاعاتی که برای پیش‌بینی استفاده می‌شوند.

axis=1 به دستور drop می‌گوید که باید یک ستون را حذف کند.

در خط دوم یک متغیر جدید به نام y ساخته می‌شود که فقط شامل ستون survived است. این ستون متغیر هدف (Target) ماست؛ یعنی چیزی که می‌خواهیم آن را پیش‌بینی کنیم (اینکه آیا مسافر زنده مانده یا خیر).

```
X.isna().sum()
```

	0
pclass	0
name	0
sex	0
age	263
sibsp	0
parch	0
ticket	0
fare	1
cabin	1014
embarked	2
boat	823
body	1188
home.dest	564

dtype: int64

این دستور تعداد مقادیر خالی یا تعریف‌نشده (NaN) را در هر ستون از X می‌شمارد. این کار به ما کمک می‌کند تا بفهمیم کدام ستون‌ها نیاز به پاک‌سازی یا پر کردن مقادیر خالی دارند.

```
X = X.drop(['name', 'ticket', 'cabin', 'boat', 'body', 'home.dest'], axis=1)
```

در این خط، چند ستون که برای پیش‌بینی مفید نیستند یا اطلاعات نامرتبطی دارند (مانند نام، شماره بلیت، شماره کابین و ...) از مجموعه ویژگی‌های X حذف می‌شوند. این کار به ساده‌سازی مدل و افزایش دقت آن کمک می‌کند.

```
X.head()
```

	pclass	sex	age	sibsp	parch	fare	embarked
0	1	female	29.0000	0	0	211.3375	S
1	1	male	0.9167	1	2	151.5500	S
2	1	female	2.0000	1	2	151.5500	S
3	1	male	30.0000	1	2	151.5500	S
4	1	female	25.0000	1	2	151.5500	S

تعداد مقادیر یکتا در هر ستون را نمایش می‌دهد. این دستور به ما کمک می‌کند تا بفهمیم کدام ستون‌ها دسته‌بندی‌شده (Categorical) هستند (مانند جنسیت) و کدام پیوسته (Continuous) (مانند سن).

```
X.nunique()
```

	0
pclass	3
sex	2
age	98
sibsp	7
parch	8
fare	281
embarked	3

dtype: int64

مجدداً تعداد مقادیر خالی را می‌شمارد تا وضعیت ستون‌ها را قبل از پاک‌سازی ببینیم.

```
X.isna().sum()
```

	0
pclass	0
sex	0
age	263
sibsp	0
parch	0
fare	1
embarked	2

dtype: int64

```
X['age'] = X['age'].fillna(X['age'].median())
X['fare'] = X['fare'].fillna(X['fare'].median())
X['embarked'] = X['embarked'].fillna(X['embarked'].mode()[0])
```

مقادیر خالی در ستون age (سن) با میانه (median) سن تمام مسافران پر می‌شود. استفاده از میانه به جای میانگین، روشی مقاوم‌تر در برابر داده‌های پرت (outliers) است.

به طور مشابه، مقادیر خالی ستون fare (کرایه) نیز با میانه کرایه‌ها پر می‌شود.

ستون embarked (بندر سوار شدن) یک متغیر دسته‌بندی‌شده است. مقادیر خالی آن با مُد (mode)، یعنی رایج‌ترین بندر، پر می‌شود. mode() یک لیست برمی‌گرداند و [0] اولین و رایج‌ترین مقدار را انتخاب می‌کند.

این دستورات در انتها دوباره اجرا شده‌اند تا تأیید شود که هیچ مقدار خالی دیگری باقی نمانده و تغییری در تعداد مقادیر یکتا (مثلاً در ستون‌های عددی) ایجاد شده است.

```
X.nunique()
```

	0
pclass	3
sex	2
age	98
sibsp	7
parch	8
fare	281
embarked	3

dtype: int64

```
X.isna().sum()
```

	0
pclass	0
sex	0
age	0
sibsp	0
parch	0
fare	0
embarked	0

dtype: int64

```
X = pd.get_dummies(X, columns=['sex', 'embarked'], drop_first=True)
```

مدل‌های یادگیری ماشین نمی‌توانند با داده‌های متنی (مثل "male" یا "female") کار کنند و نیاز به ورودی عددی دارند.

این تابع ستون‌های دسته‌بندی‌شده مشخص‌شده (sex و embarked) را به ستون‌های عددی جدید تبدیل می‌کند. این فرآیند One-Hot Encoding نام دارد.

برای مثال، ستون sex حذف و ستون جدیدی مانند sex_male ایجاد می‌شود که مقدار آن برای مسافران مرد 1 و برای بقیه 0 است.

drop_first=True: این پارامتر برای جلوگیری از یک مشکل آماری به نام هم‌خطی چندگانه (multicollinearity) استفاده می‌شود. با حذف یکی از ستون‌های ایجاد شده (مثلاً sex_female)، اطلاعات از بین نمی‌رود، زیرا اگر sex_male برابر 0 باشد، مشخص است که جنسیت زن بوده است.

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   pclass      1309 non-null   int64  
 1   age         1309 non-null   float64
 2   sibsp       1309 non-null   int64  
 3   parch       1309 non-null   int64  
 4   fare        1309 non-null   float64
 5   sex_male    1309 non-null   bool   
 6   embarked_Q  1309 non-null   bool   
 7   embarked_S  1309 non-null   bool   
dtypes: bool(3), float64(2), int64(3)
memory usage: 55.1 KB
```

همان طور که مشاهده میشود سه ستون دیگر در فرآیند One-Hot Encoding به X اضافه شده است.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
numeric_cols = ['age', 'sibsp', 'parch', 'fare']
X[numeric_cols] = scaler.fit_transform(X[numeric_cols])
```

مقیاس ستون‌های عددی مختلف (مثل سن و کرایه) با هم تفاوت زیادی دارد. این تفاوت می‌تواند عملکرد برخی از الگوریتم‌های یادگیری ماشین را تحت تأثیر منفی قرار دهد.

در این دستور عملیات استانداردسازی را روی ستون‌های عددی مشخص‌شده انجام می‌دهد. این فرآیند داده‌ها را به گونه‌ای تغییر می‌دهد که میانگین آن‌ها 0 و انحراف معیارشان 1 شود.
fit_transform ابتدا پارامترهای لازم برای تبدیل را از داده‌ها یاد می‌گیرد (fit) و سپس آن‌ها را اعمال

می‌کند (transform). این کار باعث می‌شود تمام ویژگی‌های عددی مقیاس یکسانی داشته باشند و مدلسازی با دقت بیشتری انجام شود.

```
pca = PCA(n_components=5)
X = pca.fit_transform(X)
```

پس از آماده‌سازی و پاک‌سازی داده‌ها، تعداد زیادی ستون (ویژگی) داریم. برای ساده‌سازی مدل و حذف اطلاعات اضافی، از روش تحلیل مؤلفه‌های اصلی (Principal Component Analysis - PCA) استفاده می‌شود.

