سوال ۱ — تابع هدف K-Means و فرمول مرکز خوشه

چرا مرکز خوشه = میانگین نقاط؟

می‌خواهیم:

- این داده‌ها را به **K گروه (خوشه)** تقسیم کنیم
- طوری که نقاط **هر خوشه تا حد ممکن به هم نزدیک باشند(نزدیک بودن = فاصله کم)**

تعریف تابع هدف (Objective Function)

حالا کل ایده K-Means در **یک فرمول** خلاصه می‌شود:

$$\boxed{J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2}$$

یه نقطه رو به‌عنوان مرکز هر گروه بگیر
بعد ببین مجموع فاصله‌ی همه‌ی نقطه‌ها تا این مرکز چقدر میشه
حالا مرکز رو جوری جابه‌جا کن که این مجموع فاصله **کمترین مقدار ممکن** باشه»

این «کمترین کردن» همون چیزیه که اسمش هست:
تابع هدف:

- برای هر خوشه
- فاصله تمام نقاطش تا مرکز خوشه را حساب کن
- مربع کن
  - **چرا مربع؟**
  - فاصله منفی نمی‌شه
  - نقاط خیلی دور «خیلی بدتر» حساب می‌شن
  - محاسبه‌اش راحت‌تره

- جمع بزن
- کل این عدد را **کمینه کن**

این همان چیزی است که در کد sklearn می‌بینی به اسم:

inertia = WCSS

در K-Means:

عدد بزرگ → خوشه‌بندی بد

عدد کوچک → خوشه‌بندی خوب

تمرکز روی یک خوشه (تفکیک مسئله)

فرض کن:

- خوشه‌ی kام را داریم
- نقاطش:

$$C_k = \{x_1, x_2, \ldots, x_m\}$$

تابع هدف فقط برای این خوشه:

$$J_k(\mu_k) = \sum_{i=1}^{m} \|x_i - \mu_k\|^2$$

اینجا فقط **μ_k** متغیر است. نقاط ثابت‌اند

$$(x_i - \mu)^2 = x_i^2 - 2x_i\mu + \mu^2$$

$$J(\mu) = \sum_{i=1}^{m} \left(x_i^2 - 2x_i\mu + \mu^2\right)$$

$$J(\mu) = \sum_{i=1}^{m} x_i^2 \; - \; 2\mu \sum_{i=1}^{m} x_i \; + \; \sum_{i=1}^{m} \mu^2$$

μ2 برای همه‌ی iها یکیه

$$J(\mu) = \underbrace{\sum_{i=1}^{m} x_i^2}_{\text{ثابت نسبت به } \mu} - 2\mu \sum_{i=1}^{m} x_i + m\mu^2$$

قسمت‌های مهم همون‌هایی‌اند که μ\muμ توشون هست
برای کمینه کردن یک تابع، معمولاً مشتق می‌گیریم و برابر صفر می‌ذاریم:

$$\frac{dJ}{d\mu} = 0 - 2\sum_{i=1}^{m} x_i + 2m\mu$$

$$m\mu = \sum_{i=1}^{m} x_i$$

$$\boxed{\mu = \frac{1}{m}\sum_{i=1}^{m} x_i}$$

مثلاً اگر داده 2بعدی باشد:

$$x_i = (x_{i1}, x_{i2})$$

آن‌وقت:

$$\mu = \left(\frac{1}{m}\sum x_{i1},\ \frac{1}{m}\sum x_{i2}\right)$$

ارتباط با K-Means در عمل

K-Means دو کار را تکرار می‌کند:

1. **Assign**: هر نقطه میره نزدیک‌ترین مرکز
2. **Update**: برای هر خوشه، مرکز جدید = میانگین نقاط خوشه (همین چیزی که اثبات کردیم)

و چون در مرحله 2 واقعاً بهترین مرکز را برای خوشه‌ی فعلی انتخاب می‌کنیم، مقدار هزینه کم می‌شود.

