Clustering

Clustering groups similar items without any labels. A tiny shop-customer example, the

Introduction

In lesson ML.2 you met unsupervised learning: finding structure in data that has no labels. **Clustering** is its most useful job. Given a pile of items and no answers at all, clustering sorts them into groups — clusters — where items inside a group are similar to each other and different from items in other groups.

A shop owner is the perfect example. She has records of her customers — how much each spends per visit, and how often they visit — but nobody labelled them "bargain hunter" or "big spender". Clustering finds those groups for her. One group visits often and spends little; another visits rarely but spends a lot. She did not know these groups existed until the data showed them, and now she can treat each group differently: a loyalty card for the frequent visitors, a premium offer for the big spenders.

The most famous clustering method is **KMeans**. Its idea is simple enough to follow by hand, and it is the one you will meet in code below. But it comes with a catch you must understand honestly: you have to choose how many groups to look for, and the data will not tell you the "right" number. That choice is a judgement call, not a calculation.

One warning about KMeans that surprises beginners: distance is not fair when features live on very different scales. In Example 1, spend per visit runs from 10 to 100, while visits per month runs from 1 to 9. When KMeans measures "nearest centre", a difference of 50 in spending counts far more than a difference of 5 in visits — not because spending matters more, but simply because its numbers are bigger. The centre is pulled around mostly by spending, and visits barely get a vote.

The fix is to put features on a similar scale before clustering — a step called **scaling**. A common method subtracts the average and divides by the spread, so every feature ends up centred around zero with a similar range. After scaling, a one-step difference in visits and a one-step difference in spending count comparably, and the groups reflect both features. Whenever your features have very different ranges, scale first and cluster second; skipping this step is one of the most common reasons clustering gives odd groups.

(Roman Urdu: Clustering ka matlab hai baghair labels ke cheezon ko groups mein baantna — jo aik jaisi hain woh aik group mein, jo mukhtalif hain woh doosre mein.)

Key points

- **Clustering is unsupervised.** There are no labels, no right answers to check against — only groups that the data suggests.
- **A cluster is a group of similar items.** Similar inside the group, different from other groups. What "similar" means depends on the features you choose.

- **KMeans works in three repeating steps:** (1) pick k starting centres, (2) assign every point to its nearest centre, (3) move each centre to the middle of its points. Repeat steps 2–3 until the centres stop moving.
- **k is the number of clusters, and you choose it.** The method cannot choose for you. Trying several values of k and judging the results is normal practice.
- **Choosing k is a judgement call.** Too few clusters lump different customers together; too many split one real group into meaningless slivers. Look at the groups and ask: do these make sense for my problem?
- **Features decide the groups.** Cluster the same customers by spending and you get spending groups; cluster them by age and you get age groups. The features you pick shape the answer.
- **Clusters are suggestions, not facts.** Always check whether a grouping is useful and sensible before acting on it — the algorithm will happily group random data too.
- **Scale features before clustering when their ranges differ.** Without scaling, the feature with the biggest numbers dominates the distance, and the other features barely count.
- **The elbow method helps choose k.** Run KMeans for k = 1, 2, 3, 4 and plot how much the points spread around their centres; the "elbow" — where adding another cluster stops helping much — is a reasonable k to try first.
- **Distance is the definition of "similar" here.** Two items are similar when the distance between their feature values is small; change the features, and you change what counts as similar.

Examples

Example 1 — Tiny shop-customer example, by hand

Eight customers, described by visits per month and spend per visit:

| Customer | Visits/month | Spend/visit | |---|---|---| | A | 8 | 10 | | B | 9 | 12 | | C | 7 | 11 | | D | 1 | 90 | | E | 2 | 100 | | F | 1 | 80 | | G | 4 | 45 | | H | 5 | 50 |

By eye, three groups appear: A, B, C (often, cheap); D, E, F (rare, expensive); G, H (in the middle). That is clustering done by a human. KMeans does the same job by maths: it places centres, assigns each customer to the nearest centre, moves the centres to the middle of their groups, and repeats until nothing moves.

Example 2 — KMeans in code (scikit-learn)

This example needs scikit-learn installed first: `pip install scikit-learn`. It uses only the standard `KMeans` model on the tiny table above:

```
from sklearn.cluster import KMeans

# features: [visits per month, spend per visit]
customers = [[8, 10], [9, 12], [7, 11], [1, 90],
             [2, 100], [1, 80], [4, 45], [5, 50]]

model = KMeans(n_clusters=3, random_state=42, n_init=10)
groups = model.fit_predict(customers)
print("Group for each customer:", groups)
print("Cluster centres:", model.cluster_centers_)
```

What the result means: `n_clusters=3` tells KMeans to find 3 groups, and `fit_predict` both trains the model and returns each customer's group number. The centres are the middle points of the three groups — roughly the "often and cheap" centre, the "rare and expensive" centre, and the middle centre. Group numbers (0, 1, 2) are just labels the method made up; they have no meaning until you look at who is in each group.

Example 3 — Choosing k badly

Run the same customers with `n_clusters=2` and the middle customers G and H get forced into one of the two extreme groups — the "middle" type of customer disappears from the picture. Run it with `n_clusters=5` and the three "often and cheap" customers split into tiny groups that mean nothing for the shop. Both runs "work" technically. Only a human looking at the groups can say which k tells a useful story. That is why choosing k is judgement, not maths.