در اینجا یک شی PCA ایجاد می‌کنیم و به آن می‌گوییم که می‌خواهیم تعداد ویژگی‌ها را به ۵ مولفه اصلی کاهش دهیم. این ۵ مولفه، ترکیبات خطی از ویژگی‌های اصلی هستند که بیشترین اطلاعات (واریانس) داده‌ها را در خود حفظ می‌کنند.

X = pca.fit_transform(X): این دستور، تحلیل PCA را روی داده‌های X اجرا می‌کند. ابتدا بهترین مؤلفه‌ها را یاد می‌گیرد (fit) و سپس داده‌ها را به این فضای ۵ بعدی جدید تبدیل می‌کند. در نهایت، X جدید ما به جای ده‌ها ستون، فقط ۵ ستون خواهد داشت.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

برای اینکه بتوانیم عملکرد مدل را به درستی ارزیابی کنیم، داده‌ها را به دو بخش تقسیم می‌کنیم:

```
X_train.shape
```

```
(1047, 5)
```

```
X_test.shape
```

```
(262, 5)
```

مجموعه آموزش که شامل ۸۰ درصد داده‌ها که برای "یاد دادن" الگوها به مدل استفاده می‌شود و مجموعه آزمون که شامل ۲۰ درصد باقی‌مانده است. این داده‌ها برای مدل کاملاً جدید هستند و برای سنجش عملکرد آن روی داده‌های دیده‌نشده به کار می‌روند.

X_train.shape و X_test.shape: ابعاد (تعداد سطر و ستون) مجموعه‌های آموزش و آزمون را نمایش می‌دهند تا از درستی تقسیم‌بندی مطمئن شویم.

```
clftree = tree.DecisionTreeClassifier(random_state=42)
clftree.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

در این مرحله، مدل ماشین یادگیری انتخاب و آموزش داده می‌شود.

یک مدل طبقه‌بندی از نوع درخت تصمیم (Decision Tree) ایجاد می‌کنیم. این مدل با ساختن مجموعه‌ای از قوانین "اگر-آنگاه" یاد می‌گیرد که چگونه تصمیم‌گیری کند.

در اینجا، مدل (clftree) با استفاده از داده‌های آموزشی (X_train و y_train) الگوهای موجود را یاد می‌گیرد.

```
y_train_pred = clftree.predict(X_train)
y_test_pred = clftree.predict(X_test)
```

پس از آموزش، مدل را ارزیابی می‌کنیم تا ببینیم چقدر خوب کار می‌کند.

مدل آموزش‌دیده برای پیش‌بینی مقدار هدف (زنده ماندن یا نماندن) روی داده‌های آموزشی و آزمون استفاده می‌شود.

```
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
test_confusion = confusion_matrix(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)

print("train accuracy:",f"{train_accuracy:.2f}")
print("test accuracy:",f"{test_accuracy:.2f}")
print("recall:",f"{test_recall:.2f}")
print("precision:",f"{test_precision:.2f}")
print("f1 score:",f"{test_f1:.2f}")
print("confusion matrix:",test_confusion)
```

معیارهای ارزیابی:

Accuracy (دقت): درصد کل پیش‌بینی‌های درست. train_accuracy دقت روی داده‌های آموزشی و test_accuracy دقت روی داده‌های آزمون را نشان می‌دهد.

Confusion Matrix (ماتریس درهم‌ریختگی): جدولی که نشان می‌دهد مدل چند مورد را درست و چند مورد را اشتباه پیش‌بینی کرده و نوع خطاها (مثلاً مسافری که زنده مانده را مرده پیش‌بینی کرده) را مشخص می‌کند.

Recall (بازیابی): از بین تمام کسانی که واقعاً زنده ماندند، مدل چه کسری از آن‌ها را به درستی شناسایی کرده است.

Precision (دقت پیش‌بینی): از بین تمام کسانی که مدل پیش‌بینی کرده زنده مانده‌اند، چه کسری از آن‌ها واقعاً زنده بوده‌اند.

F1-Score: میانگین هماهنگ (harmonic mean) بین Precision و Recall. این معیار زمانی مفید است که بخواهیم تعادلی بین این دو برقرار کنیم.

معیار های ارزیابی این ماشین یادگیری:

```
train accuracy: 0.95
test accuracy: 0.71
recall: 0.58
precision: 0.73
f1 score: 0.64
confusion matrix: [[119  25]
 [ 50  68]]
```

```
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)
```

RandomForestClassifier

RandomForestClassifier(random_state=42)

در اینجا به جای استفاده از یک درخت تصمیم، از مدلی بسیار قوی‌تر استفاده می‌کنیم.

RandomForestClassifier (طبقه‌بند جنگل تصادفی): این مدل یک مدل گروهی (Ensemble) است. به جای ساختن یک درخت تصمیم، تعداد زیادی درخت تصمیم (یک جنگل) می‌سازد و برای پیش‌بینی نهایی، "رأی" همه درختان را با هم ترکیب می‌کند. این کار معمولا به نتایج بسیار دقیق‌تر و پایدارتری نسبت به یک درخت تنها منجر می‌شود.

مدل جنگل تصادفی با استفاده از داده‌های آموزشی، آموزش داده می‌شود.

بخش ارزیابی و print: این قسمت دقیقاً مشابه مرحله قبل است، با این تفاوت که این بار عملکرد مدل جنگل تصادفی با تنظیمات پیش‌فرض، روی داده‌های آموزشی و آزمون سنجیده و چاپ می‌شود.

```
train accuracy: 0.97
test accuracy: 0.74
recall: 0.66
precision: 0.73
f1 score: 0.69
confusion matrix: [[115  29]
 [ 40  78]]
```

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20]
}
grid_search_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5)
grid_search_rf.fit(X_train, y_train)
rf_clf = grid_search_rf.best_estimator_
```

مدل جنگل تصادفی پارامترهای مختلفی دارد (مثلاً تعداد درختان یا عمق آن‌ها). انتخاب بهترین ترکیب از این پارامترها می‌تواند عملکرد مدل را به شدت بهبود ببخشد.

این بخش به صورت خودکار بهترین تنظیمات را پیدا می‌کند.

param_grid: یک دیکشنری تعریف می‌کنیم که مقادیر مختلف برای پارامترهایی که می‌خواهیم تست کنیم را مشخص می‌کند:

'n_estimators': تعداد درختان در جنگل (تست با ۵۰، ۱۰۰ و ۲۰۰ درخت).

'max_depth': حداکثر عمق هر درخت (تست با عمق نامحدود (None)، ۱۰ و ۲۰).

GridSearchCV (جستجوی شبکه‌ای با اعتبارسنجی متقابل): این یک ابزار بسیار قدرتمند برای بهینه‌سازی اَبَرپارامترها (Hyperparameter Tuning) است.