2) روش Elbow در K-Means

«چطور K مناسب رو انتخاب کنیم؟»

چرا Elbow لازم داریم؟

در K-Means یک مشکل اساسی داریم:

K (تعداد خوشه‌ها) را از کجا بفهمیم؟

- K-Means خودش K رو پیدا نمی‌کنه
- ما باید قبل از اجرا بگیم K چند تاست

در روش Elbow دقیقاً چه چیزی رسم می‌شود؟

محور افقی (x-axis):

K=1,2,3...

یعنی:

تعداد خوشه‌ها

محور عمودی (y-axis):

WCSS یا Inertia

یعنی:

- مجموع مربع فاصله نقاط تا مرکز خوشه‌ها
- همان تابع هدف K-Means

در کد: `kmeans.inertia_`

WCSS همیشه با افزایش K کم می‌شود

وقتی WCSS را برحسب K رسم می‌کنیم، معمولاً انتظار داریم:

- اول: افت شدید
- بعد: افت ملایم
- نمودار شبیه یک **بازوی خم‌شده (Elbow)**

اسم روش Elbow دقیقاً از همین شکل آمده.

چرا این روش دقیق ریاضی نیست؟

چون:

- «آرنج» تعریف ریاضی دقیق ندارد
- بیشتر یک **قضاوت بصری** است
- دو نفر ممکن است دو K متفاوت انتخاب کنند

به همین دلیل معمولاً:

- Elbow + Silhouette
- یا Elbow + دانش مسئله

چه زمانی نمودار Elbow آرنج واضح ندارد؟

اگر Elbow واضح نباشد، این‌ها را می‌توان نتیجه گرفت:

- یا داده ذاتاً خوشه ندارد
- یا K-Means فرض مناسبی برای داده نیست
- یا باید از معیارهای دیگر کمک گرفت
- یا تعداد خوشه‌ها «ذاتی» نیست

داده ساختار خوشه‌ای واضح ندارد(داده تقریباً یکنواخت باشدا خوشه‌ها خیلی در هم باشند)

خوشه‌ها هم‌اندازه و خیلی مشابه‌اند(همه خوشه‌ها تراکم مشابه دارند)

داده پیچیده یا غیر کروی است(اگر داده: هلالی و کشیده و یا با نویز زیاد

WCSS رفتار روشنی نشان نمی‌دهد)

ابعاد داده زیاد است (فاصله‌ها شبیه هم می‌شوند)

سوال ۳ — چرا به Hungarian نیاز داریم؟

ما دو چیز متفاوت داریم:

1) خروجی K-Means

- برچسب‌ها: 0, 1, 2, ...
- این اعداد **فقط اسم‌اند**
- هیچ معنایی ندارند

مثلاً:

- خوشه‌ی 0 ← می‌تواند «عدد 7» باشد

2) برچسب‌های واقعی (Ground Truth)

- مثلاً در دیتاست digits:
  - کلاس 0 یعنی عدد «0»
  - کلاس 1 یعنی عدد «1»
- این‌ها **معنا دارند**

خطاها؟

1) خطای مفهومی

چون:

- فرض میکند شماره‌ی خوشه معنا دارد
- در حالی که ندارد

2) خطای عددی / ارزیابی

چون:

- عدد Accuracy یا Precision کاملاً گمراه‌کننده می‌شود
- مدل خوب، بد گزارش می‌شود

Hungarian Algorithm چی کار می‌کند؟

بهترین نگاشت یک‌به‌یک بین خوشه‌ها و کلاس‌ها را پیدا می‌کند
طوری که مجموع خطا کمینه (یا تطابق بیشینه) شود

یعنی:

- خوشه‌ها را **بهترین‌طور ممکن** به کلاس‌ها نگاشت کنیم

نقش Hungarian در ارزیابی دقیقاً چیست؟

1) حذف اثر جابه‌جایی برچسب‌ها

- دیگر مهم نیست خوشه 0 باشد یا 7
- فقط تطابق واقعی مهم است

2) ایجاد یک معیار عادلانه

بعد از نگاشت:

- Accuracy معنی‌دار می‌شود
- Precision/Recall قابل تفسیر می‌شوند

نگاشت (mapping) یعنی:

این تصمیم که بگیم:

- خوشه‌ی 0 ↔ کلاس 7
- خوشه‌ی 1 ↔ کلاس 2
- خوشه‌ی 2 ↔ کلاس 9

یعنی: هر خوشه نماینده‌ی کدام کلاس واقعی فرض شود؟
این نگاشت **از قبل معلوم نیست** و باید پیدا شود.

اگر Hungarian نباشد چه می‌شود؟

- ممکن است نگاشت اشتباه انتخاب شود
- Accuracy خیلی پایین گزارش شود
- در حالی که خوشه‌بندی در واقع خوب بوده

یعنی:

خطای ارزیابی، نه خطای مدل

Hungarian کمک می‌کند بفهمیم اگر هر خوشه را به «بهترین» کلاس ممکن وصل کنیم، عملکرد واقعی خوشه‌بندی چقدر خوب بوده است.

اگر اشتباه مقایسه کنیم چه می‌شود؟

اگر بگیم:

- خوشه 0 ↔ گربه
- خوشه 1 ↔ سگ

آن‌وقت:

- همه داده‌ها غلط حساب می‌شوند
- Accuracy = صفر

در حالی که مدل بی‌نقص کار کرده!

پس راه درست چیه؟

باید اول این سؤال رو جواب بدیم:

هر خوشه بیشتر شبیه کدوم کلاس واقعیه؟

یعنی:

- خوشه‌ای که بیشتر گربه داره → گربه
- خوشه‌ای که بیشتر سگ داره → سگ

سؤال ۴ — ARI و NMI خیلی ساده و مفهومی

ما داریم خوشه‌بندی بدون ناظر انجام می‌دیم، ولی:

- **برچسب واقعی داریم (برای ارزیابی)**
- **Accuracy به تنهایی قابل اعتماد نیست**
- **شماره‌ی خوشه‌ها معنا ندارن**

ARI دقیقاً چی رو اندازه می‌گیره؟

آیا الگوریتم، جفت‌داده‌ها را مثل واقعیت کنار هم گذاشته یا جدا کرده؟»نتیجه رو نسبت به شانس تصحیح می‌کنه

بازه‌ی ARI:

- **1 → تطابق کامل**
- **0 → تقریباً تصادفی**
- **منفی → بدتر از تصادفی**

یعنی چی؟

به هر **دو نقطه** نگاه می‌کنه و می‌گه:

- آیا در واقعیت با هم بودن؟
- آیا در خوشه‌بندی هم با هم افتادن؟

NMI دقیقاً چی رو اندازه می‌گیره؟

NMI می‌پرسه:

«چقدر اطلاعات خوشه‌بندی، درباره‌ی کلاس‌های واقعی به من می‌ده؟»

دقت کن:

- **نمی‌پرسه لیبل خوشه چند است**
- **می‌پرسه رابطه‌ی بین دو دسته‌بندی چطوره**

بازه‌ی NMI:

- **0** → هیچ اطلاعات مشترکی
- **1** → تطابق کامل

یعنی:

- **اگر برچسب خوشه رو بدونم**
- **چقدر می‌تونم حدس بزنم کلاس واقعی چی بوده؟**

حساسیت به جابه‌جایی برچسب‌ها

سؤال: اگر شماره‌ی خوشه‌ها عوض شود چه می‌شود؟

مثلاً:

- خوشه 0 ↔ خوشه 3
- خوشه 1 ↔ خوشه 7

پاسخ:

- **ARI: کاملاً پایدار** ✅
- **NMI: کاملاً پایدار** ✅

هیچ‌کدام به «اسم خوشه» نگاه نمی‌کنند

فقط به **روابط بین داده‌ها** نگاه می‌کنند

چرا ممکن است ARI و NMI برای یک خوشه‌بندی فرق زیادی داشته باشند؟

تعداد خوشه‌ها با تعداد کلاس‌ها فرق دارد

مثلاً:

- کلاس واقعی: 3 تا
- خوشه‌بندی: 6 تا خوشه (ریزتر)

◆ NMI:

- ممکنه هنوز بالا باشه
- چون اطلاعات کلاس‌ها تا حدی حفظ شده

🔹 ARI:

- ممکنه پایین بیاد
- چون خیلی از **جفت‌ها** برخلاف واقعیت جدا شده‌اند

تفسیر:

خوشه‌بندی اطلاعات خوبی دارد، ولی بیش‌ازحد خوشه‌سازی کرده

خوشه‌ها هم‌پوشانی دارند

- بعضی داده‌ها مرزی‌اند
- خوشه‌بندی کاملاً تمیز نیست

🔹 ARI:

- حساس‌تر به این اشتباهات
- سریع‌تر افت می‌کند

🔹 NMI:

- ممکن است هنوز متوسط یا بالا بماند

📌 تفسیر:

ساختار کلی حفظ شده، ولی مرزها دقیق نیستند

کلاس‌ها نامتوازن هستند

مثلاً:

- یک کلاس خیلی بزرگ
- چند کلاس خیلی کوچک

🔹 ARI:

- به روابط جفتی حساس است
- ممکن است شدیداً تحت‌تأثیر کلاس بزرگ قرار گیرد

🔹 NMI:

- متوازن‌تر رفتار می‌کند

اگر NMI بالا و ARI پایین بود:

- معمولاً یعنی **Over-clustering** یا حفظ "اطلاعات کلی" ولی خراب بودن روابط دقیق
- خوشه‌ها خالص‌اند، اما یک کلاس را تکه‌تکه کرده‌ای

اگر ARI بالا و NMI پایین (کمتر رایج، ولی ممکن):

- می‌تواند یعنی برخی روابط جفتی خوب حفظ شده ولی توزیع اطلاعات بین خوشه/کلاس خوب نیست
- یا تعداد خوشه‌ها خیلی کم/زیاد و توزیع‌ها نامناسب است
  (در عمل این حالت کمتر از "NMI بالا، ARI پایین" دیده می‌شود)

اگر هر دو پایین:

- خوشه‌بندی واقعاً با کلاس‌ها همخوان نیست
- یا داده خوشه‌پذیر نیست
- یا الگوریتم/فاصله/اسکیلینگ مناسب نیست

۵)تفاوت Accuracy با ARI و NMI

«چرا Accuracy برای خوشه‌بندی مناسب نیست؟»

ما داریم درباره‌ی **خوشه‌بندی بدون ناظر (Clustering)** صحبت می‌کنیم،
ولی **برچسب واقعی** هم داریم (برای ارزیابی).

حالا سه معیار داریم:

- Accuracy
- ARI
- NMI

سؤال اینه:

چرا Accuracy معیار خوبی نیست،
ولی ARI و NMI مناسب‌ترند؟

Accuracy می‌پرسد:

«چند درصد نمونه‌ها **دقیقاً همان برچسب درست** را گرفته‌اند؟»

$$Accuracy = \frac{\text{تعداد پیش‌بینی درست}}{\text{کل نمونه‌ها}}$$

نکته‌ی خیلی مهم:

Accuracy فرض می‌کند:

- **عدد برچسب مهم است**
- کلاس 0 یعنی 0
- کلاس 1 یعنی 1

📌 این فرض در **Classification** درست است
📌 اما در **Clustering** **کاملاً غلط است**

چرا Accuracy برای خوشه‌بندی بدون ناظر مناسب نیست؟

در خوشه‌بندی:

شماره‌ی خوشه‌ها **دلخواه و بدون معنا** هستند

یعنی:

- خوشه 0 می‌توانست خوشه 5 باشد
- خوشه 1 می‌توانست خوشه 2 باشد

حتی با Hungarian هم Accuracy محدود است

فرض کنیم Hungarian استفاده کردیم و:

- بهترین نگاشت خوشه ↔ کلاس را پیدا کردیم

حالا Accuracy عدد معقولی می‌دهد،
اما باز هم یک مشکل بزرگ دارد

Accuracy فقط می‌پرسد:

«در نهایت چند تا درست شد؟»

اما نمی‌پرسد:

- ساختار خوشه‌بندی چقدر خوب بوده؟
- آیا خوشه‌ها بیش‌ازحد ریز شده‌اند؟
- آیا داده‌ها تصادفی کنار هم افتاده‌اند؟

ARI چه ویژگی خاصی دارد؟

«آیا **روابط بین جفت‌داده‌ها** درست حفظ شده یا نه؟»

ویژگی مهم ARI

خوشه‌بندی تصادفی → ARI ≈ 0

بیش‌ازحد خوشه‌سازی → جریمه می‌شود

NMI چه ویژگی خاصی دارد؟

«دانستن خوشه‌ها، چقدر درباره‌ی کلاس‌های واقعی به ما اطلاعات می‌دهد؟»

اگر هر خوشه تقریباً فقط یک کلاس باشد → NMI بالا

اگر خوشه‌ها قاطی باشند → NMI پایین

ویژگی مهم NMI

- نسبت به جابه‌جایی برچسب‌ها پایدار است
- به ساختار کلی داده حساس است
- به عدد برچسب‌ها کاری ندارد

اگر فقط به Accuracy تکیه کنیم چه سوءبرداشت‌هایی ایجاد می‌شود؟

سوءبرداشت 1

خوشه‌بندی خوب است
در حالی که فقط برچسب‌ها اتفاقی همنام شده‌اند

سوءبرداشت 2

خوشه‌بندی بد است
در حالی که فقط شماره خوشه‌ها جابه‌جا شده‌اند

سوءبرداشت 3

مدل ساده بهتر از مدل پیچیده است
چون Accuracy بیشتری دارد
(در حالی که ساختار داده را نابود کرده)

۶)تفاوت مفهومی K-Means ،DBSCAN و Hierarchical Clustering

K-Means → «تقسیم‌بندی بر اساس فاصله تا مرکز»

فرض اصلی:

خوشه‌ها **کروی (گرد)** هستند

همه‌ی خوشه‌ها اندازه و تراکم تقریباً مشابه دارند

چرا؟

چون از فاصله اقلیدسی تا مرکز (میانگین) استفاده می‌کند

نتیجه:

برای خوشه‌های کشیده، هلالی، پیچ‌خورده → بد عمل می‌کند

حتماً باید K را از قبل بدانیم

اگر K را اشتباه انتخاب کنیم:

- نتیجه کاملاً عوض می‌شود
- **بسیار حساس به مقیاس**
- حتماً نیاز به:
  - normalization
  - یا standardization

پارامتر مهم:

- K
- initialization

نویز را تشخیص نمی‌دهد

- هر نقطه‌ای باید به یک خوشه برود
- outlier می‌تواند مرکز خوشه را جابه‌جا کند

DBSCAN → «پیدا کردن نواحی پرتراکم»

فرض اصلی:

خوشه‌ها نواحی با **چگالی بالا** هستند

شکل خوشه مهم نیست (کروی، هلالی، مارپیچ...)