Example 4 — Worked example: nearest centre, calculated by hand

Take customer G from Example 1: 4 visits per month, spend 45 per visit. Suppose the three centres after one round of KMeans are: cheap-and-often centre (8, 11), expensive-and-rare centre (about 1.3, 90), and middle centre (4.5, 47.5). Measure the straight-line distance from G to each:

- To (8, 11): the gaps are 4 in visits and 34 in spending, distance about 34.2.
- To (1.3, 90): the gaps are 2.7 in visits and 45 in spending, distance about 45.1.
- To (4.5, 47.5): the gaps are 0.5 in visits and 2.5 in spending, distance about 2.5.

G joins the middle cluster, because 2.5 is by far the smallest distance. That is the whole assignment step: measure to every centre, join the nearest. Then the centres move to the middle of their new members, and the measuring repeats. Notice also how spending dominates the first two distances — the reason scaling (see the introduction above) matters when features have different ranges.

Example 5 — Worked example: trying several k values, in code (scikit-learn)

This example needs scikit-learn installed first: `pip install scikit-learn`. It runs KMeans with k = 2, 3 and 4 on the same customers, and prints a spread measure (called inertia: the total squared distance of points from their centres) for each:

```
from sklearn.cluster import KMeans

customers = [[8, 10], [9, 12], [7, 11], [1, 90],
             [2, 100], [1, 80], [4, 45], [5, 50]]

for k in [2, 3, 4]:
    model = KMeans(n_clusters=k, random_state=42, n_init=10)
    model.fit(customers)
    print("k =", k, "inertia =", round(model.inertia_, 1))
```

Run it and watch the pattern: inertia always falls as k grows — more centres always fit tighter, just as more groups always "explain" more. So you cannot pick k by inertia alone; a falling number is guaranteed. Instead, look for the elbow: the k where the fall slows down sharply, and then look at the actual groups for that k and ask whether they tell a useful story for the shop. The number guides you; the story decides.

Real-world case — How a delivery company plans its routes

A delivery company has hundreds of drop-off addresses every morning and no labels for them — just locations. It clusters the addresses into groups of nearby stops, and each driver gets one cluster as their route for the day. That is clustering doing real work: no right answers existed, but "nearby stops together" is a useful grouping. The company must still judge the result: a cluster split by a river with no nearby bridge looks close on a map but is a terrible route in practice. So a human checks the routes, moves a few stops by hand, and only then sends drivers out. The algorithm proposes; the human, who knows the roads, disposes. That partnership is the honest picture of clustering in production.

Quick recap

- Clustering groups similar items with no labels — unsupervised learning.
- KMeans: pick k centres, assign points to the nearest centre, move centres to the middle, repeat.
- You choose k. Too small merges real groups; too large splits them pointlessly.
- Your features decide what "similar" means, and therefore what groups you get.
- Clusters are suggestions to check, not facts to blindly follow.

Common mistakes

1. **Treating cluster numbers as real labels** — group 0 is not "better" than group 1; the numbers are arbitrary. Fix: describe each group by its actual members before using it.
2. **Picking k without looking at the groups** — any k runs without error, even a silly one. Fix: try several values of k and judge which grouping makes sense for your problem.
3. **Clustering on the wrong features** — clustering customers by name spelling gives nonsense groups. Fix: choose features that matter for the decision you actually want to make.

1. **Clustering without scaling, then wondering why one feature ruled** — a feature measured in thousands will swamp one measured in single digits. Fix: scale all features to a similar range before running KMeans.
2. **Reporting clusters without describing them** — "group 2 grew 10%" means nothing until you say who group 2 is. Fix: always describe each cluster by its typical member: visits, spending, and size.

Practice

1. Using the table in Example 1, cluster the same 8 customers by hand into 2 groups instead of 3. Which customers move, and what story does the 2-group version tell?
2. Run the code in Example 2 with `n_clusters=2` and then `n_clusters=4`. Write one line for each about which grouping looks more useful for the shop owner.
3. Think of a grouping problem at your school or home (books, clothes, expenses). Write down the features you would use and how many groups you would try first.

1. Using the distances in Example 4, work out by the same method which cluster customer H (5 visits, 50 spend) joins. Show your three distances.
2. Take the 8 customers in Example 1 and cluster them by only one feature — spend per visit. How do the groups change, and what does that teach you about feature choice?
3. Run the code in Example 5, then add a ninth customer (9 visits, 95 spend — often AND expensive). Does any k value's grouping change? Write one line about what that customer does to the story.

Practice quiz

1. What is clustering?

- Sorting labelled data
- Grouping similar rows without any labels
- Predicting numbers
- Cleaning data

Answer: Grouping similar rows without any labels

2. What is the basic idea of KMeans?

- Pick k centres, assign points to the nearest centre, move centres, repeat
- Draw k charts
- Sort the data k times
- Ask k questions

Answer: Draw k charts

3. How do you choose k, the number of clusters?

- It is fixed at 3

- It is a judgement call — try a few values and see which grouping makes sense
- The data tells you exactly
- Always use 10

Answer: It is a judgement call — try a few values and see which grouping makes sense

---

Watch the video lesson: https://youtu.be/5fJv1SnVVeg

Read online: https://scopetreasure.com/notes/ml-clustering · More free tools and notes at https://scopetreasure.com