این ابزار تمام ترکیبات ممکن از param_grid را تست می‌کند (در اینجا ۳ × ۳ = ۹ ترکیب).

cv=5: به این معنی است که برای هر ترکیب، از (5-Fold Cross-Validation) استفاده می‌کند. یعنی داده‌های آموزشی را به ۵ بخش تقسیم کرده، ۴ بخش را برای آموزش و ۱ بخش را برای ارزیابی به کار می‌برد و این کار را ۵ بار تکرار می‌کند تا مطمئن شود که نتایج قابل اعتماد هستند.

grid_search_rf.fit(X_train, y_train): این دستور فرآیند جستجو را آغاز می‌کند. این کار ممکن است کمی زمان‌بر باشد، زیرا در پشت صحنه (۹ ترکیب × ۵ بخش) = ۴۵ مدل مختلف را آموزش می‌دهد.

rf_clf = grid_search_rf.best_estimator_: پس از پایان جستجو، بهترین مدل که بالاترین امتیاز را در اعتبارسنجی متقابل کسب کرده، در متغیر rf_clf ذخیره می‌شود.

در نهایت، مدلی که حالا بهترین تنظیمات ممکن را دارد، دوباره ارزیابی می‌شود.

```
train accuracy: 0.94
test accuracy: 0.75
recall: 0.64
precision: 0.76
f1 score: 0.70
confusion matrix: [[120  24]
 [ 42  76]]
```

```
gb_clf = GradientBoostingClassifier(random_state=42)
gb_clf.fit(X_train, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

Gradient Boosting یک تکنیک قدرتمند است که مدل‌ها (معمولاً درخت‌های تصمیم) را به صورت متوالی می‌سازد. هر درخت جدید روی خطاهای باقیمانده (residuals) از مجموعه درختان قبلی آموزش داده می‌شود. این فرآیند مانند یک تیم است که هر عضو جدید تلاش می‌کند نقاط ضعف اعضای قبلی را پوشش دهد. این کد یک مدل GradientBoostingClassifier را با تنظیمات پیش‌فرض ایجاد، آموزش و برای پیش‌بینی استفاده می‌کند.

معیار های ارزیابی برای این ماشین یادگیری:

```
train accuracy: 0.88
test accuracy: 0.74
recall: 0.58
precision: 0.78
f1 score: 0.67
confusion matrix: [[125  19]
 [ 49  69]]
```

```
ada_clf = AdaBoostClassifier(random_state=42)
ada_clf.fit(X_train, y_train)
```

```
AdaBoostClassifier
AdaBoostClassifier(random_state=42)
```

AdaBoost (Adaptive Boosting) یکی از اولین و پایه‌ای‌ترین الگوریتم‌های بوستینگ است. ایده اصلی آن این است که در هر مرحله، به نمونه‌هایی از داده که توسط مدل قبلی به اشتباه طبقه‌بندی شده‌اند، وزن بیشتری می‌دهد. به این ترتیب، مدل بعدی تمرکز بیشتری روی یادگیری "مثال‌های سخت" خواهد داشت. این کد نیز یک مدل AdaBoostClassifier را با تنظیمات پیش‌فرض آموزش می‌دهد.

معیار های ارزیابی برای این ماشین یادگیری:

```
train accuracy: 0.73
test accuracy: 0.69
recall: 0.42
precision: 0.80
f1 score: 0.55
confusion matrix: [[132  12]
 [ 69  49]]
```

```
xgb_clf = xgb.XGBClassifier(random_state=42, eval_metric='logloss')
xgb_clf.fit(X_train, y_train)
```

XGBoost (eXtreme Gradient Boosting) یک پیاده‌سازی بهینه‌سازی‌شده و بسیار کارآمد از الگوریتم گرادیان بوستینگ است. این مدل به دلیل سرعت بالا و عملکرد فوق‌العاده، یکی از محبوب‌ترین الگوریتم‌ها در مسابقات علم داده است. XGBoost شامل ویژگی‌هایی مانند تنظیم‌گری (regularization) برای جلوگیری از بیش‌برازش (overfitting) و قابلیت پردازش موازی است.

معیار های ارزیابی برای این ماشین یادگیری:

```
train accuracy: 0.96
test accuracy: 0.76
recall: 0.66
precision: 0.77
f1 score: 0.71
confusion matrix: [[121  23]
 [ 40  78]]
```

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15, 20],
    'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.3],
    'subsample': [0.6, 0.8, 1.0]
}
grid_search_xgb = GridSearchCV(xgb.XGBClassifier(random_state=42, eval_metric='logloss'), param_grid, cv=5)
grid_search_xgb.fit(X_train, y_train)
xgb_clf = grid_search_xgb.best_estimator_
```

این بخش، مشابه کاری که برای جنگل تصادفی انجام شد، به دنبال یافتن بهترین ترکیب ابرپارامترها برای مدل XGBoost است.

param_grid: یک شبکه جستجوی بسیار بزرگ‌تر تعریف شده است که شامل پارامترهای جدیدی است:

learning_rate (نرخ یادگیری): کنترل می‌کند که هر درخت جدید چقدر می‌تواند خطاهای قبلی را اصلاح کند. مقادیر کمتر، فرآیند یادگیری را کندتر اما معمولاً پایدارتر می‌کنند.

subsample (زیرنمونه): مشخص می‌کند که چه کسری از داده‌های آموزشی برای ساخت هر درخت استفاده شود. این کار به جلوگیری از بیش‌برازش کمک می‌کند.

GridSearchCV(...): جستجوی شبکه‌ای را با مدل XGBoost، پارامترهای تعریف‌شده و اعتبارسنجی متقابل ۵-بخشی (cv=5) راه‌اندازی می‌کند.

grid_search_xgb.fit(...): فرآیند جستجو را اجرا می‌کند. با توجه به تعداد زیاد ترکیبات، این مرحله می‌تواند بسیار زمان‌بر باشد.

xgb_clf = grid_search_xgb.best_estimator_: پس از اتمام جستجو، بهترین مدل یافت‌شده استخراج و در متغیر xgb_clf جایگزین می‌شود.

در نهایت، این مدل بهینه‌شده برای پیش‌بینی نهایی استفاده می‌شود تا عملکرد آن پس از فرآیند بهبود، ارزیابی گردد.

```
train accuracy: 0.85
test accuracy: 0.74
recall: 0.55
precision: 0.81
f1 score: 0.66
confusion matrix: [[129  15]
 [ 53  65]]
```

سوال پنجم

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score as sk_silhouette_score
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
import random

df = pd.read_csv('spotify.csv')
df.head()
```

در این بخش، کتابخانه‌های لازم فراخوانی و داده‌ها برای تحلیل آماده می‌شوند.

فایل spotify.csv که حاوی اطلاعات آهنگ‌هاست، در یک دیتافریم به نام df بارگذاری می‌شود.