نتیجه:

برای خوشه‌های غیر کروی عالی است

می‌تواند شکل واقعی داده را کشف کند

- به‌شدت حساس به:
  - (epsilon (ε
  - min_samples

اگر مقیاس درست نباشد:

- ε معنی خود را از دست می‌دهد
- کل خوشه‌بندی خراب می‌شود

📌 scaling در DBSCAN **حیاتی** است

نویز را به‌صورت طبیعی تشخیص می‌دهد

- نقاط کم‌چگالی → noise
- برچسب -1

📌 یکی از بزرگ‌ترین مزیت‌های DBSCAN

نیازی به K ندارد

- تعداد خوشه‌ها به‌صورت خودکار از داده در می‌آید
- حتی ممکن است:
  - 0 خوشه
  - یا خوشه‌های کم/زیاد

Hierarchical → «ساختن ساختار درختی از داده‌ها»

فرض اصلی:

داده‌ها دارای **ساختار سلسله‌مراتبی** هستند

خوشه‌بندی را می‌شود در سطوح مختلف دید

📌 شکل خوشه:

به نوع **linkage** بستگی دارد

می‌تواند کروی یا غیرکروی باشد

حساس به **linkage**:

- single → زنجیره‌ای
- complete → فشرده
- average → متعادل
- ward → کروی‌تر

به مقیاس هم حساس است (بسته به linkage)

نویز را صریح تشخیص نمی‌دهد

- ولی در dendrogram:
  - نقاط دورافتاده دیرتر وصل می‌شوند
- می‌توان با برش مناسب اثر نویز را کم کرد

سؤال ۷ — non-convex بودن تابع هدف K-Means

non-convex بودن چه اثری روی بهینه‌سازی دارد؟

تابع هدف K-Means چی بود؟

مجموع مربع فاصله‌ی نقاط تا مراکز خوشه‌ها (WCSS)

وقتی:

- همه‌ی مراکز خوشه را **همزمان** در نظر می‌گیریم

این تابع:

- **چندین مینیمم محلی** دارد
- فقط **یک مینیمم سراسری** دارد (بهترین حالت ممکن)

چرا initialization های مختلف جواب‌های مختلف می‌دهند؟

زیرا نقطه‌ی شروع مسیر را تعیین می‌کند

مشکل این است که:

- K-Means فقط **پایین رفتن** را بلد است
- **بالا رفتن و دوباره پایین آمدن** بلد نیست

یعنی:

- اگر به یک مینیمم محلی رسید
- دیگر راه فرار ندارد

در عمل چه کار می‌کنیم که این مشکل کمتر شود؟

راهکار 1: اجرای چندباره الگوریتم (n_init)

رایج‌ترین و ساده‌ترین راه:

- K-Means را **چند بار** اجرا می‌کنیم
- هر بار با مراکز اولیه‌ی متفاوت
- بهترین نتیجه (کمترین WCSS) را انتخاب می‌کنیم

راهکار 2: استفاده از K-Means++

به‌جای انتخاب کاملاً تصادفی مراکز اولیه:

- مرکز اول تصادفی
- بقیه مراکز:
  - دور از هم
  - با احتمال بیشتر برای نقاط دورتر

📌 نتیجه:

- شروع بهتر
- افتادن کمتر در مینیمم‌های بد

در sklearn:

```
init="k-means++"
```

راهکار 3: نرمال‌سازی / Scaling داده

چون:

- K-Means به فاصله حساس است
- اگر یک ویژگی خیلی بزرگ باشد، همه چیز را خراب می‌کند

پس:

- StandardScaler
- MinMaxScaler

📌 این باعث می‌شود:

سطح تابع هدف «منطقی‌تر» شود

راهکار 4: انتخاب K مناسب

- K خیلی کم → خوشه‌بندی بد
- K خیلی زیاد → مینیمم‌های محلی زیاد

Elbow و Silhouette کمک می‌کنند:

- در ناحیه‌ی معقول K کار کنیم

✅ راهکار 5: چند الگوریتم یا چند Seed مختلف

در پروژه‌های جدی:

- چند بار اجرا
- با seed های مختلف
- نتایج مقایسه می‌شود