```
features = ['danceability', 'energy', 'loudness', 'speechiness', 'acousticness',
            'instrumentalness', 'liveness', 'valence', 'tempo']
data = df[['track_name'] + features].dropna()
data.head()
```

انتخاب ویژگی‌ها: برای خوشه‌بندی آهنگ‌ها، باید آن‌ها را بر اساس مشخصات موسیقایی‌شان گروه‌بندی کرد. در اینجا، لیستی از ویژگی‌های عددی مانند danceability (قابلیت رقص)، energy

(انرژی)، loudness (بلندی صدا) و غیره انتخاب شده‌اند. ستون‌های غیرعددی مانند نام خواننده کنار گذاشته شده‌اند، زیرا برای محاسبات خوشه‌بندی مناسب نیستند.

یک دیتافریم جدید به نام data ساخته می‌شود که فقط شامل نام آهنگ و ویژگی‌های انتخاب‌شده است. دستور .dropna() هر آهنگی که در یکی از این ویژگی‌ها مقدار خالی داشته باشد را حذف می‌کند تا داده‌ها کاملاً تمیز و آماده تحلیل باشند.

	track_name	danceability	energy	loudness	speechiness	acousticness	instrumentalness	liveness	valence	tempo
0	I Don't Care (with Justin Bieber) - Loud Luxur...	0.748	0.916	-2.634	0.0583	0.1020	0.000000	0.0653	0.518	122.036
1	Memories - Dillon Francis Remix	0.726	0.815	-4.969	0.0373	0.0724	0.004210	0.3570	0.693	99.972
2	All the Time - Don Diablo Remix	0.675	0.931	-3.432	0.0742	0.0794	0.000023	0.1100	0.613	124.008
3	Call You Mine - Keanu Silva Remix	0.718	0.930	-3.778	0.1020	0.0287	0.000009	0.2040	0.277	121.956
4	Someone You Loved - Future Humans Remix	0.650	0.833	-4.672	0.0359	0.0803	0.000000	0.0833	0.725	123.976

```python
def standard_scalar(data):
    # Calculate mean and standard deviation for each column
    means = np.mean(data, axis=0)
    stds = np.std(data, axis=0)
    # Avoid division by zero
    stds = np.where(stds == 0, 1, stds)
    # Standardize the data: (x - mean) / std
    scaled_data = (data - means) / stds
    return scaled_data

# Apply standard scaler to the numerical features
X = data[features].values
X_scaled = standard_scalar(X)
```

الگوریتم‌های خوشه‌بندی به مقیاس داده‌ها حساس هستند. برای مثال، ویژگی tempo (ضرب‌آهنگ) مقادیری بین ۶۰ تا ۲۰۰ دارد، در حالی که danceability بین ۰ و ۱ است. این تفاوت بزرگ باعث می‌شود الگوریتم به tempo وزن بیشتری بدهد. برای حل این مشکل، داده‌ها را استاندارد می‌کنیم.

تابع standard_scalar: این تابع به صورت دستی، فرآیند استانداردسازی را پیاده‌سازی می‌کند. در این روش، برای هر مقدار، میانگین مقادیر ستون از آن کم شده و نتیجه بر انحراف معیار آن ستون تقسیم می‌شود.

فرمول: انحراف معیار / (مقدار - میانگین)

این کار باعث می‌شود هر ویژگی دارای میانگین ۰ و انحراف معیار ۱ شود و همه ویژگی‌ها مقیاس یکسانی پیدا کنند.

اعمال استانداردسازی: ابتدا مقادیر عددی ویژگی‌ها از دیتافریم data استخراج و در یک آرایه نام‌پای به نام X ریخته می‌شوند.

سپس تابع standard_scalar روی این آرایه اعمال شده و نتیجه در X_scaled ذخیره می‌شود. این آرایه X_scaled ورودی نهایی ما برای الگوریتم خوشه‌بندی خواهد بود.

```
def reduce_dimension(embedding, n_components):
    """
    Performs dimensional reduction using PCA with n components left behind

    Parameters
    ----------
    embeddings : List
        A list of embeddings of documents

    n_components: int
        Number of components to keep

    Returns a list of reduced embeddings
    """
    pca = PCA(n_components=n_components)
    reduced_embeddings = pca.fit_transform(embedding)
    return reduced_embeddings, pca
```

داده‌های ما در حال حاضر ۹ بُعد (ویژگی) دارند. کار کردن با ابعاد بالا می‌تواند محاسبات را پیچیده کرده و نتایج را تحت تأثیر قرار دهد. PCA به ما کمک می‌کند تا این ۹ ویژگی را به تعداد کمتری "مؤلفه اصلی" تبدیل کنیم، در حالی که بیشترین اطلاعات ممکن از داده‌های اصلی حفظ شود.

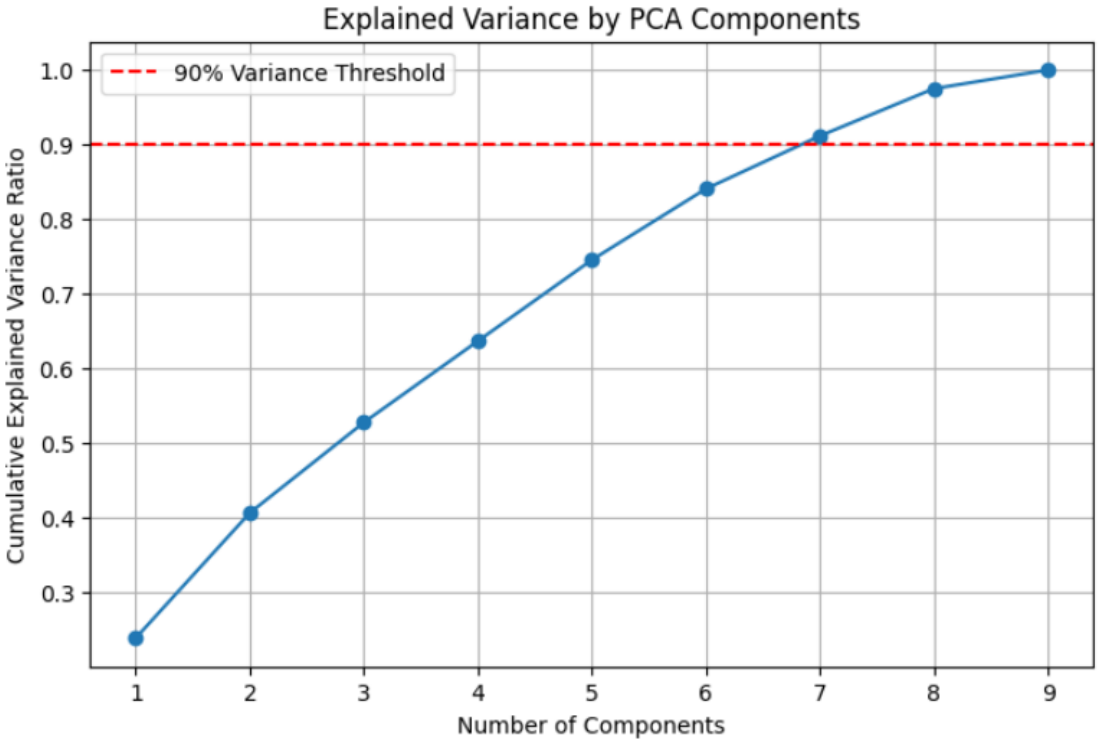
قبل از اینکه ابعاد را کم کنیم، باید تصمیم بگیریم که چند مؤلفه را نگه داریم. این کد به ما در این تصمیم کمک می‌کند.

pca_full = PCA().fit(X_scaled): یک مدل PCA روی تمام داده‌های استاندارد شده (X_scaled) اجرا می‌شود، اما هنوز هیچ کاهشی صورت نمی‌گیرد. هدف این خط، تحلیل ویژگی‌های داده است.

explained_variance_ratio_: این ویژگی به ما نشان می‌دهد که هر کدام از مؤلفه‌های اصلی جدید، چند درصد از واریانس (پراکندگی اطلاعات) کل داده‌ها را توضیح می‌دهند.

np.cumsum(...): این دستور، واریانس تجمعی را محاسبه می‌کند. یعنی واریانس مؤلفه اول، سپس مجموع واریانس اول و دوم، سپس مجموع سه تای اول و الی آخر.

رسم نمودار: نمودار حاصل، واریانس تجمعی را بر حسب تعداد مؤلفه‌ها نمایش می‌دهد. این نمودار به ما کمک می‌کند تا ببینیم با اضافه کردن هر مؤلفه، چقدر به حفظ اطلاعات کل نزدیک‌تر می‌شویم. خط قرمز افقی در نمودار، آستانه ۹۰٪ را نشان می‌دهد که یک استاندارد رایج است.




```
n_components = np.argmax(cumulative_variance >= 0.9) + 1
print(f'Minimum number of components to retain 90% variance: {n_components}')
```

در این قسمت، به صورت خودکار بهترین تعداد مؤلفه‌ها را پیدا می‌کنیم.

مقدار آستانه (Cutoff Value): مقدار آستانه در اینجا 0.9 یا ۹۰٪ انتخاب شده است.

توضیح: این آستانه یک نقطه تعادل است. ما می‌پذیریم که ۱۰٪ از اطلاعات جزئی و کم‌اهمیت‌تر داده‌ها را از دست بدهیم، اما در عوض، پیچیدگی مدل را به شدت کاهش دهیم (مثلاً از ۹ بُعد به ۵ یا ۶ بُعد برسیم). این کار به الگوریتم خوشه‌بندی کمک می‌کند تا روی الگوهای مهم‌تر تمرکز کند و نتایج بهتری تولید کند.

np.argmax(cumulative_variance >= 0.9) + 1: این کد به طور هوشمندانه عمل می‌کند. ابتدا یک لیست بولی (True/False) می‌سازد که نشان می‌دهد در کدام نقاط واریانس تجمعی از ۹۰٪ بیشتر شده است. np.argmax ایندکس اولین True را برمی‌گرداند و با اضافه کردن عدد ۱ به آن، کوچک‌ترین تعداد مؤلفه‌ای که برای رسیدن به آستانه ۹۰٪ لازم است، به دست می‌آید.

```
X_reduced, pca = reduce_dimension(X_scaled, n_components)

# For visualization, reduce to 2 components
X_2d, _ = reduce_dimension(X_scaled, 2)
```

```
Minimum number of components to retain 90% variance: 7
```

حالا که تعداد بهینه مؤلفه‌ها (n_components) را پیدا کردیم، کاهش ابعاد را انجام می‌دهیم.

تابع reduce_dimension: این یک تابع کمکی است که فرآیند اجرای PCA و تبدیل داده‌ها را در خود کپسوله کرده تا کد تمیزتر باشد.

X_reduced: این مجموعه داده اصلی ما برای خوشه‌بندی خواهد بود. ابعاد آن از ۹ به n_components (مثلاً ۵ یا ۶) کاهش یافته است، در حالی که ۹۰٪ از اطلاعات اصلی را حفظ کرده است.

X_2d: این یک مجموعه داده جداگانه است که ابعاد آن فقط به ۲ کاهش یافته است. دلیل این کار صرفاً بصری‌سازی (Visualization) است. ما نمی‌توانیم داده‌ها را در فضای ۵ یا ۶ بعدی رسم کنیم، اما می‌توانیم آن‌ها را به راحتی در یک نمودار ۲ بعدی نمایش دهیم تا ببینیم خوشه‌ها به صورت تقریبی کجا قرار گرفته‌اند.

```
def cluster_kmeans(emb_vecs, n_clusters):
    """
    Clusters input vectors using K-means method

    Parameters
```

پیاده‌سازی الگوریتم K-Means از ابتدا (cluster_kmeans)

این تابع، همانطور که در کامنت TODO خواسته شده، الگوریتم K-Means را به صورت دستی پیاده‌سازی می‌کند. K-Means یک الگوریتم تکراری است که تلاش می‌کند داده‌ها را به k (یا n_clusters) خوشه تقسیم کند.

```python
emb_matrix = np.array(emb_vecs)
n_samples = emb_matrix.shape[0]

# Randomly initialize centroidsnp.random.seed(42)
indices = np.random.choice(n_samples, n_clusters, replace=False)
initial_centroids = emb_matrix[indices]
centroids = initial_centroids.copy()

for _ in range(100):  # You can adjust the number of iterations
    # Compute distances between data points and centroids
    distances = np.sqrt(((emb_matrix - centroids[:, np.newaxis])**2).sum(axis=2))

    # Assign each point to the closest centroid
    cluster_indices = np.argmin(distances, axis=0)

    # Update centroids
    new_centroids = np.zeros_like(centroids)
    for i in range(n_clusters):
        points = emb_matrix[cluster_indices == i]
        if len(points) > 0:
            new_centroids[i] = points.mean(axis=0)
        else:
            new_centroids[i] = centroids[i]  # Keep old centroid if no points assigned
```

مقداردهی اولیه (Initialization)

ابتدا n_clusters نقطه از داده‌ها به صورت تصادفی به عنوان مراکز اولیه خوشه‌ها (Initial Centroids) انتخاب می‌شوند.

np.random.seed(42) استفاده شده تا این انتخاب تصادفی در هر بار اجرای کد، یکسان باشد و نتایج قابل تکرار باشند

حلقه تکرار (Iteration Loop)

الگوریتم در یک حلقه (اینجا تا ۱۰۰ بار) دو مرحله اصلی "تخصیص" و "به‌روزرسانی" را تکرار می‌کند.

تخصیص (Assignment Step)

distances فاصله اقلیدسی هر آهنگ تا هر یک از مراکز خوشه‌ها محاسبه می‌شود.

cluster_indices = np.argmin(...): برای هر آهنگ، نزدیک‌ترین مرکز خوشه پیدا شده و آن آهنگ به آن خوشه تخصیص داده می‌شود.

به‌روزرسانی (Update Step)

پس از اینکه تمام آهنگ‌ها به خوشه‌ها تخصیص داده شدند، مراکز خوشه‌ها باید به‌روز شوند.

برای هر خوشه، میانگین موقعیت تمام آهنگ‌های داخل آن محاسبه می‌شود. این میانگین به عنوان مرکز جدید آن خوشه در نظر گرفته می‌شود.

(اگر خوشه‌ای خالی بماند، مرکز آن تغییر نمی‌کند تا از خطا جلوگیری شود).

```
    # Check for convergence
    if np.all(centroids == new_centroids):
        break
    centroids = new_centroids

return centroids.tolist(), cluster_indices.tolist()
```

شرط همگرایی (Convergence)

پس از هر بار به‌روزرسانی، کد بررسی می‌کند که آیا مراکز خوشه‌ها نسبت به مرحله قبل تغییری کرده‌اند یا خیر.

اگر مراکز هیچ تغییری نکرده باشند، یعنی الگوریتم به پایداری (همگرایی) رسیده و دیگر نیازی به تکرار نیست، بنابراین حلقه با دستور break متوقف می‌شود.

خروجی:

در نهایت، تابع موقعیت نهایی مراکز خوشه‌ها و شماره خوشه‌ای که هر آهنگ به آن تعلق دارد را برمی‌گرداند.

```
def wss_score(emb_vecs, centroids, cluster_indices):
    emb_matrix = np.array(emb_vecs)
    centroids = np.array(centroids)
    wss = 0
    for i in range(len(centroids)):
        points = emb_matrix[np.array(cluster_indices) == i]
        if len(points) > 0:
            wss += ((points - centroids[i])**2).sum()
    return wss

def silhouette_score(emb_vecs, cluster_indices):
    # Use sklearn's silhouette_score explicitly
    return sklearn_silhouette_score(emb_vecs, cluster_indices) if len(set(cluster_indices)) > 1 else 0
```

wss_score (مجموع مربعات درون خوشه‌ای)

مفهوم: این معیار، فشردگی و تراکم خوشه‌ها را اندازه‌گیری می‌کند.

WSS (Within-Cluster Sum of Squares) مجموع مربعات فاصله هر نقطه تا مرکز خوشه‌ی خودش را محاسبه می‌کند.

تفسیر: مقدار کمتر WSS بهتر است و نشان می‌دهد که نقاط به مرکز خوشه خود نزدیک هستند و خوشه‌ها متراکم‌ترند.

silhouette_score (امتیاز سیلوعه)

مفهوم: این یک معیار پیشرفته‌تر است که دو چیز را همزمان می‌سنجد:

انسجام (Cohesion): هر نقطه چقدر به نقاط دیگر در خوشه خودش نزدیک است.

جدایی (Separation): هر نقطه چقدر از نقاط موجود در خوشه‌های همسایه دور است.

تفسیر: امتیاز سیلوئت مقداری بین ۱- تا ۱+ است.

نزدیک به ۱+: بهترین حالت؛ یعنی خوشه‌ها متراکم و به خوبی از هم جدا شده‌اند.

نزدیک به ۰: خوشه‌ها با هم همپوشانی دارند.

نزدیک به ۱-: بدترین حالت؛ یعنی نقاط به اشتباه در خوشه‌ها قرار گرفته‌اند.

نکته پیاده‌سازی: این تابع به درستی از پیاده‌سازی استاندارد و قابل اعتماد scikit-learn برای محاسبه این امتیاز استفاده می‌کند.

```
k_values = range(2, 11)
sil_scores = []
wss_scores = []

for k in k_values:
    centroids, cluster_indices = cluster_kmeans(X_reduced, k)
    sil_scores.append(silhouette_score(X_reduced, cluster_indices))
    wss_scores.append(wss_score(X_reduced, centroids, cluster_indices))
```

محاسبه امتیازات برای مقادیر مختلف k

ما می‌خواهیم تعداد خوشه‌ها را از ۲ تا ۱۰ آزمایش کنیم تا ببینیم کدام یک بهترین نتیجه را می‌دهد.

حلقه for k in k_values: این حلقه به ازای هر مقدار k (از ۲ تا ۱۰)، مراحل زیر را تکرار می‌کند:

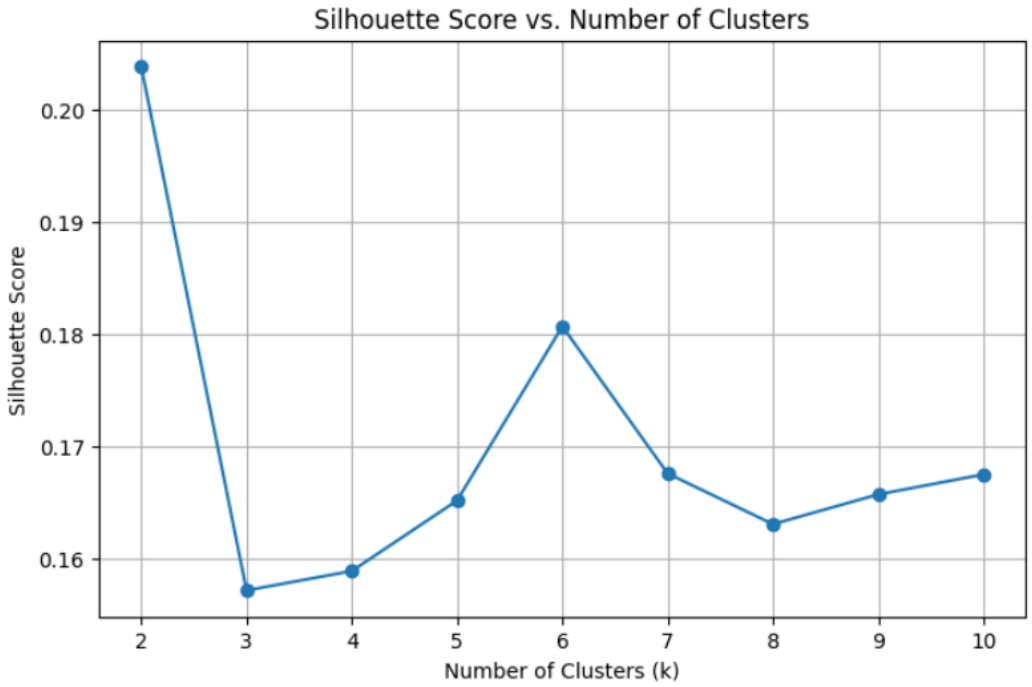
الگوریتم cluster_kmeans را روی داده‌های کاهش‌یافته (X_reduced) با تعداد خوشه k اجرا می‌کند.

امتیاز سیلوئت (silhouette_score) را برای خوشه‌بندی حاصل محاسبه و در لیست sil_scores ذخیره می‌کند.

امتیاز WSS (wss_score) را نیز محاسبه و در لیست wss_scores ذخیره می‌کند.

پس از پایان حلقه، ما دو لیست از امتیازات داریم که عملکرد الگوریتم را برای هر مقدار k نشان می‌دهند.

تحلیل نمودار امتیاز سیلوئت (Silhouette Score)



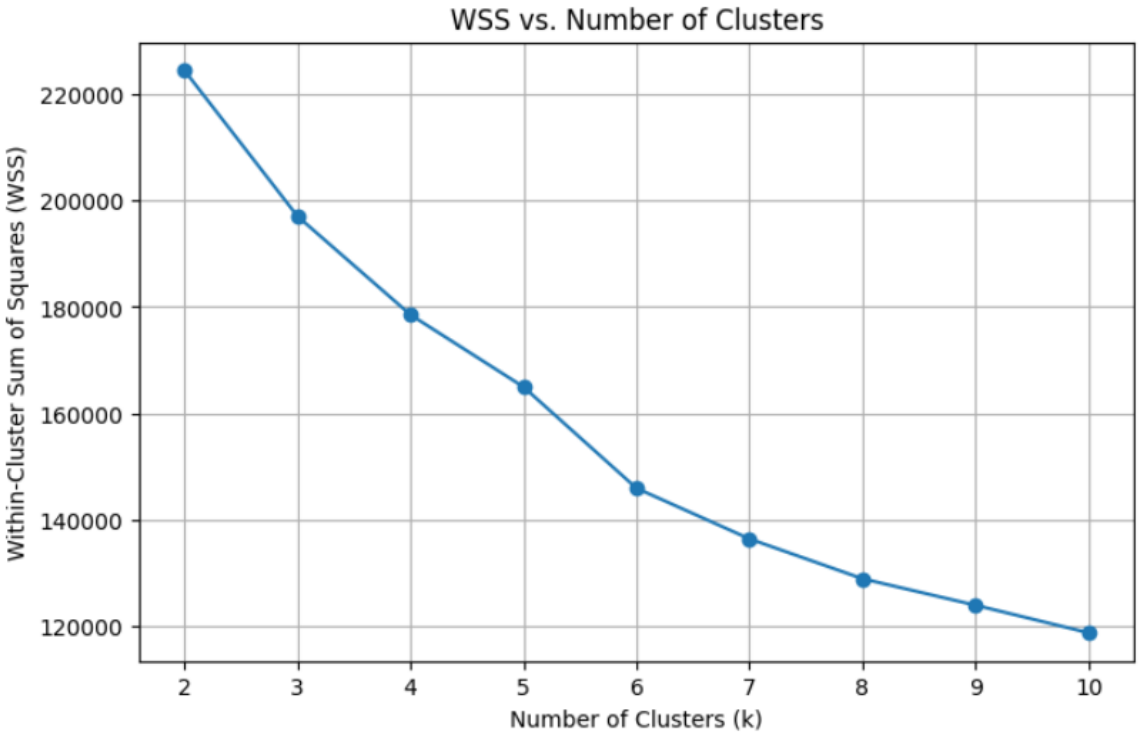
این نمودار امتیاز سیلوئت را بر حسب مقادیر مختلف k نمایش میدهد.

هدف: پیدا کردن تعداد خوشه‌ای که بهترین توازن بین انسجام و جدایی را دارد.

در این نمودار، ما به دنبال نقطه‌ای هستیم که بیشترین مقدار (قله نمودار) را دارد. هرچه امتیاز سیلوئت بالاتر باشد، ساختار خوشه‌بندی بهتر است. بنابراین، مقدار k که به بالاترین امتیاز سیلوئت منجر شود، به عنوان مقدار بهینه در نظر گرفته می‌شود.

که در اینجا k=2 بیشترین امتیاز را دارد.

تحلیل نمودار WSS (روش آرنج - The Elbow Method)



این کد نمودار امتیاز WSS را بر حسب مقادیر مختلف k رسم می‌کند. این روش به "روش آرنج" معروف است.

هدف: پیدا کردن تعداد خوشه‌ای که پس از آن، اضافه کردن خوشه‌های بیشتر، بهبود چشمگیری در فشردگی خوشه‌ها ایجاد نمی‌کند.

معیار WSS با افزایش k همیشه کاهش می‌یابد (چون خوشه‌ها بیشتر و کوچکتر می‌شوند). اما ما به دنبال نقطه‌ای هستیم که شیب نمودار به طور ناگهانی کم می‌شود و حالتی شبیه به "آرنج خم شده" پیدا می‌کند.

این نقطه آرنج (Elbow Point)، بهترین نقطه تعادل بین کم بودن تعداد خوشه‌ها و فشرده بودن آن‌ها را نشان می‌دهد. این نقطه، k بهینه بر اساس این معیار است.

یافتن نقطه آرنج (Elbow Point)

هدف ما پیدا کردن نقطه‌ای است که نمودار در آن مانند یک "آرنج" خم می‌شود. این نقطه، بهترین تعادل را بین کم بودن تعداد خوشه‌ها و فشرده بودن آن‌ها نشان می‌دهد.

در این نمودار، واضح‌ترین نقطه آرنج در k=4 رخ داده است.

دلیل: افت WSS از k=3 به k=4 بسیار قابل توجه است. اما بعد از آن، از k=4 به k=5 و بعدتر، کاهش WSS بسیار کمتر می‌شود. به عبارت دیگر، اضافه کردن خوشه پنجم، ششم و... دیگر تاثیر چشمگیری در بهبود تراکم خوشه‌ها ندارد و ما به نقطه "بازده نزولی" (diminishing returns) رسیده‌ایم.

می‌توان یک "آرنج" ضعیف‌تر و ثانویه را نیز در k=6 مشاهده کرد که پس از آن، نمودار تقریباً به یک خط صاف تبدیل می‌شود.

نمونه‌گیری و مقایسه آهنگ‌های درون خوشه‌ها

پس از اینکه مشخص شد بهترین تعداد خوشه k=4 است، در این بخش خوشه‌بندی نهایی را با همین مقدار انجام داده و نتایج را بررسی می‌کنیم.

```
best_k = 4
centroids, cluster_indices = cluster_kmeans(X_reduced, best_k)

# Add cluster labels to the dataframe
data['cluster'] = cluster_indices

# Sample 2 songs from each cluster
for i in range(best_k):
    cluster_songs = data[data['cluster'] == i]['track_name'].sample(2, random_state=42)
    print(f'Cluster {i} sample songs: {list(cluster_songs)}')

# To check if songs are close, we can compare their feature values
for i in range(best_k):
    cluster_data = data[data['cluster'] == i]
    sample_indices = cluster_data.sample(2, random_state=42).index
    sample_features = data.loc[sample_indices, features]
    distance = np.linalg.norm(sample_features.iloc[0] - sample_features.iloc[1])
    print(f'Cluster {i} Euclidean distance between samples: {distance:.4f}')
```

این کد دو نوع بررسی را برای درک بهتر ماهیت خوشه‌ها انجام می‌دهد:

بررسی کیفی (چاپ نام آهنگ‌ها):

ابتدا شماره خوشه‌ای که هر آهنگ به آن تعلق دارد، به عنوان یک ستون جدید به دیتافریم data اضافه می‌شود.

سپس از هر یک از ۴ خوشه، ۲ آهنگ به صورت تصادفی انتخاب و نام آن‌ها چاپ می‌شود.

هدف: این کار یک درک شهودی به ما می‌دهد. با دیدن نام آهنگ‌هایی که در یک گروه قرار گرفته‌اند، می‌توانیم حدس بزنیم که هر خوشه چه سبک یا "شخصیت" موسیقایی دارد (مثلاً خوشه آهنگ‌های آرام، خوشه آهنگ‌های پرانرژی و غیره).

بررسی کمّی (محاسبه فاصله):

در حلقه دوم، دوباره دو آهنگ تصادفی از هر خوشه انتخاب می‌شود، اما این بار فاصله اقلیدسی بین بردارهای ویژگی آن‌ها محاسبه می‌شود.

هدف: این یک تأیید عددی است. فاصله کم بین دو آهنگ به این معنی است که ویژگی‌های موسیقایی آن‌ها (مانند danceability, energy, loudness) بسیار به هم شبیه است. این کار به ما اطمینان می‌دهد که الگوریتم، آهنگ‌های واقعاً مشابه را در یک گروه قرار داده است.

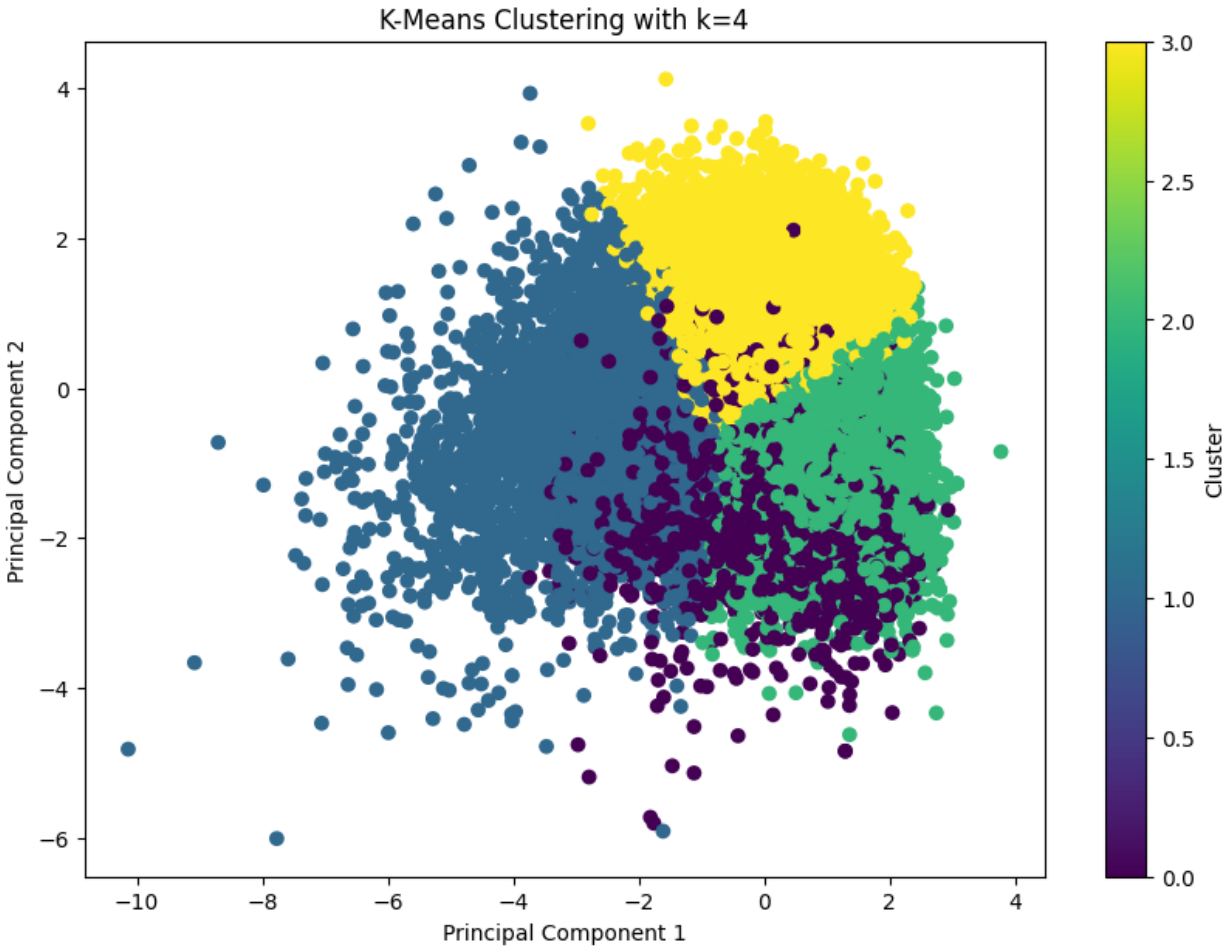
```
Cluster 0 sample songs: ['Take Off', 'A Flicker Of Hope']
Cluster 1 sample songs: ['Big White Room - Live', 'On & On']
Cluster 2 sample songs: ['Gassed Up', 'I Need You (feat. Olaf Blackwood) - DubVision Remix']
Cluster 3 sample songs: ['B.L.O.W.', 'In My Blood']
Cluster 0 Euclidean distance between samples: 7.2341
Cluster 1 Euclidean distance between samples: 17.6178
Cluster 2 Euclidean distance between samples: 14.3069
Cluster 3 Euclidean distance between samples: 26.8267
```

در نهایت، نتایج خوشه‌بندی را در یک نمودار دو بعدی به تصویر می‌کشیم.

هدف: ارائه یک نمای بصری از نحوه گروه‌بندی آهنگ‌ها.

این نمودار از داده‌های X_2d استفاده می‌کند که قبلاً فقط برای همین منظور (بصری‌سازی) آماده شده بود. محور افقی مؤلفه اصلی اول و محور عمودی مؤلفه اصلی دوم است.

مهم‌ترین بخش c=cluster_indices است. این آرگومان به پلات می‌گوید که هر نقطه (آهنگ) را بر اساس شماره خوشه‌ای که به آن تعلق دارد، رنگ‌آمیزی کند. 'cmap='viridis نیز پالت رنگی را مشخص می‌کند.



خوشه‌بندی انجام شده تا حد زیادی موفق و معنادار است. اگرچه مرزهای کاملاً تفکیک‌شده‌ای بین تمام گروه‌ها وجود ندارد، اما الگوریتم K-Means توانسته است:

یک گروه بسیار خاص و متمایز از آهنگ‌ها را شناسایی کند (خوشه زرد).

سایر آهنگ‌ها را در چندین گروه بزرگ‌تر و عمومی‌تر دسته‌بندی کند که این خود نشان‌دهنده طبیعت پیچیده و درهم‌تنیده موسیقی است.

این نمودار به خوبی تأیید می‌کند که الگوریتم توانسته ساختار پنهان در داده‌ها را کشف کرده و آهنگ‌ها را بر اساس شباهت‌های موسیقایی‌شان به گروه‌های منطقی تقسیم کند.